

```
In [1]: import tensorflow as tf
import numpy as np
import PIL
import skimage
import albumentations

#필요한 라이브러리를 로드합니다.
import os
import math
import numpy as np
import tensorflow as tf

from PIL import Image
import matplotlib.pyplot as plt
from skimage.io import imread
from skimage.transform import resize
from glob import glob

from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow.keras.optimizers import *

import cv2
import matplotlib.pyplot as plt # matplotlib.pyplot으로 임포트
from collections import Counter

from imageio import imread

print('슝=3')
```

슝=3

Step 1. KITTI 데이터셋 수집과 구축

- 다운로드한 KITTI 데이터에 data augmentation을 적용한 형태로 데이터셋을 구축
- U-Net++는 메모리 사용량이 U-Net보다 꽤 많아집니다. 그러니 배치 사이즈를 적절히 줄여서 설정하시기를 권장

데이터 확인

```
In [2]: # 데이터 디렉토리 및 클래스 확인
mask_paths = glob("./data/training/semantic/*.png") # 마스크 이미지 경로
class_counts = Counter()

# 각 마스크에서 클래스 학생 수 집계
for mask_path in mask_paths:
    mask = cv2.imread(mask_path, 0) # 흑백 이미지로 마스크 로드
    unique, counts = np.unique(mask, return_counts=True) # 고유한 클래스
    class_counts.update(dict(zip(unique, counts)))

# 결과 출력
print("클래스별 학생 수:", class_counts)

# 시각화 (히스토그램)
plt.bar(class_counts.keys(), class_counts.values())
plt.xlabel("클래스")
plt.ylabel("학생 수")
plt.title("클래스별 학생 수 분포")
plt.show()
```

