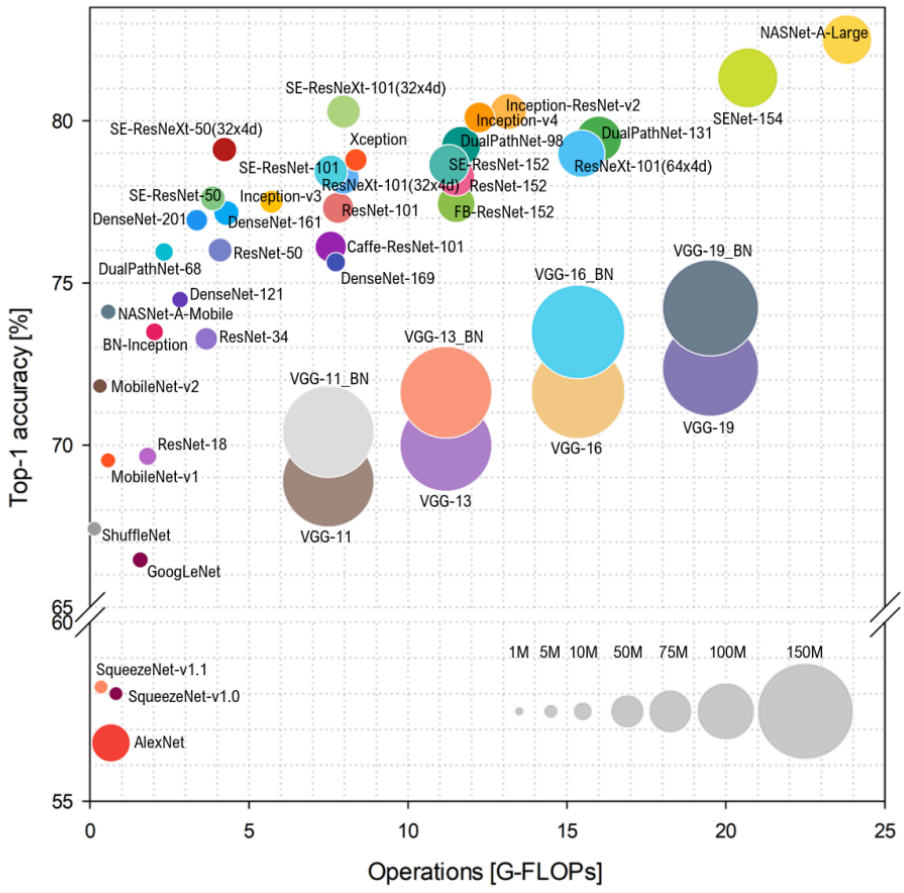
BIG DATA 경진대회

김태인, 김동근

1. Competition_No1

모델 선정
논문 “Searching for MobileNetV3, Benchmark Analysis of Representative Deep Neural Network Architectures, Do ImageNet Classifiers Generalize to ImageNet?” 및 Keras Applications, google Ai, Naver Deview 2019를 참고하여 모델을 선정하였음. 기존 모델의 이미지에 대한 성능과 파라미터수를 고려하여 Transfer Learning 에 적합한 모델을 선정하였음.



[그림 1] Top-1 accuracy(%) vs Operations[G-FLOPs]

