과목명	이미지 딥러닝			
평가 내용	2023년도 2학기 기말 프로젝트 최종 보고서			
프로그램명	안개 제거 유무에 따른 도로 인스턴스 분할 성능의 차이			
프로그램 내용	안개 제거를 한 이미지와 그렇지 않은 이미지를 비교하여 도로 인스턴스 분할 모델을 돌렸을 때 안개 제거를 한 것이 더 성능이 좋은지 확인하는 프로그램			
주요 기능	- 안개가 있는 이미지 인지 없는 이미지인지 구별하는 모델 생성 - 도로 사진에서 도로, 보도, 울타리, 식물, 하늘 등 여러 가지 물체를 구별할 수 있는 모델 생성 - 안개 제거(화이트 밸런스 조절) - IoU 계산을 통해 어떤 이미지가 좋은지 구별			
학생 정보	학번	201904020	이름	윤건용

1. 서론

이 프로젝트는 이미지 내의 안개 유무를 확인하고, 안개 존재를 확인한다면 이미지의 안개 제거를 시도 합니다. 안개 제거 이미지와 안개를 제거하지 않은 원본 이미지 두 개를 가지고 도로 인스턴스 추출을 수행합니다. 각각의 이미지와 인스턴스 분리를 표시한 이미지를 비교하여 IoU 값을 구하고 안개를 제거한 이미지가 원본 이미지보다 인스턴스 분리를 잘 수행하는지 확인하는 프로젝트입니다. 이 프로젝트를 진행하기 위해서 이미지 내 안개 검출 모델과 도로 인스턴스 분리 모델 학습 후 완성하였습니다. 이미지의 안개 제거 방법은 딥러닝 모델을 사용하는 것이 아닌 화이트벨런스 조절을 통하여 안개를 제거하였습니다.

2. 안개 유무 확인 모델 생성(Fog_Detection.ipynb)

