2021 Small Object AI Challenge

TRACK 2. 수식/도형/낙서/기호/OCR 데이터 중
인쇄체 인식 AI 모델 개발

NSML Baseline Code Guide

Overview

- 목적: 수식 이미지로부터 TeX 표현식을 예측하는 모델 개발

ex1 이미지: $= x^4 + 4x^3 - 35x^2 - 78x + 360$

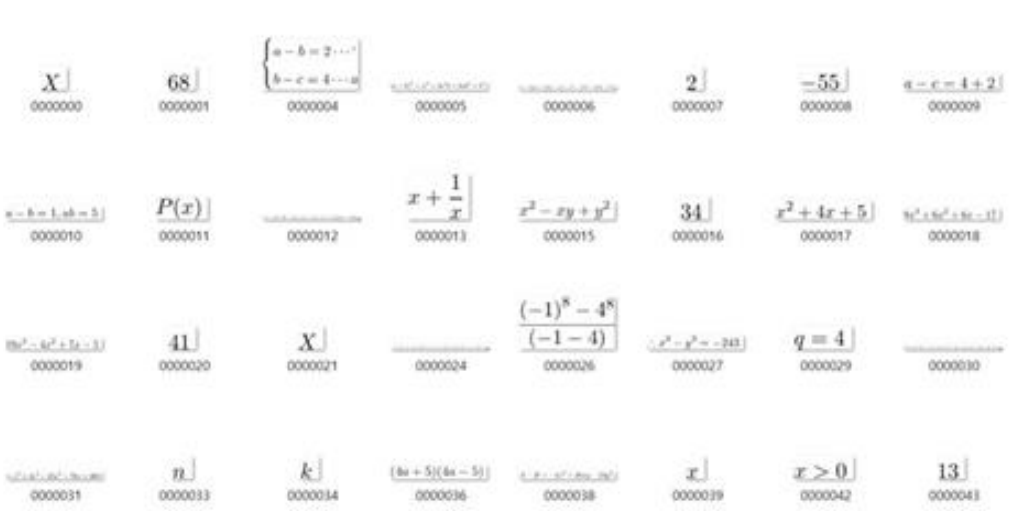
표현식: ={x}^{4}+4{x}^{3}-35{x}^{2}-78x+360

ex2 이미지: $\Rightarrow x > -12 \cdots$

표현식: ₩Rightarrow x > -12 ₩cdots

Data info

- 학습용 이미지 및 라벨 예시




```
X
68
#begin{cases} a-b=2 #cdots ##b-c=4 #cdots #end{cases}
{a+b}^{3}=a^{3}+3a^{2}b+3ab^{2}+b^{3}
(x-4)(x-9)(x-2)=x^{3}-15x^{2}+62x-72
2
-55
a-c=4+2
a-b=1, ab=5
P(x)
7a #times -9b+6b #times -6a= #{ 7 #times (-9)+6 #times (-6) #} ab=-99ab
x=#frac{1}{x}
x^{2}-xy+y^{2}
34
x^{2}+4x+5
9x^{3}+6x^{2}+6x-17
19x^{3}-4x^{2}+7x-5
41
X
 #therefore A+8-C=(8+3)x^{3}+(-2+3+3)x^{2}+(2+1)x+(-9-6+5)
#frac{}{-1}^{8}-4^{8}{}{-1-4}
 #therefore (x)^{3}-(y)^{3}=-243
q=4
{2a+3b+8c}^{2}={-2a+3b+8c}^{2}+{2a-3b+8c}^{2}+{2a+3b-8c}^{2}
=(x)^{4}+40x^{3}-35x^{2}-78x+360
```