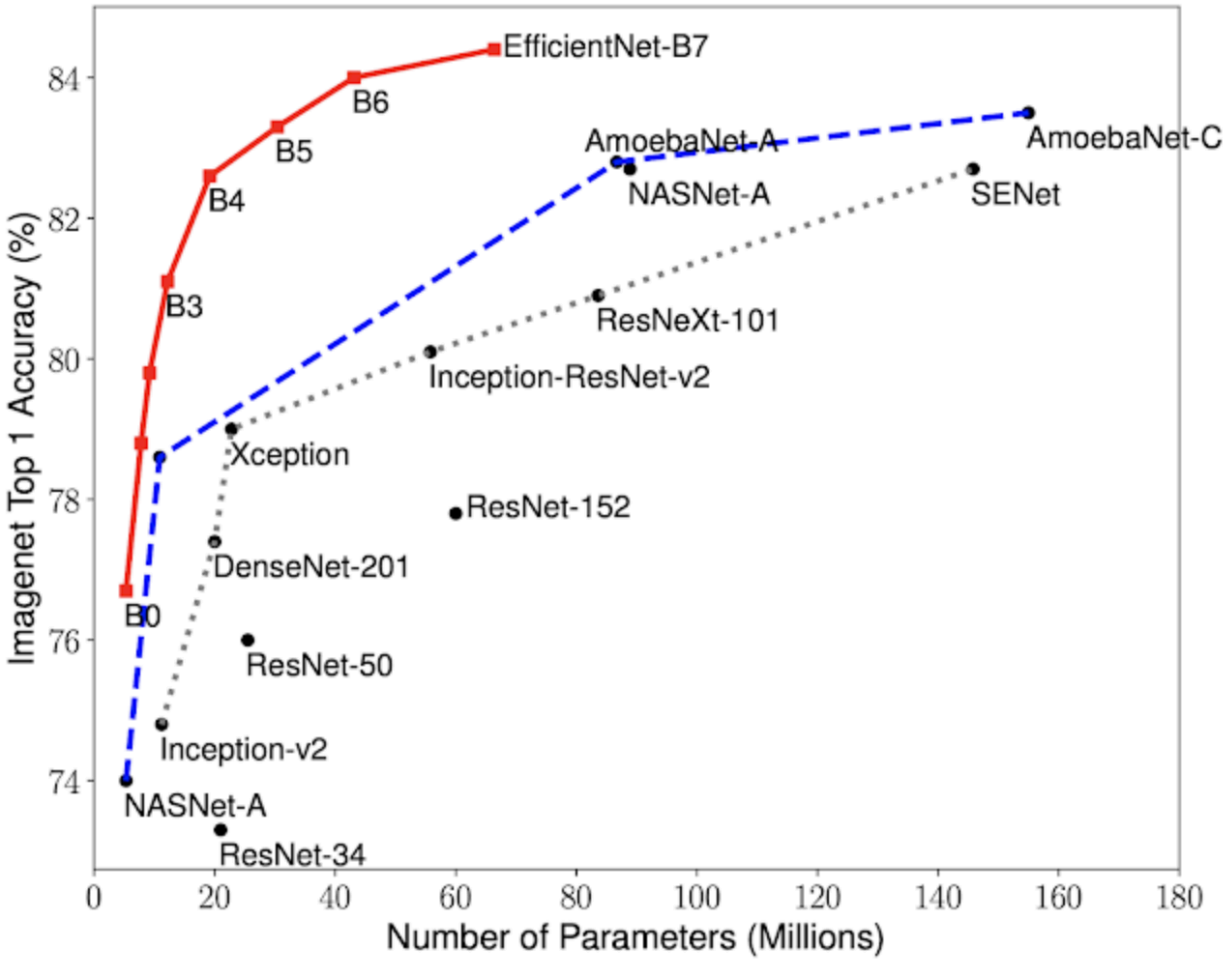
Keras Applications

In particular, EfficientNet-B7 achieves new state-of-the-art 84.4% top-1 / 97.1% top-5 accuracy, while being 8.4x smaller than the best existing CNN.
- google ai blog -

Model	Size(MB)	Top-1 Accuracy(%)	Top-5 Accuracy(%)	Parameters	Depth	Time(ms) per inference step(CPU)	Time(ms) per inference step(GPU)
MobileNet V2	14	0.713	0.901	3,538,984	88	25.90	3.83