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 32)      896

 activation (Activation)     (None, 148, 148, 32)      0

 max_pooling2d (MaxPooling2  (None, 74, 74, 32)        0
 D)
```

conv2d_1 (Conv2D)	(None, 72, 72, 32)	9248
activation_1 (Activation)	(None, 72, 72, 32)	0
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 32)	0
conv2d_2 (Conv2D)	(None, 34, 34, 64)	18496
activation_2 (Activation)	(None, 34, 34, 64)	0
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 64)	0
flatten (Flatten)	(None, 18496)	0
dense (Dense)	(None, 64)	1183808
activation_3 (Activation)	(None, 64)	0
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 1)	65
activation_4 (Activation)	(None, 1)	0

Total params: 1212513 (4.63 MB)
Trainable params: 1212513 (4.63 MB)
Non-trainable params: 0 (0.00 Byte)

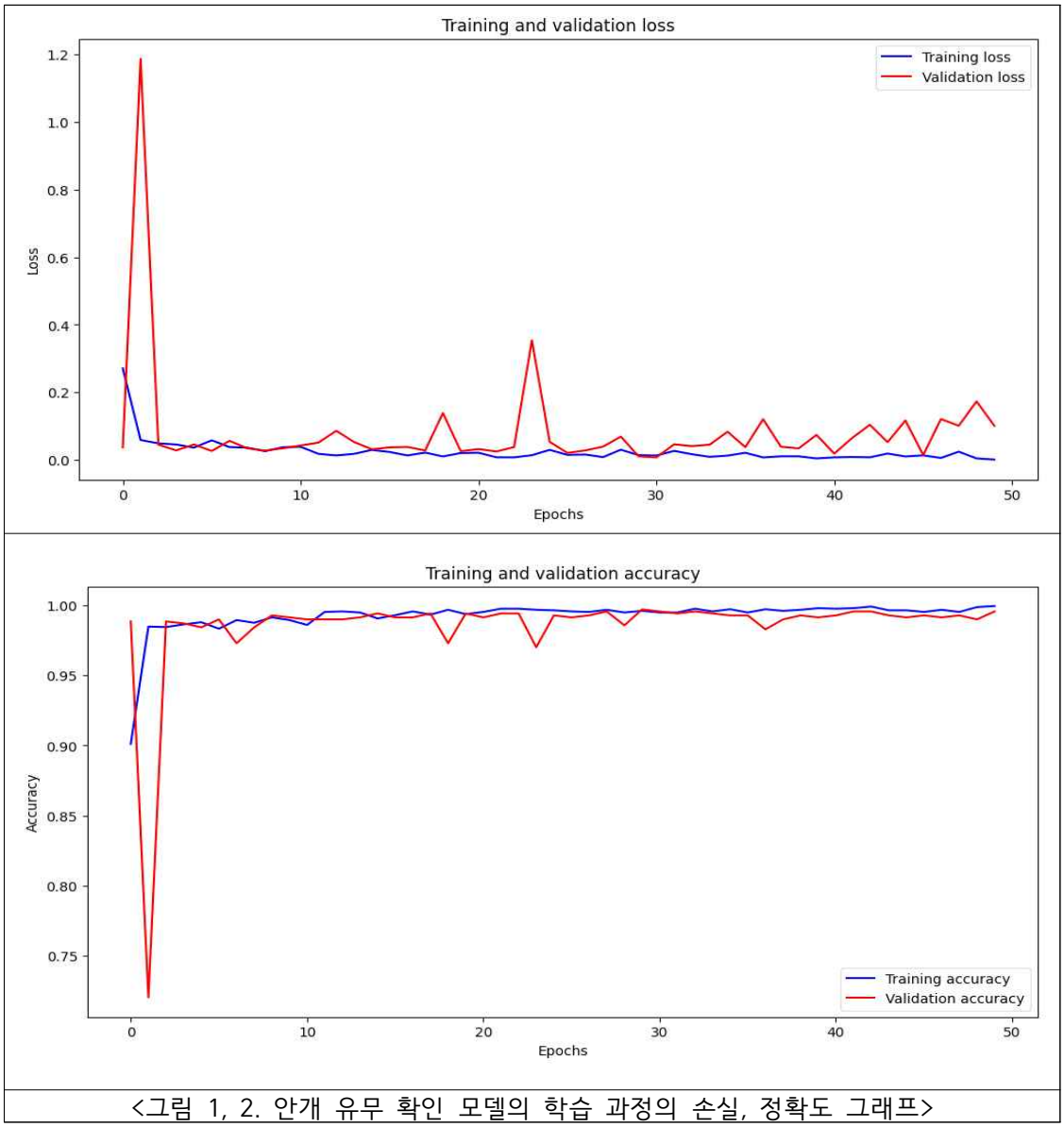
< 표 1 >

안개 확인 모델은 수업시간에 배웠던 컨볼루션 모델을 사용하였다. 층과 층 사이에 활성화 함수 ReLu를 쌓았고, 총 12층 모델을 사용하였다.