Data info

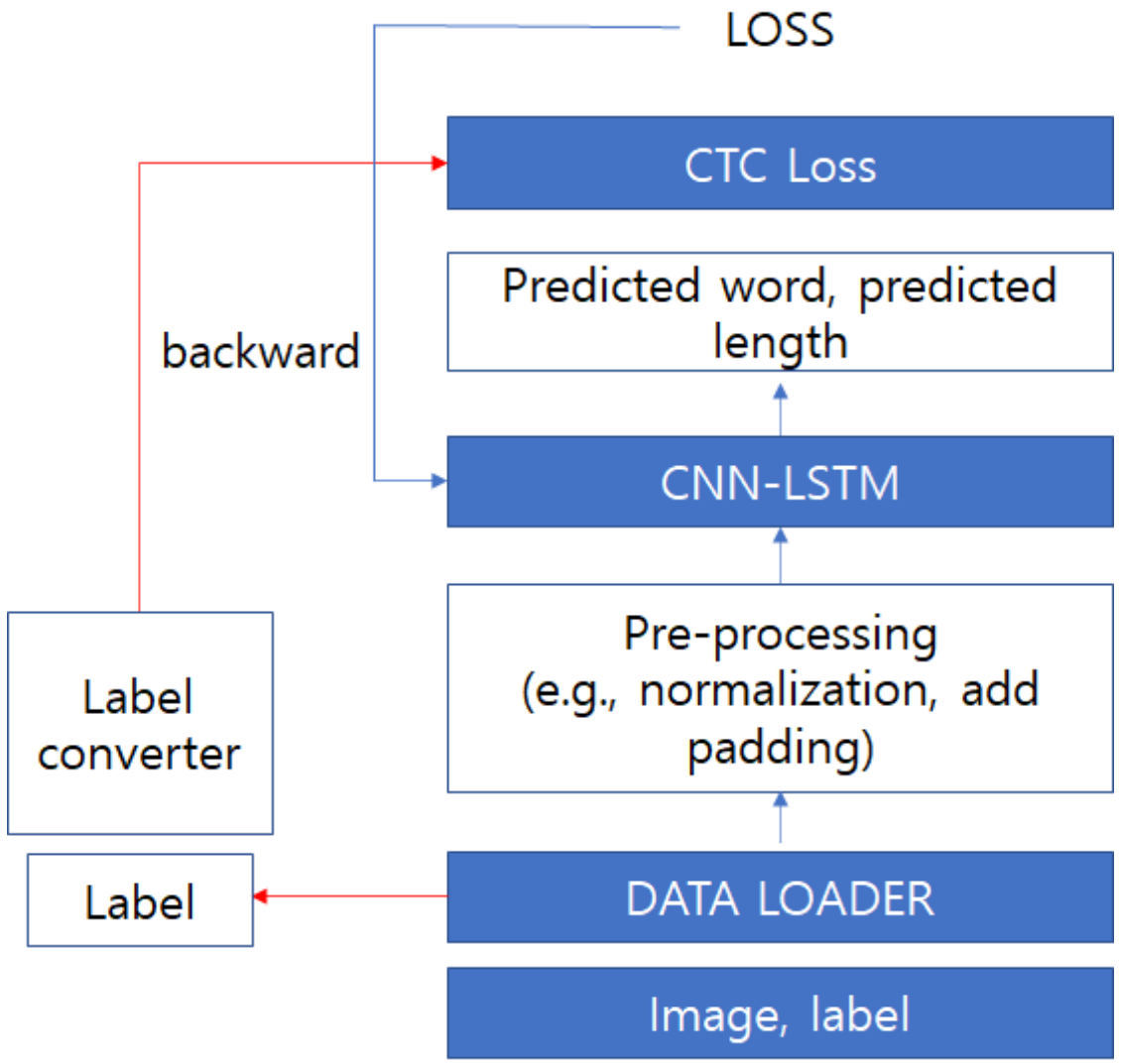
- 데이터 형식

  - train/trian_data 폴더 안에 모든 학습용 이미지가 있으며,
    각 이미지의 표현식(라벨)은 train/train_label 파일의 라인에 해당.
      예) 0000000.png 의 라벨은 train_label 파일의 0번째 줄에 해당하는 X
          0000001.png 의 라벨은 train_label 파일의 1번째 줄에 해당하는 68
  - 이미지 heigh는 32 픽셀로 동일하나 이미지 width는 이미지별로 상이함.

- 데이터 규모

  - 약 50,000장의 학습용 이미지

Baseline Process



main.py

```
import nsml
from nsml import DATASET_PATH
```

①

```
def bind_model(model):
    def load(filename, **kwargs):
        state = torch.load(os.path.join(filename, 'model.pt'))
        model.load_state_dict(state['model'])
        print('Model loaded')

    def save(filename, **kwargs):
        state = {
            'model': model.state_dict(),
        }
        torch.save(state, os.path.join(filename, 'model.pt'))
```

②

- ① 데이터셋 경로입니다.