Model	Size(MB)	Top-1 Accuracy(%)	Top-5 Accuracy(%)	Parameters	Depth	Time(ms) per inference step(CPU)	Time(ms) per inference step(GPU)
EfficientNetB0	29	-	-	5,330,571	-	46.00	4.91
ResNet152	232	0.780	0.942	60,380,648	-	107.50	6.64
NasNetLarge	343	0.825	0.960	88,949,818	-	344.51	19.96
EfficientNetB7	256	84.4	97.1	66,658,687	-	308.33	15.12

[표1] Keras Applications 모델 성능



[그림 2] accuracy vs. computational complexity

모델 설명

1. 개발 과정

preprocess.py 파일에서 적합한 모델에 대해 실험을 진행하였습니다. ImageDataGenerator를 통해 이미지 데이터의 부족 현상을 해결하기 위해 rotation_rage, width_shift_range, height_shift_range, shear_range, zoom_range, brightness_range의 속성을 주어 train 셋을 변화시켰습니다.

```
ImageDataGenerator(rescale=1. / 255,
                   rotation_range=20, width_shift_range=0.2,
                   height_shift_range=0.2, shear_range=0.2, zoom_range=0.2,
                   brightness_range=[0.8, 1.2],
                   validation_split=0.2)
```

[그림 3] ImageDataGenerator 변수 값

flow_from_directory를 통해 train 셋을 불러온 후 0.2의 비율을 validation으로 설정하고 rescale 과정만 진행하였습니다.
전이학습의 모델 결과를 확인하기 위해 preprocess.py 파일을 통해 epoch 5번 결과의 val_accuracy와 시간을 측정하였습니다.

epoch(5)	val_accuracy(0.2) (%)	time(sec)	Input size	Batch size
MobileNetV2	0.8184	0:10:15.913060	224 x 224	32
EfficientNetB0	0.5042	0:10:22.713614	224 x 224	32
ResNet152	0.8322	0:20:51.688205	224 x 224	32
NasNetLarge	0.9228	1:50:02.982123	331 x 331	16

[표 2] 전이 학습(epoch=5)의 결과

추가 진행
EfficientNetB7은 EfficientNetB0의 batch size를 조정하는 과정에서 1보다 작은 값을 줄 수 없었기에 전이학습을 진행하지 못하였습니다. 모델 테스트 결과 NasNet의 validation accuracy가 높게 나왔지만 실제 overfitting 된 결과를 보여주었습니다. 파라미터의 개수가 너무 많은것에 비해 이미지 데이터의 수가 부족하여 생긴 상황이라 판단하였습니다. 이에 ResNet152를 이용하여 모델을 구성하게 되었습니다. 전이학습을 통해 얻은 결과와 파인 튜닝 과정을 통해 GlobalAveragePollling2D 과정과 오버피팅을 방지하기위해 Dropout 0.25의 값을 주었습니다. 마지막으로 sigmoid 활성화 함수와 binary cross entropy의 과정과 Adam learning rate를 조정하는 과정을 통해 모델을 학습하게 되었습니다.

추가적으로 이미지 데이터에서 화물 적재와 적재하지 않은 이미지 데이터의 불균형이 존재하였기에 0과 1번 파일의 갯수에 따른 class_weight를 주어 0.8 : 1.33의 비율로 진행되게 되었습니다. 데이터의 수가 변하면 class weight 또한 변할 수 있게 적용하였습니다.

Modelcheckpoint를 통해 학습 에폭당 val loss에 따라 h5파일에 저장하게 되었고 이를 test.py에서 테스트를 진행하였습니다.
EarlyStopping을 주어 Model의 val loss가 5회 이상 개선되지 않을 시 학습을 종료하도록 하였습니다.

