SR을 적용한 자동차 번호판 추정 향상

과목명 인공지능 응용시스템
학과 정보통신공학과
학번 12171843, 12161721
이름 장동훈, 김현
제출일자 2022. 06. 09

목차

주제

SR을 적용한 자동차 번호판 추정향상

개요

Object detection에서 SR의 필요성

자동차의 번호판은 차량을 식별할 수 있는 중요 정보를 담고 있습니다. 블랙박스 카메라의 화질이나 기상 상황, 도로 상황 등에 따라서 Object detection이 안되거나 잘못된 위치를 찾는 일이 발생합니다. Super Resolution을 통해서 Pre-processing 이후, Object Detection을 진행 시 더 높은 인식률과 정확도를 기대해볼 수 있다.

실험 및 결과

Base Code

- 데이터: Kaggle 'Car License Plate Detection'
- VGG16을 이용한 자동차 번호판 Detection

Dataset information

The number of images: 432장
Original Image Size: 500 x 268 size
Translated Image Size: 224x224크기로 Resize (VGGNet 모델 사용을 위해)

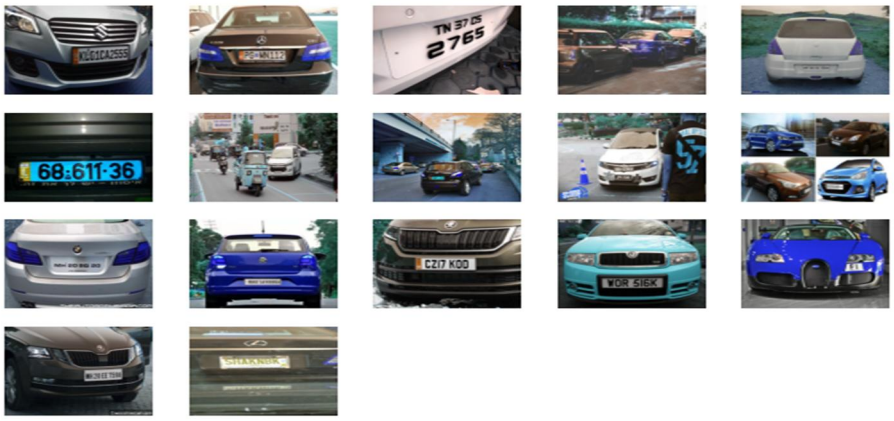

그림 1 - 기본 image

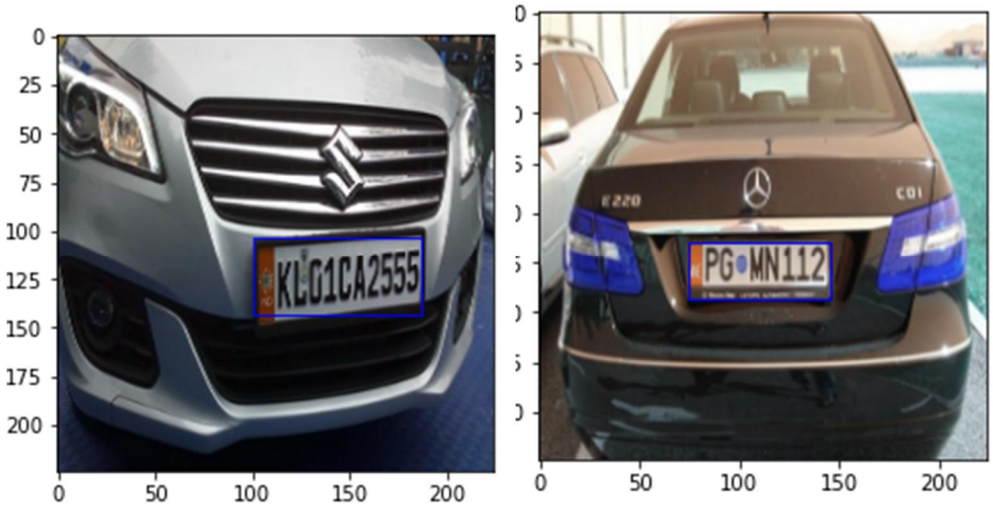

그림 2 - label box 좌표

Train_test_split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.1, random_state=1)
```

➔ 먼저, train과 test dataset을 8:2로 쪼갠다. 그리고 검증데이터를 얻기 위해서 train dataset을 9:1 비율로 쪼개서 데이터를 나눴다.

Model