학습용 이미지의 경로는

DATASET_PATH + "train/train_data" 입니다.

- ② 학습 된 모델을 저장하고 불러오는 함수입니다.

nsml.save(epoch) 를 통해 매 epoch마다 모델의 파라미터를 저장하실 수 있습니다.

저장 된 모델은 nsml submit 시 불러와 사용하실 수 있습니다.

main.py

```python
def infer(data_path, imgdir):
    config = {
        'path':data_path,
        'imgdir':imgdir
    }
    data = CustomDataset_infer(config)
    loader = torch.utils.data.DataLoader(data,
        batch_size=8,
        collate_fn=SynthCollator_infer())

    net = model
    device = 'cuda' if torch.cuda.is_available() else 'cpu'
    net.to(device)
    net.eval()
    math_list = []
    with open('train_label', 'r', encoding='utf-8') as f:
        lines = f.readlines()
        math_list.extend(lines)
    math_list = ''.join(math_list)
    alphabet = ''.join(set(math_list))
    alphabet = ''.join(sorted(alphabet))                    ①

    converter = OCRLabelConverter(alphabet)
    predictions = []
    for iteration, batch in enumerate(tqdm(loader)):
        input_ = batch['img'].to(device)
        logits = net(input_).transpose(1, 0)
        logits = torch.nn.functional.log_softmax(logits, 2)
        logits = logits.contiguous().cpu()
        T, B, H = logits.size()
        pred_sizes = torch.LongTensor([T for i in range(B)])
        probs, pos = logits.max(2)
        pos = pos.transpose(1, 0).contiguous().view(-1)
        sim_preds = converter.decode(pos.data, pred_sizes.data, raw=False)
        predictions.extend(sim_preds)

    return predictions
```

- nsml submit 시 사용되는 inference 함수

모델에 맞추어 자유롭게 작성하시면 되나, Output 형식은 ['prediction1', 'prediction2', ...]를 지켜 주셔야 합니다. 데이터 경로의 테스트 이미지로부터 예측한 수식의 값을 차례대로 넣어 리스트 형식으로 리턴 해 주세요.

예) predictions = ['₩Rightarrow x >', '={x}^{4}+4{x}^{3}', ...]

① train_label 파일로부터 사용된 고유한 문자를 추출합니다. 학습용 데이터의 경우 표현식에 사용된 고유한 문자를 바탕으로 학습 시 클래스로 사용합니다.

main.py

```python
if __name__ == "__main__":

    # torch.backends.cudnn.deterministic = True
    # torch.manual_seed(71)
    # np.random.seed(71)
    # random.seed(71)

    parser = ArgumentParser()
    if nsml.IS_ON_NSML:
        parser.add_argument("--path", type=str, default=DATASET_PATH)
        parser.add_argument("--train_label_path", type=str, default=DATASET_PATH + '/tr
    else:
        parser.add_argument("--path", type=str, default='./data')
        parser.add_argument("--train_label_path", type=str, default='./data/train/train
    parser.add_argument("--name", type=str, default='exp1')
    parser.add_argument("--imgdir", type=str, default='train/train_data')
    parser.add_argument("--lr", type=float, default=0.001)
    parser.add_argument("--imgH", type=int, default=32)
    parser.add_argument("--nHidden", type=int, default=256)
    parser.add_argument("--nChannels", type=int, default=1)
    parser.add_argument("--batch_size", type=int, default=8)
    parser.add_argument("--epoch", type=int, default=0)
    parser.add_argument("--epochs", type=int, default=4)

##################################################################
    # DONOTCHANGE: They are reserved for nsml
    parser.add_argument("--mode", type=str, default='train', help='submit일때 해당값이 t
    parser.add_argument("--pause", type=int, default=0)
##################################################################

    args = parser.parse_args()

    if args.mode == 'test':
        args.imgdir = 'test/test_data'

    if nsml.IS_ON_NSML:
        train_label_path = DATASET_PATH + '/train/train_label'
    else:
        train_label_path = './data/train/train_label'
```

① ②

- ① argparse.ArgumentParser() 를 통해 parser를 생성해 add_argument를 이용해 입력 받고자 하는 인자의 조건을 설정합니다. parser.parse_arg()를 통해 인자들을 파싱하여 args에 저장합니다.