Resnet152 epoch	train_loss	train_accuracy	val_loss	val_accuracy
1	0.3375	0.8340	0.9925	0.5091
2	0.1377	0.9477	0.8498	0.6049
3	0.0935	0.9658	0.4991	0.7865
4	0.0707	0.9757	0.5040	0.8577
5	0.0534	0.9815	0.6669	0.7829
6	0.0443	0.9846	0.6389	0.8201

[표3] Resnet 152 (dense, dropout 2개 이상, activation relu 추가)

Resnet152 epoch	train_loss	train_accuracy	val_loss	val_accuracy
1	0.2858	0.8640	1.1780	0.5111
2	0.1149	0.9521	0.6110	0.7117
3	0.0678	0.9751	0.5234	0.8132
4	0.0565	0.9781	0.4632	0.8273
5	0.407	0.9849	0.5563	0.8528
6	0.0349	0.9875	0.4640	0.8552
7	0.0269	0.9904	0.4989	0.8463
8	0.0325	0.9886	0.5393	0.8734

[표 4] Resnet 152 (dense 16, activation relu 추가, lr = 2e-5)

Resnet152 epoch	train_loss	train_accuracy	val_loss	val_accuracy
1	0.3017	0.8591	0.7562	0.5410
2	0.1236	0.9510	0.7356	0.5892
3	0.0730	0.9740	0.4577	0.7901
4	0.0497	0.9817	0.4046	0.8350
5	0.0391	0.9845	0.4752	0.8532
6	0.0376	0.9846	0.4236	0.8670
7	0.0281	0.9892	0.5389	0.8641
8	0.0237	0.9911	0.5318	0.8524
9	0.0246	0.9909	0.4345	0.8674

[표 5] Resnet 152 (dense 16, activation relu 제거, lr = 2e-5)

여러번의 실험 과정을 통해 가장 성능이 좋게 나온 전이 학습 모델 과정을 찾아내었고 val loss에 따라 이를 현재 디렉터리에 h5 파일로 저장하여 test.py에서 테스트를 진행하게 되었습니다.

2. Demo 실행(테스트용)

preprocess.py에서 h5파일로 저장한 모델을 test.py 파일에서 불어왔습니다. 이를 ImageDataGenerator 및 flow_from_directory를 통해 test 디렉터리 이미지 파일을 rescale에 대해 진행 후 predict를 진행하였습니다. 이 후 csv 파일로 data frame을 test 디렉터리에 저장을 하도록 진행하였습니다.

최종 테스트의 결과입니다.

	Image list	GT	Prediction	accuracy
0	0\empty_V1.jpg	0	[0]	TRUE
1	0\empty_V10.jpg	0	[1]	FALSE
2	0\empty_V2.jpg	0	[0]	TRUE
3	0\empty_V3.jpg	0	[0]	TRUE
4	0\empty_V4.jpg	0	[0]	TRUE
5	0\empty_V5.jpg	0	[0]	TRUE
6	0\empty_V6.jpg	0	[1]	FALSE
7	0\empty_V7.jpg	0	[1]	FALSE
8	0\empty_V8.jpg	0	[0]	TRUE
9	0\empty_V9.jpg	0	[0]	TRUE
10	1\loaded_V1.jpg	1	[1]	TRUE
11	1\loaded_V10.jpg	1	[1]	TRUE
12	1\loaded_V2.jpg	1	[1]	TRUE
13	1\loaded_V3.jpg	1	[1]	TRUE
14	1\loaded_V4.jpg	1	[1]	TRUE
15	1\loaded_V5.jpg	1	[1]	TRUE
16	1\loaded_V6.jpg	1	[1]	TRUE
17	1\loaded_V7.jpg	1	[1]	TRUE
18	1\loaded_V8.jpg	1	[1]	TRUE
19	1\loaded_V9.jpg	1	[1]	TRUE
20	Average Accuracy	0.85		
21	Inference Speed(ms)	300.6310701		

[그림4] 학습 결과

20개의 test 데이터셋에 대해 85%의 정확도를 가지게 되었습니다.