```
history = model.fit(
        train_generator,
        steps_per_epoch=2622// 32,
        epochs=50,
        validation_data=validation_generator,
        validation_steps=726// 32
)
```

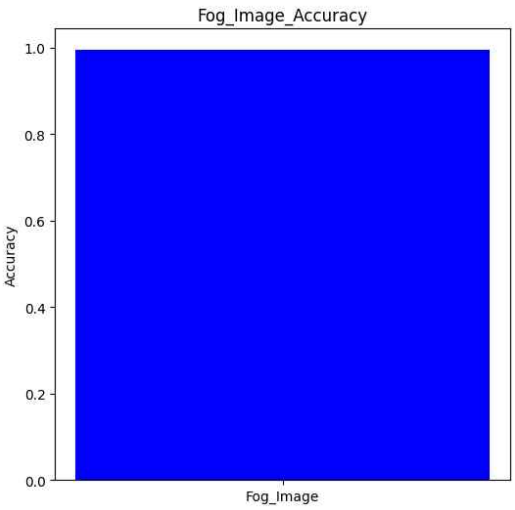
<코드 1. 안개 유무 확인 모델 학습 방법>

모델 학습에서 한 에포크당 수행할 학습 스탭 수를 학습하는 이미지 수에서 32로 나눈 값으로 정하였습니다. 이러한 방법을 사용한 이유는 가지고 있는 학습 이미지를 모두 사용하기 위해서입니다. 이러한 방법을 배치 처리라 한다. 커다란 신경망에서는 데이터 전송이 병목이 되는 경우가 자주 있는데 배치 처리를 함으로써 버스에 주는 부하를 줄여 이미지 1장당 처리시간을 대폭 줄여준다는 이점이 있다.

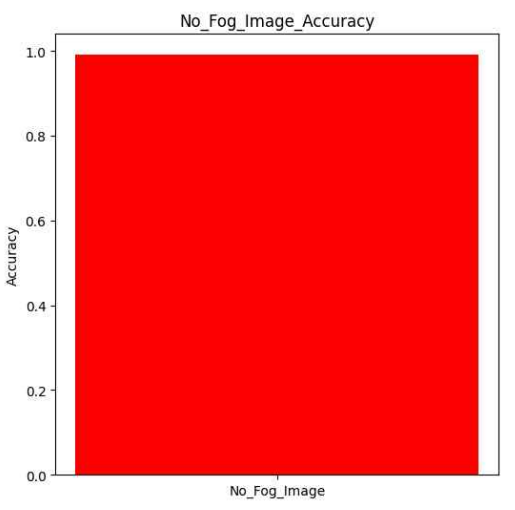


<그림 1, 2. 안개 유무 확인 모델의 학습 과정의 손실, 정확도 그래프>

생성된 모델은 loss: 0.0012, accuracy: 0.9996의 수치를 가지고 있다. 여러 이미지로 다시 검증을 해보니 <그림 3, 4>와 같은 결과가 나왔다.



정확도: 99.48849104859335%



정확도: 99.1044776119403%

<그림 3, 4. 모델 정확도>

3. 도로 인스턴스 분리 모델 생성(Road_Instances_Segmentation.ipynb)

도로 인스턴스 분리 모델을 학습시키기 위한 데이터로 BDD100K 데이터 셋을 사용하였다.

표시되는 각 인스턴스의 색은 <표 2>와 같이 설정하였다.

도로	회색	사람	빨강
보도	연한 노랑	차	파랑
울타리	주황	트럭	파랑
식물	녹색	버스	파랑
하늘	하늘색	알 수 없음	검정

< 표 2. 인스턴스 색상 설정 >

모델은 의료 영상 처리 등에서 세그먼테이션(segmentation) 문제를 해결하기 위해 설계된 모델인 U-net 모델을 사용하였다.

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 192, 256, 3)]     0         []
```

conv2d (Conv2D)	(None, 192, 256, 64)	1792	['input_1[0][0]']
conv2d_1 (Conv2D)	(None, 192, 256, 64)	36928	['conv2d[0][0]']
max_pooling2d (MaxPooling2D)	(None, 96, 128, 64)	0	['conv2d_1[0][0]']
conv2d_2 (Conv2D)	(None, 96, 128, 128)	73856	['max_pooling2d[0][0]']
conv2d_3 (Conv2D)	(None, 96, 128, 128)	147584	['conv2d_2[0][0]']
max_pooling2d_1 (MaxPooling2D)	(None, 48, 64, 128)	0	['conv2d_3[0][0]']
conv2d_4 (Conv2D)	(None, 48, 64, 256)	295168	['max_pooling2d_1[0][0]']
conv2d_5 (Conv2D)	(None, 48, 64, 256)	590080	['conv2d_4[0][0]']
max_pooling2d_2 (MaxPooling2D)	(None, 24, 32, 256)	0	

['conv2d_5[0][0]']
g 2 D)

conv2d_6 (Conv2D) (None, 24, 32, 512) 1180160
['max_pooling2d_2[0][0]']

conv2d_7 (Conv2D) (None, 24, 32, 512) 2359808
['conv2d_6[0][0]']

max_pooling2d_3 (MaxPoolin (None, 12, 16, 512) 0
['conv2d_7[0][0]']
g 2 D)

conv2d_8 (Conv2D) (None, 12, 16, 1024) 4719616
['max_pooling2d_3[0][0]']

conv2d_9 (Conv2D) (None, 12, 16, 1024) 9438208
['conv2d_8[0][0]']

dropout (Dropout) (None, 12, 16, 1024) 0
['conv2d_9[0][0]']