- ② nsml submit 시 필요한 인자입니다. 수정하지 말아 주세요.

dataset.py

①

```
class CustomDataset(Dataset):
    def __init__(self, opt):
        super(CustomDataset, self).__init__()
        self.path = os.path.join(opt.path, opt.imgdir)
        self.images = os.listdir(self.path)
        self.nSamples = len(self.images)
        f = lambda x: os.path.join(self.path, x)
        self.imagepaths = list(map(f, self.images))
        transform_list =  [transforms.Grayscale(1),
                           transforms.ToTensor(),
                           transforms.Normalize((0.5,), (0.5,))]
        self.transform = transforms.Compose(transform_list)
        self.collate_fn = SynthCollator()
        self.train_label_path = opt.train_label_path

    def __len__(self):
        return self.nSamples

    def __getitem__(self, index):
        assert index <= len(self), 'index range error'
        imagepath = self.imagepaths[index]
        imagefile = os.path.basename(imagepath)
        img = Image.open(imagepath)
        if self.transform is not None:
            img = self.transform(img)
        item = {'img': img, 'idx':index}
        math_list = []
        image_idx = int(imagefile.split('.')[0])
        with open(self.train_label_path, 'r', encoding='utf-8') as f:
            lines = f.readlines()
            for i in range(len(lines)):
                lines[i] = lines[i][:-1]
            math_list.extend(lines)
        item['label'] = math_list[image_idx]
        return item
```

②

```
class SynthCollator(object):

    def __call__(self, batch):
        width = [item['img'].shape[2] for item in batch]
        indexes = [item['idx'] for item in batch]
        imgs = torch.ones([len(batch), batch[0]['img'].shape[0], batch[0]['img'].shape[1],
                           max(width)], dtype=torch.float32)
        for idx, item in enumerate(batch):
            try:
                imgs[idx, :, :, 0:item['img'].shape[2]] = item['img']
            except:
                print(imgs.shape)
        item = {'img': imgs, 'idx':indexes}
        if 'label' in batch[0].keys():
            labels = [item['label'] for item in batch]
            item['label'] = labels
        return item
```

- ① Custom data loader입니다. 이미지 경로에서 이미지를 , label 파일로부터 label을 읽습니다.
- ② 데이터의 height는 32로 동일하지만, width는 이미지별로 다르므로, batch내에서 max(width) 를 기준으로 패딩을 넣어줍니다.

main.py

```
data = CustomDataset(args)
args.collate_fn = SynthCollator()
train_split = int(0.9*len(data))
val_split = len(data) - train_split
args.data_train, args.data_val = random_split(data, (train_split, val_
print('Traininig Data Size:{}\nVal Data Size:{}'.format(
    len(args.data_train), len(args.data_val)))
```

①

```
math_list = []
with open('train_label', 'r', encoding='utf-8') as f:
    lines = f.readlines()
    math_list.extend(lines)
math_list = ''.join(math_list)
alphabet = ''.join(set(math_list))
alphabet = ''.join(sorted(alphabet))
```

②

```
args.alphabet = alphabet
args.nClasses = len(args.alphabet)
model = CRNN(args)
args.criterion = CustomCTCLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=args.lr)
learner = Learner(model, optimizer)

bind_model(model)
if args.pause:
    nsml.paused(scope=locals())

learner.fit(args)
```

③

④

- ① 학습데이터를 학습, 평가데이터로 나눕니다.
- ② 학습 라벨로부터 사용된 문자를 추출합니다. Baseline 모델의 경우 추출한 유니크한 문자를 모델 학습시에 분류 클래스로 사용합니다.
- ③ CNN-LSTM 기반의 모델과 torch.nn.CTCLoss를 사용합니다.
- ④ 메인 학습 함수

성능 평가 지표

- Word accuracy (exact match)

- 전체 테스트 데이터 중 맞춘 단어 비율

Ex) ground truth : ['aaa', 'bbb', 'ccc', 'ddd', 'eeeee']
prediction : ['aaa', 'b', 'abc', 'ddd', 'eee']

- word accuracy = 2/5

TRACK 2. 수식/도형/낙서/기호/OCR 데이터 중 인쇄체 인식 AI 모델 개발

NSML Baseline Code Guide