3. 모델 학습

다음의 명령어를 터미널 창에 입력하면 학습이 진행됩니다

Window

python preprocess.py

python test.py

2. Competition_No2

모델 선정

처음 ocr 작업을 진행할 때 파이썬 내부에서 사용할 수 있는 Tesseract, OpenCV, EasyOCR 라이브러리를 활용하하려 했으나 성능이 많이 떨어져서, 효율적으로 학습을 진행할 수 있는 방법을 생각하다 이미 학습되어 있는 이미지 처리 딥러닝 모델을 가져와서 custom data에 맞게 추가학습시키는 방법을 생각하였습니다.

논문 “What Is Wrong With Scene Text Recognition Model Comparisons? Dataset and Model Analysis”. Jeonghun Baek and Geewook Kim and Junyeop Lee and Sungrae Park and Dongyoon Han and Sangdoo Yun and Seong Joon Oh and Hwalsuk L, year={2019}. ICCV 을 참고해서 naver 의 ocr pretrained 모델을 가져와서 custom data에 맞게 fine tuning 하는 방식을 선정했습니다. 이 모델은 ICDAR2013 focused scene text, ICDAR2019 ArT 에서 1등의 성능. ICDAR2017 COCO-Text, ICDAR2019 ReCTS (task1)에서 3등의 성능을 기록하였습니다.

What Is Wrong With Scene Text Recognition Model Comparisons? Dataset and Model Analysis

Jeonghun Baek[1] Geewook Kim[2*] Junyeop Lee[1] Sungrae Park[1]
Dongyoon Han[1] Sangdoo Yun[1] Seong Joon Oh[1] Hwalsuk Lee[1†]
[1]Clova AI Research, NAVER/LINE Corp. [2]Kyoto University
{jh.baek, junyeop.lee, sungrae.park, dongyoon.han, sangdoo.yun, hwalsuk.lee}@navercorp.com
geewook@sys.i.kyoto-u.ac.jp coallaoh@linecorp.com

Abstract

Many new proposals for scene text recognition (STR) models have been introduced in recent years. While each claim to have pushed the boundary of the technology, a holistic and fair comparison has been largely missing in the field due to the inconsistent choices of training and evaluation datasets. This paper addresses this difficulty with three major contributions. First, we examine the inconsistencies of training and evaluation datasets, and the performance

18, 28, 30, 4, 17, 5, 2, 3, 19] have proposed multi-stage pipelines, where each stage is a deep neural network addressing a specific challenge. For example, Shi *et al.* [24] have suggested using a recurrent neural network to address the varying number of characters in a given input, and a connectionist temporal classification loss [6] to identify the number of characters. Shi *et al.* [25] have proposed a transformation module that normalizes the input into a straight text image to reduce the representational burden for downstream modules to handle curved texts.

[그림 5] 논문 자료

모델 설명

1. 추가적인 환경 설치

저는 Anaconda 환경에서 설치를 진행하였습니다. 환경설치와 필요한 라이브러리에 관한 부분은 REAEME.txt와 requirements.txt에 정리하였습니다. CUDA 11.2, Python 3.7을 사용하였습니다

2. Demo 실행 (테스트용)

여러 pre-trained model 중에 가장 성능이 높은 모델의 체크포인트(TPS-ResNet-BiLSTM-Attn-case-sensitive.pth)를 다운받고, deep-text-recognition-benchmark 폴더 안에 넣기

*case-sensitive는 숫자와 알파벳뿐만이 아니라 특수기호도 포함해서 학습한 모델

링크 : https://drive.google.com/drive/folders/15WPsuPJDCzhp2SvYZLRj8mAlT3zmoAMW

```
# OCR 폴더로 이동 (ubuntu)
cd ~/deep-text-recognition-benchmark

# window
cd deep-text-recognition-benchmark

# demo.py 실행 (UBUNTU)
CUDA_VISIBLE_DEVICES=0 python3 demo.py --Transformation TPS --FeatureExtraction ResNet --SequenceMod
```

[그림 6] pertained model 추론 결과

데모를 실행하면 데모파일에 대해서 pretrained model이 추론한 결과를 보여줍니다. 정상적으로 실행되어 이제 finetuning 단계로 진행합니다.