conv2d_transpose (Conv2DTr (None, 24, 32, 512) 4719104
['dropout[0][0]']
a n s p o s e)

concatenate (Concatenate) (None, 24, 32, 1024) 0
['conv2d_7[0][0]',

'conv2d_transpose[0][0]']

conv2d_10 (Conv2D) (None, 24, 32, 512) 4719104
['concatenate[0][0]']

conv2d_11 (Conv2D) (None, 24, 32, 512) 2359808
['conv2d_10[0][0]']

conv2d_transpose_1 (Conv2D (None, 48, 64, 256) 1179904
['conv2d_11[0][0]']
Transpose)

concatenate_1 (Concatenate (None, 48, 64, 512) 0
['conv2d_5[0][0]',
)
'conv2d_transpose_1[0][0]']

conv2d_12 (Conv2D) (None, 48, 64, 256) 1179904
['concatenate_1[0][0]']

conv2d_13 (Conv2D) (None, 48, 64, 256) 590080
['conv2d_12[0][0]']

conv2d_transpose_2 (Conv2D (None, 96, 128, 128) 295040
['conv2d_13[0][0]']
Transpose)

concatenate_2 (Concatenate (None, 96, 128, 256) 0
['conv2d_3[0][0]',
)

'conv2d_transpose_2[0][0]']

conv2d_14 (Conv2D) (None, 96, 128, 128) 295040
['concatenate_2[0][0]']

conv2d_15 (Conv2D) (None, 96, 128, 128) 147584
['conv2d_14[0][0]']

conv2d_transpose_3 (Conv2D (None, 192, 256, 64) 73792
['conv2d_15[0][0]']
Transpose)

concatenate_3 (Concatenate (None, 192, 256, 128) 0
['conv2d_1[0][0]',
)
'conv2d_transpose_3[0][0]']

conv2d_16 (Conv2D) (None, 192, 256, 64) 73792
['concatenate_3[0][0]']

conv2d_17 (Conv2D) (None, 192, 256, 64) 36928
['conv2d_16[0][0]']

conv2d_18 (Conv2D) (None, 192, 256, 20) 1300
['conv2d_17[0][0]']

==========
Total params: 34514580 (131.66 MB)
Trainable params: 34514580 (131.66 MB)
Non-trainable params: 0 (0.00 Byte)

< 표 3 >

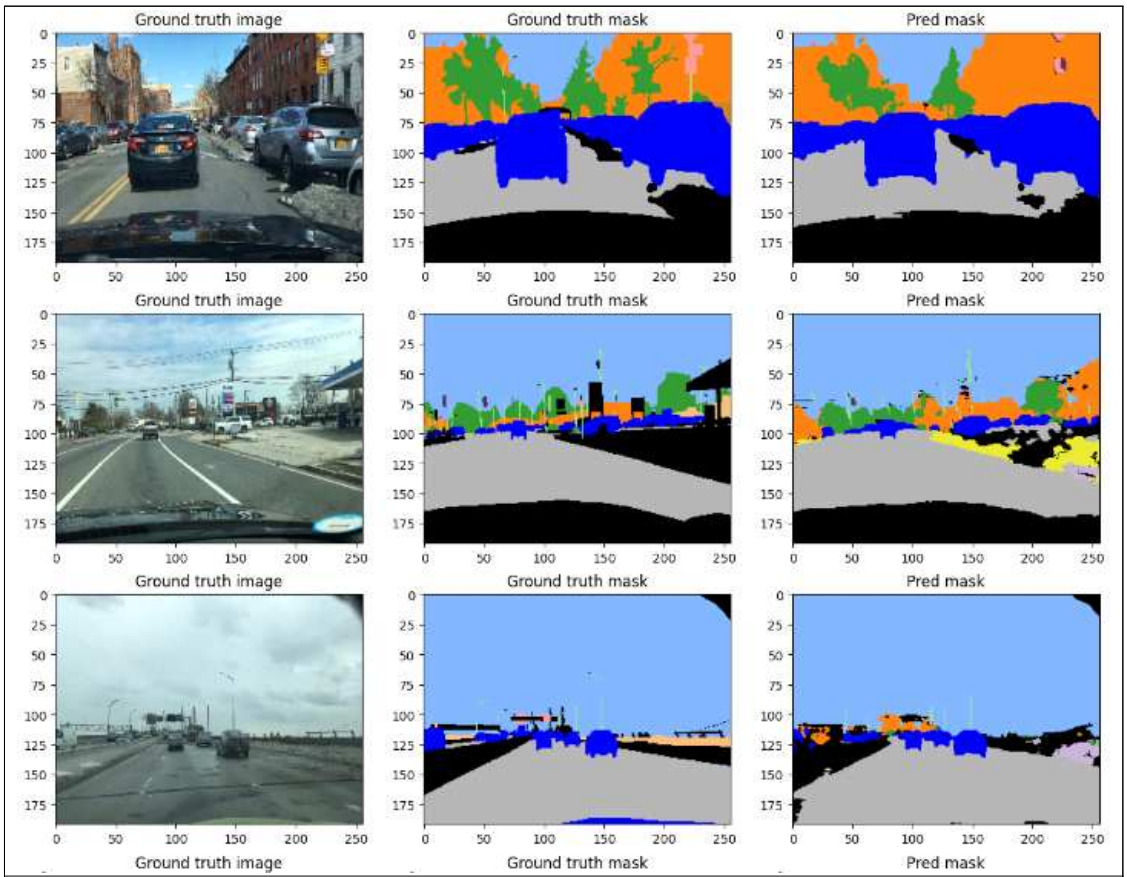
이번에는 앞서 사용한것과 달리 U-net이라는 구조의 컨볼루션 신경망을 사용하였다. U-net은 이미지 세그먼테이션에 특화된 CNN 모델로 인코더와 디코더 두 부분으