클래스별 학생 수: Counter({21: 28235606, 7: 21507188, 23: 9933656, 22: 8771223, 11: 7594828, 26: 5596521, 8: 3561071, 17: 1266263, 4: 1152119, 12: 855506, 13: 777371, 14: 609495, 9: 558685, 20: 512798, 10: 494826, 16: 299469, 19: 285696, 6: 213731, 27: 208331, 31: 199971, 30: 88031, 24: 87275, 15: 75107, 28: 61718, 5: 60834, 33: 53666, 25: 25810, 29: 8843, 32: 8729, 0: 7349, 18: 2930})

```
/opt/conda/lib/python3.9/site-packages/matplotlib/backends/backend_agg.py:240: RuntimeWarning: Glyph 53364 missing from current font.
    font.set_text(s, 0.0, flags=flags)
/opt/conda/lib/python3.9/site-packages/matplotlib/backends/backend_agg.py:240: RuntimeWarning: Glyph 47000 missing from current font.
    font.set_text(s, 0.0, flags=flags)
/opt/conda/lib/python3.9/site-packages/matplotlib/backends/backend_agg.py:240: RuntimeWarning: Glyph 49828 missing from current font.
    font.set_text(s, 0.0, flags=flags)
/opt/conda/lib/python3.9/site-packages/matplotlib/backends/backend_agg.py:240: RuntimeWarning: Glyph 48324 missing from current font.
    font.set_text(s, 0.0, flags=flags)
/opt/conda/lib/python3.9/site-packages/matplotlib/backends/backend_a
```

```
In [9]: # 마스크의 각 클래스 ID에 대한 고유 색상 생성
class_ids = list(class_counts.keys())
np.random.seed(0) # 결과 재현성을 위해 시드 설정
class_colors = {class_id: np.random.randint(0, 256, 3).tolist() for c
                 in class_ids}

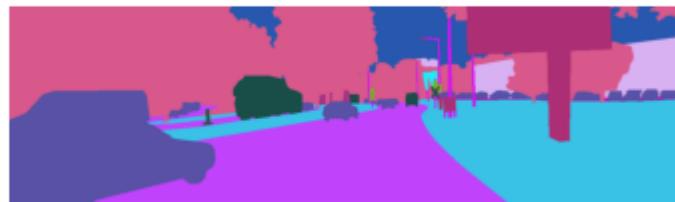
def apply_class_colors(mask, color_map):
    # 빈 컬러 이미지 생성
    color_mask = np.zeros((*mask.shape, 3), dtype=np.uint8)

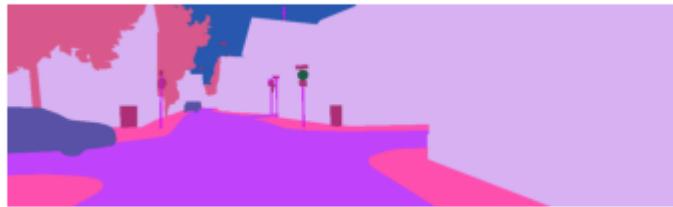
    for class_id, color in color_map.items():
        # 마스크에서 클래스 ID에 해당하는 픽셀에 색상 적용
        color_mask[mask == class_id] = color

    return color_mask

# 무작위 이미지 샘플 시각화
sample_images = random.sample(mask_paths, 5) # 예시로 5개 샘플
for img_path in sample_images:
    mask = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE) # 마스크 이미지를
    colored_mask = apply_class_colors(mask, class_colors) # 클래스 색상으로 칠해진 마스크

    plt.imshow(colored_mask)
    plt.axis('off') # 축을 제거해서 깔끔하게 표시
    plt.show()
```





```
In [11]: # 도로 클래스 ID 설정
road_class_id = 7 # 도로 클래스 ID로 가정

def apply_single_class_color(mask, target_class_id, color):
    # 빈 컬러 이미지 생성
    color_mask = np.zeros((*mask.shape, 3), dtype=np.uint8)

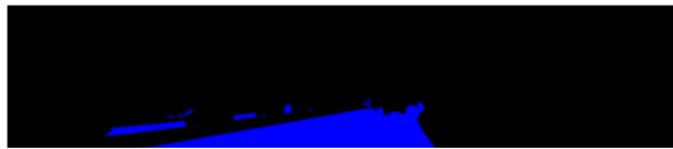
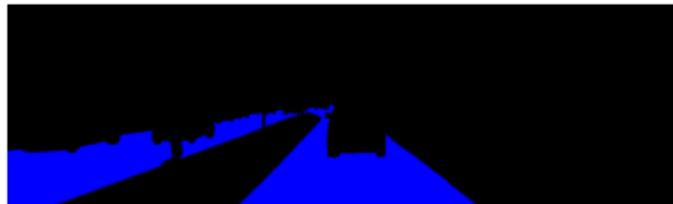
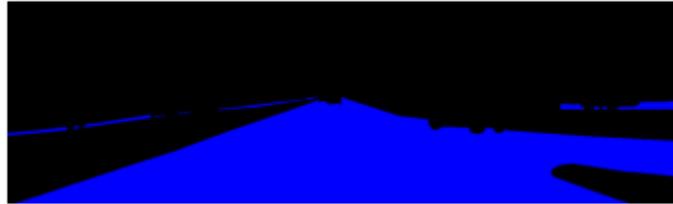
    # 마스크에서 대상 클래스 ID에 해당하는 픽셀에만 색상 적용
    color_mask[mask == target_class_id] = color

    return color_mask

# 도로 클래스 색상 설정 (예: 파란색)
road_color = [0, 0, 255] # 파란색 (BGR 포맷)

# 무작위 이미지 샘플 시작화
sample_images = random.sample(mask_paths, 5) # 예시로 5개 샘플
for img_path in sample_images:
    mask = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE) # 마스크 이미지를
    colored_mask = apply_single_class_color(mask, road_class_id, road_color)

    plt.imshow(colored_mask)
    plt.axis('off') # 축을 제거해서 깔끔하게 표시
    plt.show()
```