3. 커스텀 데이터셋 제작

네이버 ocr text recognition 모델의 경우 일반 이미지 형식(jpg, png) 가 아닌 lmdb의 형식으로 학습을 하기 때문에 custom data를 lmdb의 형식으로 변환해주어야 합니다. 단 test 시에는 jpg, png 파일로도 추론이 가능합니다.

(3-1) deep-text-recognition-benchmark 폴더 안에 lmdb 파일을 담을 ‘predata' 폴더 생성

(3-2 predata 폴더 안에 'training', 'validation' 폴더 생성

(3-3) training, validation 폴더 안에 학습 및 검증용 이미지 넣기

Validation data는 train 데이터의 01, 02, 03 폴더에 존재하는 수의 비율과 유사하게 01, 02 폴더에서 300개, 03 폴더에서 30개를 사용하였습니다.

(3-4) data 폴더 안에 _gt.txt 파일 생성 후 라벨링하기

파일을 lmdb 의 형식으로 변환할 때 gt.txt 파일을 읽어들여서 이를 사용하여 이미지 파일을 불러 옵니다.

(3-5) 이미지와 라벨링 텍스트 파일을 lmdb 형태로 변환

```
# create train_lmdb file
python3 create_lmdb_dataset.py --inputPath data/training --gtFile data/train_gt.txt --outputPath data

# create valid_lmdb file
python3 create_lmdb_dataset.py --inputPath data/validation --gtFile data/valid_gt.txt --outputPath da
```

[그림 7] lmdb 형태 변환

4. ~/deep-text-recognition-benchmark/train.py 코드 수정

학습에 대한 파라미터를 custom data에 맞게 조정합니다.

if __name__ == '__main__': 문 아래에 있는 add_argument들 수정

--batch_size에서 default=192의 값을 자신의 그래픽카드 환경에 맞게 수정하기

batch_size를 64로 수정했습니다.

--select_data에서 default='MJ-ST'를 '/' 로 수정

OCR 학습데이터로 유명한 MJ, ST 데이터가 아니라, 커스텀 데이터셋을 사용하므로 그 경로에 맞게 수정하였습니다.

--batch_ratio에서 default='0.5-0.5'를 ['0.482', '0.482', '0.036']로 수정

커스텀 데이터셋 3종류를 사용하고 03폴더의 데이터가 특히 적으므로 01,02,03의 데이터 수에 근사하게 위와 같은 비율적용

--character에서 경우 특수문자도 학습해야 해서, Default = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLNMOPQRSTUVWXYZ' 으로 수정

--valInterval 인자

이 인자에 설정된 값만큼 해당 에포크에 검증 결과가 나옵니다. 처음에는 2000으로 설정되어 있어서, 2000에포크마다 결과가 나오는데, 저는 50 에포크마다 보고 싶어서 값을 50으로 설정했습니다.’

학습데이터를 나눠서 학습시킨 과정은 다음과 같습니다.

```
-------------------------------------------------------------------------------
dataset_root: predata/train
opt.select_data: ['01', '02', '03']
opt.batch_ratio: ['0.482', '0.482', '0.036']
-------------------------------------------------------------------------------
dataset_root:    predata/train dataset: 01
sub-directory:        /01          num samples: 3438
num total samples of 01: 3438 x 1.0 (total_data_usage_ratio) = 3438
num samples of 01 per batch: 64 x 0.482 (batch_ratio) = 31
-------------------------------------------------------------------------------
dataset_root:    predata/train dataset: 02
sub-directory:        /02          num samples: 3694
num total samples of 02: 3694 x 1.0 (total_data_usage_ratio) = 3694
num samples of 02 per batch: 64 x 0.482 (batch_ratio) = 31
-------------------------------------------------------------------------------
dataset_root:    predata/train dataset: 03
sub-directory:        /03          num samples: 270
num total samples of 03: 270 x 1.0 (total_data_usage_ratio) = 270
num samples of 03 per batch: 64 x 0.036 (batch_ratio) = 2
-------------------------------------------------------------------------------
Total_batch_size: 31+31+2 = 64
-------------------------------------------------------------------------------
```