```
from keras.models import Sequential

from keras.layers import Dense, Flatten

from keras.applications.vgg16 import VGG16
```

```
# Create the model
model = Sequential()
model.add(VGG16(weights="imagenet", include_top=False, input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3)))
model.add(Flatten())
model.add(Dense(128, activation="relu"))
model.add(Dense(128, activation="relu"))
model.add(Dense(64, activation="relu"))
model.add(Dense(4, activation="sigmoid"))

model.layers[-6].trainable = False

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 128)               3211392

 dense_1 (Dense)             (None, 128)               16512

 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 4)                 260

=================================================================
Total params: 17,951,108
Trainable params: 3,236,420
Non-trainable params: 14,714,688
_________________________________________________________________
```

➔ VGG16 모델을 사용

Hyperparameters

```
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
```

```
train = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=50, batch_size=32, verbose=1)
```

Result

```
test_loss, test_accuracy = model.evaluate(X_test, y_test,steps=int(100))

print("Test results ₩n Loss:",test_loss,'₩n Accuracy',test_accuracy)
```

```
 82/100 [=======================>......] - ETA: 0s - loss: 0.0070 - accuracy: 0.8293WARNIN
rupting training. Make sure that your dataset or generator can generate at least `steps_pe
atches). You may need to use the repeat() function when building your dataset.
100/100 [==============================] - 1s 6ms/step - loss: 0.0070 - accuracy: 0.8276
Test results
 Loss: 0.006950060371309519
 Accuracy 0.8275862336158752
```

➔ 약 82.7%의 정확도

Blur처리된 이미지

image

➔ 왼쪽 상단부터 오른쪽으로 순서대로 16x16 Blur, 8x8 Blur, 4x4 Blur, Basic image

Result

Blur Kernel Size	Loss	Accuracy
16x16	0.00912	70.1
8x8	0.00691	79.3
4x4	0.00666	80.4
Basic	0.00695	82.7

그림 3 - Blur 16x16

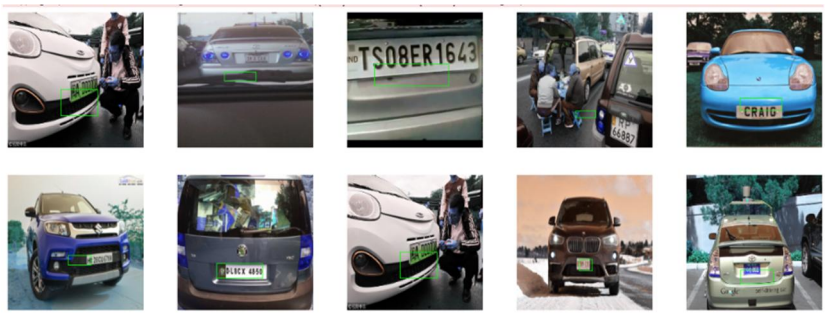

그림 4 - Basic

Super Resolution적용된 이미지 (BSRGAN)

Image

그림 5 - Basic image

그림 6 - BSRGAN이 적용된 image

Test

1. BSRGAN을 적용한 이미지 학습 (base code, Apply-SR.html)
2. Weight Initialization(가중치 초기화) he_normal 사용 (Apply-SR-Copy.html)

```
# Create the model
model = Sequential()

model.add(VGG16(weights="imagenet", include_top=False, input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3)))

model.add(Flatten())
model.add(Dense(128, activation="relu", kernel_initializer='he_normal'))
model.add(Dense(128, activation="relu", kernel_initializer='he_normal'))
model.add(Dense(64, activation="relu", kernel_initializer='he_normal'))
model.add(Dense(4, activation="sigmoid"))

model.layers[-6].trainable = False

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 128)               3211392

 dense_1 (Dense)             (None, 128)               16512

 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 4)                 260

=================================================================
Total params: 17,951,108
Trainable params: 3,236,420
Non-trainable params: 14,714,688
_________________________________________________________________
```