In []:

```
In [3]: from albumentations import HorizontalFlip, RandomSizedCrop, Compose, F
def build_augmentation(is_train=True):
    if is_train: # 훈련용 데이터일 경우
        print("Creating augmentation for training")
        return Compose([
            HorizontalFlip(p=0.5), # 50%의 확률로 좌우대칭
            RandomSizedCrop( # 50%의 확률로 RandomSizedCrop
                min_max_height=(300, 370),
                w2h_ratio=370/1242,
                height=224,
                width=224,
                p=0.5
            ),
            RandomBrightnessContrast( # 밝기와 대비를 랜덤하게 조절
                brightness_limit=0.2, # 밝기 조절 범위 (-0.2에서 0.2 사이)
                contrast_limit=0.2, # 대비 조절 범위 (-0.2에서 0.2 사이)
                p=0.5 # 50% 확률로 적용
            ),
            Resize( # 입력 이미지를 224X224로 resize
                width=224,
                height=224
            )
        ])
    else: # 테스트용 데이터일 경우에는 224X224로 resize만 수행
        print("Creating augmentation for testing")
        return Compose([
            Resize(width=224, height=224)
        ])

```

```
In [4]: dir_path = os.getenv('HOME')+'/aiffel/semantic_segmentation/data/train'
augmentation_train = build_augmentation(is_train=True)
augmentation_test = build_augmentation(is_train=False)
input_images = glob(os.path.join(dir_path, "image_2", "*.png"))

len(input_images)
```

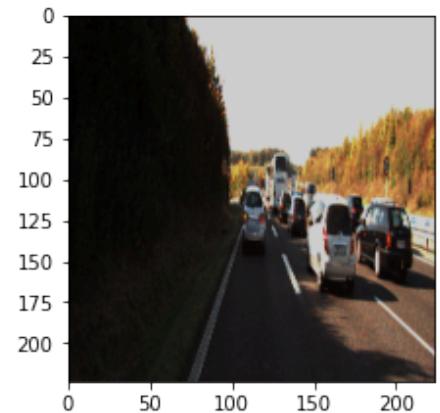
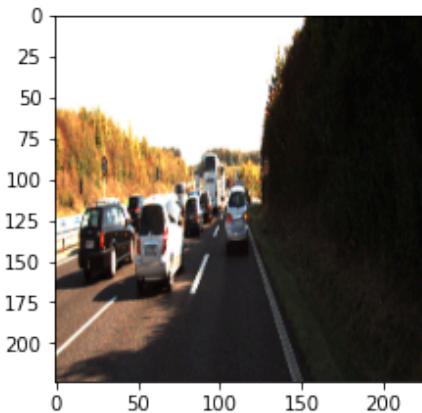
Creating augmentation for training
 Creating augmentation for testing