[그림 8] 학습 과정

5. 모델 학습

다음의 명령어를 터미널 창에 입력하면 학습이 진행됩니다

Window

python train.py --train_data data_lmdb/training --valid_data data_lmdb/validation --Transformation TPS --FeatureExtraction ResNet --SequenceModeling BiLSTM --Prediction Attn

saved_models 폴더에서 log_train.txt, opt.txt 파일에서 학습 관련 로그를 확인할 수 있습니다!

최종 학습 후 sample test 결과는 다음과 같습니다. 최종적으로 best accuracy가 91.111인 모델을 체크포인트로 저장하였습니다. Epoch를 500까지만 한 이유는 크게 했을 때 valid loss가 500이후로 더 이상 낮아지지 않고 진동하였기 때문입니다.

```
-------------------------------------------------------------------------------
[500/500] Train loss: 0.04392, Valid loss: 0.08413, Elapsed_time: 184.42395
Current_accuracy : 90.952, Current_norm_ED  : 0.97
Best_accuracy    : 91.111, Best_norm_ED     : 0.97
-------------------------------------------------------------------------------
Ground Truth          | Prediction          | Confidence Score & T/F
-------------------------------------------------------------------------------
9012H15B17            | 9012H15B17          | 0.5084          True
983S11B41             | 983S11B41           | 0.9466 True
3801                  | 3801                | 0.9990      True
4303                  | 4303                | 0.9929      True
983S17T54             | 983S17T54           | 0.9740 True
-------------------------------------------------------------------------------
```

[그림 9] epoch 학습 과정

기타 hyperparameter 은 다음과 같습니다

```
---------- Options -------------
exp_name: TPS-ResNet-BiLSTM-Attn-Seed1111
train_data: predata/train
valid_data: predata/val
manualSeed: 1111
workers: 0
batch_size: 64
num_iter: 500
valInterval: 50
saved_model: models/TPS-ResNet-BiLSTM-Attn-case-sensitive.pth
FT: True
adam: False
lr: 1
beta1: 0.9
rho: 0.95
eps: 1e-08
grad_clip: 5
baiduCTC: False
select_data: ['01', '02', '03']
batch_ratio: ['0.482', '0.482', '0.036']
total_data_usage_ratio: 1.0
batch_max_length: 25
imgH: 32
imgW: 100
rgb: False
character: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&'()*+,-./:;<=>?@[₩]^_`{|}·
sensitive: True
PAD: False
data_filtering_off: True
Transformation: TPS
FeatureExtraction: ResNet
SequenceModeling: BiLSTM
Prediction: Attn
num_fiducial: 20
input_channel: 1
output_channel: 512
hidden_size: 256
num_gpu: 1
num_class: 96
----------------------------------------
```

[그림 10] hyperparameter

6. 코드 설명(아키텍쳐 분석)

6-1) train.py (모델 학습 코드)

(1) 최적화 함수 (Optimizer)

default는 Adadelta 입니다. 그리고 추가적인 인자를 넣어서 Adam으로 변경할 수 있습니다. 각 알고리즘의 자세한 설명은 구글링이 훨씬 이해하기 좋을 것입니다! 추가적인 인자를 넣는 방법은 2가지가 있습니다. 첫 번째는 위의 train.py 명령어 뒤에 --adam을 추가하는 것입니다. 두 번째는 코드 자체에서 optimizer를 adam으로 설정하는 것입니다.