로 구성되었다는 특징이 있다.

```
# train the model
initial_epoch = 20
epochs = 21
batch_size = 16
model.fit(
       X_train,
       Y_train,
       initial_epoch=initial_epoch,
       steps_per_epoch=7000// 32,
       epochs=epochs,
       validation_data=(X_val, Y_val),
       validation_steps=1000// batch_size
)
```

<코드 2. 배치 처리 방식을 사용한 도로 인스턴스 모델 학습 방법>

<코드 2>도 위의<코드 1>처럼 배치 처리 방식을 이용하여 성능 향상을 위해 노력해 보았다. 학습 중에 데이터 셋이 부족하여 학습이 중단된 적이 있어 이 문제를 해결하기 위해 32로 설정되었던 batch_size를 16으로 줄여보았다.



< 그림 5. 원본, 라벨링 데이터, 모델을 사용하여 얻은 인스턴스 분리 이미지 비교>

해당 모델 성능의 평가는 객체 검출(Object Detection)과 세그먼테이션(Segmentation) 등의 작업에서 예측의 정확도를 측정하는 데 사용되는 지표인 IoU(Intersection over Union)을 사용하였다.

모델을 사용하여 얻은 예측 마스크 값과 실제 마스크 값을 비교하여 IoU를 구하였다.

```
Accuracy: 0.8488716
Mean IoU: 0.32686156
```

IoU 값은 0에서 1 사이의 값을 가지며, 값이 1에 가까울수록 예측이 정확하다는 것을 의미하고, 0에 가까울수록 예측이 부정확하다는 것을 의미한다. 일반적으로 IoU 값이 0.5 이상이면 해당 예측을 '정확하다'라고 판단하는 경우가 많다.

한정된 자원과 시간으로 충분한 학습을 진행하지 못하여 아직 모델을 완성하지 못하였다. 따라서 21 epoch까지 학습시킨 모델을 다음 단원에 사용하였다.