Out [4]: 200

In [14]: # 훈련 데이터셋에서 5개만 가져와 augmentation을 적용해 봅시다.

```
plt.figure(figsize=(12, 20))
for i in range(5):
    image = imread(input_images[i])
    image_data = {"image":image}
    resized = augmentation_test(**image_data)
    processed = augmentation_train(**image_data)
    plt.subplot(5, 2, 2*i+1)
    plt.imshow(resized["image"]) # 왼쪽이 원본이미지
    plt.subplot(5, 2, 2*i+2)
    plt.imshow(processed["image"]) # 오른쪽이 augment된 이미지

plt.show()
```




```
In [5]: class KittiGenerator(tf.keras.utils.Sequence):
    """
    KittiGenerator는 tf.keras.utils.Sequence를 상속받습니다.
    우리가 KittiDataset을 원하는 방식으로 preprocess하기 위해서 Sequence를 커스텀합니다.
    """

    def __init__(self,
                 dir_path,
                 batch_size=16,
                 img_size=(224, 224, 3),
                 output_size=(224, 224),
                 is_train=True,
                 augmentation=None):
        """
        dir_path: dataset의 directory path입니다.
        batch_size: batch_size입니다.
        img_size: preprocess에 사용할 입력이미지의 크기입니다.
        output_size: ground_truth를 만들어주기 위한 크기입니다.
        is_train: 이 Generator가 학습용인지 테스트용인지 구분합니다.
        augmentation: 적용하길 원하는 augmentation 함수를 인자로 받습니다.
        """

        self.dir_path = dir_path
        self.batch_size = batch_size
        self.is_train = is_train
        self.dir_path = dir_path
        self.augmentation = augmentation
        self.img_size = img_size
        self.output_size = output_size

    # load_dataset()을 통해서 kitti dataset의 directory path에서 라벨과
    self.data = self.load_dataset()

    def load_dataset(self):
        # kitti dataset에서 필요한 정보(이미지 경로 및 라벨)를 directory에서 확인
        # 이때 is_train에 따라 test set을 분리해서 load하도록 해야합니다.
        input_images = glob(os.path.join(self.dir_path, "image_2", "*"))
        label_images = glob(os.path.join(self.dir_path, "semantic", "*"))
        input_images.sort()
        label_images.sort()
        assert len(input_images) == len(label_images)
        data = [ _ for _ in zip(input_images, label_images) ]

        if self.is_train:
            return data[:-30]
        return data[-30:]

    def __len__(self):
        # Generator의 length로서 전체 dataset을 batch_size로 나누고 소수점 첫째자리 반올림
        return math.ceil(len(self.data) / self.batch_size)

    def __getitem__(self, index):
        # 입력과 출력을 만듭니다.
        # 입력은 resize 및 augmentation이 적용된 input image이고
        # 출력은 semantic label입니다.
        batch_data = self.data[
                    index * self.batch_size:
                    (index + 1) * self.batch_size
                    ]
        inputs = np.zeros([self.batch_size, *self.img_size])
        outputs = np.zeros([self.batch_size, *self.output_size])

        for i, data in enumerate(batch_data):
```

```

        input_img_path, output_path = data
        _input = imread(input_img_path)
        _output = imread(output_path)
        _output = (_output==7).astype(np.uint8)*1
    data = {
        "image": _input,
        "mask": _output,
    }
    augmented = self.augmentation(**data)
    inputs[i] = augmented["image"]/255
    outputs[i] = augmented["mask"]
return inputs, outputs

def on_epoch_end(self):
    # 한 epoch가 끝나면 실행되는 함수입니다. 학습중인 경우에 순서를 random shuffle
    self.indexes = np.arange(len(self.data))
    if self.is_train == True :
        np.random.shuffle(self.indexes)
    return self.indexes

```

In [6]:

```

augmentation = build_augmentation()
test_preproc = build_augmentation(is_train=False)

train_generator = KittiGenerator(
    dir_path,
    augmentation=augmentation,
)

test_generator = KittiGenerator(
    dir_path,
    augmentation=test_preproc,
    is_train=False
)

```

Creating augmentation for training
Creating augmentation for testing

In [7]: # 첫 번째 배치를 가져옵니다.

```

batch_images, batch_masks = next(iter(train_generator))

# 입력 이미지와 출력 마스크의 shape 확인
print("Input images shape:", batch_images.shape)
print("Output masks shape:", batch_masks.shape)

```

Input images shape: (16, 224, 224, 3)
Output masks shape: (16, 224, 224)

In [8]: # 첫 번째 배치를 가져옵니다.

```

batch_images, batch_masks = next(iter(test_generator))

# 입력 이미지와 출력 마스크의 shape 확인
print("Input images shape:", batch_images.shape)
print("Output masks shape:", batch_masks.shape)