(2) Loss 함수

Loss 함수는 Prediction 모듈에 따라서 달라집니다. 먼저 Prediction 모듈을 CTC(Connectionist Temporal Classificaton)로 설정했을 경우, CTCLoss라는 CTC 전용 Loss 함수를 사용합니다. 이 함수는 torch의 기본 nn 라이브러리에서 제공됩니다

Prediction 모듈을 Attn(Attention-based Sequence Prediction)으로 설정하면, Loss 함수는 우리가 흔히 알고 있는 CrossEntropyLoss로 변경됩니다.

6-2) modules 분석 (모델 모듈 코드)

(1) feature_extraction.py (Feat : VGG I RESNET)

이미지의 feature를 추출하는 부분으로 VGG, RCNN, ResNet ,GRCL(Gated RCNN)으로 총 4종류가 있습니다. 이미지 feature 추출 task에서 좋은 성능을 보인다고 알려진 ResNET을 사용하였습니다.

(2) sequence_modeling.py (Seq : None | BiLSTM)

: select SequenceModeling module

BiLSTM 을 사용해서 양방향으로 feature를 학습할 수 있게 설정합니다. 코드에서는 nn.LSTM의 인자에 bidirectional=True만 추가하면 BiLSTM이 구현됩니다

(3) -Transformation: (None | TPS)

: Transformation 모듈을 사용하는 옵션입니다.

7. 결과

20개의 sample data로 ocr 결과를 측정한 결과 평균 99%의 정확도를 얻었습니다.

	Image list	GT	Prediction	accuracy
0	01/000000	6601	6601	100.00%
1	01/000006	3603	3603	100.00%
2	01/000006	1003	1003	100.00%
3	01/000007	9102	9102	100.00%
4	01/000009	9901	9901	100.00%
5	01/000010	2801	2801	100.00%
6	01/000015	7702	7702	100.00%
7	01/000020	1003	1003	100.00%
8	01/000021	4601	4601	100.00%
9	01/000021	2003	2003	100.00%
10	02/001552	995B17R0	995B17R0	100.00%
11	02/001554	8093T42B	8093T42B	100.00%
12	02/001554	8093D41U	8093D41U	100.00%
13	02/001557	8108F51F8	8108F51F8	100.00%
14	02/001557	8109B37A	8109B37A	100.00%
15	02/001558	8108F51F8	8108F51F8	100.00%
16	02/001558	8109B37A	8109B37A	100.00%
17	02/001558	8057D17L	8057D17L	100.00%
18	02/001559	C1051BAA	C1051BAA	100.00%
19	02/001559	C1051BAA	C1051B8A	80.00%
20	03/000623	8109S17H	8109S17H	100.00%
21	03/000623	FB875209-	FB875209-	100.00%
22	03/000624	8094E31J1	8094E31J1	100.00%
23	03/000624	FC359707-	FC359707-	100.00%
24	03/000624	995E21F64	995E21F64	100.00%
25	03/000624	FC360027-	FC360027-	100.00%
26	03/000624	8109E51H	8109E51H	100.00%
27	03/000624	FC360144-	FC360144-	100.00%
28	03/000624	81B9A322	8109A322	90.00%
29	03/000624	FB871275-	FB871275-	100.00%
30	Average A	99		
31	Inference	0.138873		

[그림11] test predict 결과

참고자료

https://github.com/clovaai/deep-text-recognition-benchmark

Google ai 블로그 - https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html
Naver DEVIEW 2019 - 모바일 서비스를 위한 가벼운 이미지 인식/검출용 딥러닝 모델 설계

논문
Searching for MobileNetV3 - ICCV 2019
Benchmark Analysis of Representative Deep Neural Network Architectures - IEEE Access
Do ImageNet Classifiers Generalize to ImageNet? - arXiv
“What Is Wrong With Scene Text Recognition Model Comparisons? Dataset and Model Analysis”. Jeonghun Baek. ICCV.2019.