```
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
```

```
train = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=50, batch_size=32, verbose=1)
```

➔ 단순 가중치 초기화 he_normal을 적용

3. Batch Size 64 + Epoch 증가 + EarlyStopping 적용 (Apply-SR-Copy2.html)

```python
# Create the model
model = Sequential()
model.add(VGG16(weights="imagenet", include_top=False, input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3)))
model.add(Flatten())
model.add(Dense(128, activation="relu"))
model.add(Dense(128, activation="relu"))
model.add(Dense(64, activation="relu"))
model.add(Dense(4, activation="sigmoid"))

model.layers[-6].trainable = False

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 7, 7, 512)         14714688

flatten (Flatten)            (None, 25088)             0

dense (Dense)                (None, 128)               3211392

dense_1 (Dense)              (None, 128)               16512

dense_2 (Dense)              (None, 64)                8256

dense_3 (Dense)              (None, 4)                 260

=================================================================
Total params: 17,951,108
Trainable params: 3,236,420
Non-trainable params: 14,714,688
_________________________________________________________________
```

```python
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
```

```python
from keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(patience = 50)

train = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=1000, batch_size=64, verbose=1, callbacks = [ea
```

Result

Type	Loss	Accuracy
Basic	0.00695	82.7
BSRGAN 적용	0.00626	85.5
Weight Initialization	0.00667	88.5
Batch Size + Epoch + EarlyStopping	0.00618	88.5

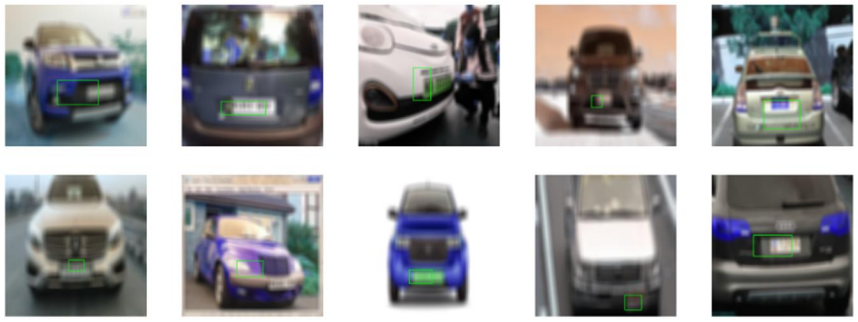

그림 5 - blur16x16 image

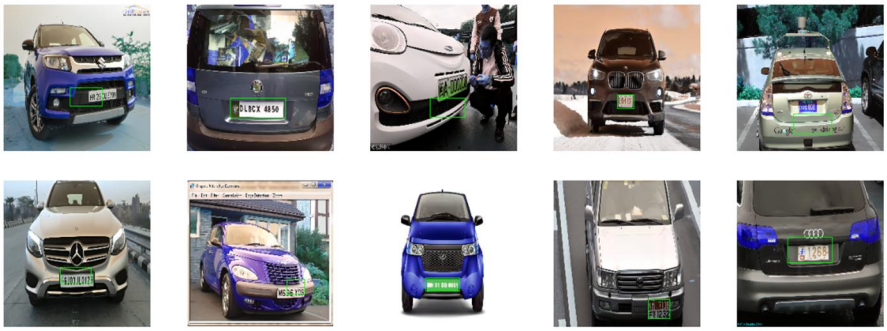

그림 6 - BSRGAN 적용 후 image

A case of trying but bad results

1. Optimizer 수정 (RMSProp, RAdam, NAdam, SGD 등)
2. Loss Function 수정
3. Dropout, Batch Normalized 기법 적용
4. 모델 수정 (Layer 층 추가, activation function 수정 등)

참고 자료

https://www.kaggle.com/datasets/andrewmvd/car-plate-detection

https://www.kaggle.com/code/mclikmb4/vehicle-license-plate-detection-vgg16

https://github.com/kairess/BSRGAN

https://velog.io/@heaseo/BSRGAN-Designing-a-Practical-Degradation-Model-for-Deep-Blind-Image-Super-Resolution