4. 안개제거 유무에 따른 도로 인스턴스 분할 성능의 차이(All.ipynb)

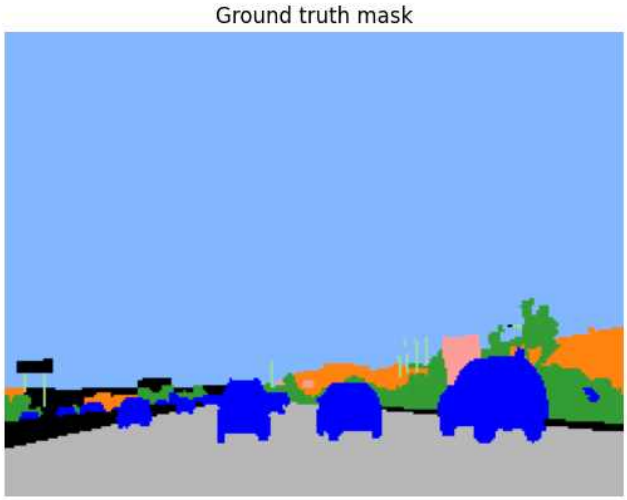
안개 제거를 한 이미지와 제거를 하지 않은 이미지를 각각 인스턴스 분할 모델을 사용하여 인스턴스 분할을 하고 각각의 분할된 이미지 예측 마스크 값들과 실제 마스크 값을 비교하여 안개 제거를 한 이미지가 세그먼테이션 성능에 어느 정도 영향을 끼치는지 확인을 해 보았다.

세그먼테이션을 한 이미지를 비교하기 위해서는 실제 마스크 값이 필요로 한다, 실제 마스크 값은 BDD100K 데이터 셋에 존재하지만 이 데이터 셋 내부에는 만족스러운 안개 이미지가 존재하지 않았다. 이를 해결하기 위해서 데이터 셋에 있는 이미지에 안개를 씌어 데이터를 준비하였다(Make_Foggy_Image.py 사용).

코드를 실행하면 설정한 이미지에 대해 Chapter2에서 만든 안개 유무 확인 모델을 가지고 이미지 내 안개의 유무를 판단해 준다. 이후 준비된 안개 제거 코드를 이용하여 이미지의 안개를 제거한다.

< 그림 6. 원본 안개 이미지와 안개 제거 이미지 >

안개를 제거한 이미지와 원본 이미지는 각각 Chapter3에서 만든 모델을 이용하여 세그멘테이션을 진행, 각각의 예측 마스크 값과 실제 마스크 값을 비교하여 IoU를 구하여 안개 제거가 세그먼테이션에 효과적인지 확인해 보았다.



< 그림 7. 실제 마스크 >

< 그림 8. 안개 제거 이미지의 세그멘테이션, 안개 이미지의 세그멘테이션 >

```
1/1 [==============================] - 0s 225ms/step
data\dehazed_image.jpg's IoU_Value: 0.9149979745161461
1/1 [==============================] - 0s 264ms/step
data/make_foggy_image.jpg's IoU_Value: 0.8202321800797318
```

위의 결괏값처럼 안개를 제거한 이미지가 안개가 있는 이미지보다 좋은 IoU 값을 가지고 있다는 것을 확인할 수 있다.

5. 결론

이 프로젝트를 처음 기획했을 때 예상한 결과는 안개 제거가 세그먼테이션 과정에 큰 도움이 될 거라 예상을 하였다. 결과가 예상한 값대로 나와 좋은 것 같다.

이 프로젝트를 진행하면서 여러 문제에 봉착한 경우가 많았다. 가장 큰 문제는 이미지 인스턴스 분리 모델의 학습이 충분히 되지 않았다는 것이다. 좋은 기회가 되어 3090이 장착된 서버를 사용할 수 있는 기회가 있었지만 그래픽 카드의 메모리 부족으로 학습이 되지 않았던 경우도 있었다. 결국 개인 컴퓨터에 있는 CPU를 이용하여 학습을 진행할 수밖에 없었고 한 에포크당 약 2시간이나 걸리는 바람에 끝까지 학습을 진행 시키지 못하였다. 이후 이 모델의 학습이 충분히 된다면 더 좋은 결괏값이 나올 거라 생각한다.

6. 참고문헌

- https://bdd-data.berkeley.edu/portal.html (BBD100K)
- https://www.kaggle.com/datasets/solesensei/solesensei_bdd100k
- https://paperswithcode.com/paper/bdd100k-a-diverse-driving-video-database-with
- https://www.kaggle.com/code/hikmatullahmohammadi/road-segmentation-for-adas-bdd100k-cpu
- https://github.com/sohyun-l/fifo