```

Input images shape: (16, 224, 224, 3)
Output masks shape: (16, 224, 224)

```
In [9]: # 훈련 데이터셋에서 샘플 이미지 시각화
def visualize_dataset_sample(generator, num_samples=3):
    plt.figure(figsize=(12, num_samples * 4))
    for i in range(num_samples):
        # 데이터셋에서 i번째 배치를 가져옵니다.
        images, masks = generator[i]

        # 배치 중 첫 번째 이미지와 마스크를 시각화합니다.
        image = images[0]
        mask = masks[0]

        # 이미지와 마스크 시각화
        plt.subplot(num_samples, 2, 2 * i + 1)
        plt.imshow(image)
        plt.title("Augmented Image")
        plt.axis("off")

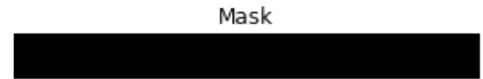
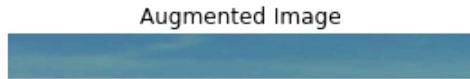
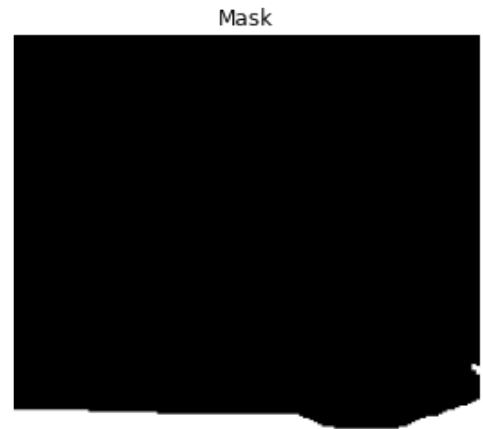
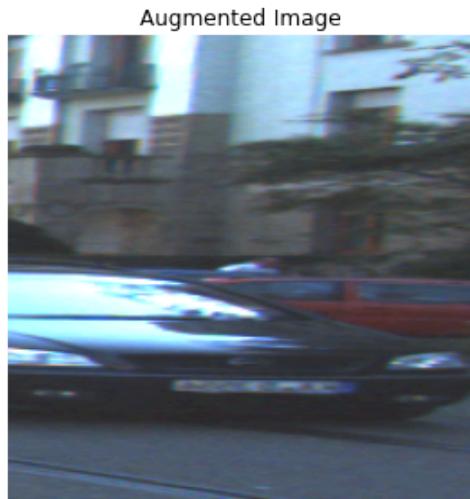
        plt.subplot(num_samples, 2, 2 * i + 2)
        plt.imshow(mask, cmap="gray")
        plt.title("Mask")
        plt.axis("off")

    plt.tight_layout()
    plt.show()

# 훈련 데이터셋과 테스트 데이터셋에서 샘플 시각화
print("Training Data Sample:")
visualize_dataset_sample(train_generator)

print("Test Data Sample:")
visualize_dataset_sample(test_generator)
```

Training Data Sample:



In []:

Step 2. U-Net++ 모델의 구현

- U-Net의 모델 구조와 소스코드를 면밀히 비교해 보면, U-Net++를 어떻게 구현할 수 있을지에 대한 방안을 떠올릴 수 있을 것입니다.
- 이 과정을 통해 U-Net 자체에 대한 이해도도 증진될 것입니다.
- 그 외 적절히 U-Net의 백본 구조, 기타 파라미터 변경 등을 통해 추가적인 성능 향상이 가능할 수도 있습니다.

```
In [10]: import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D
from tensorflow.keras.models import Model

def conv_block(inputs, filters):
    x = Conv2D(filters, (3, 3), activation='relu', padding='same')(inputs)
    x = Conv2D(filters, (3, 3), activation='relu', padding='same')(x)
    return x

def unet_plus_plus(input_shape=(224, 224, 3), num_classes=1):
    inputs = Input(input_shape)

    # Encoder path
    x0 = conv_block(inputs, 64)
    p0 = MaxPooling2D((2, 2))(x0)

    x1 = conv_block(p0, 128)
    p1 = MaxPooling2D((2, 2))(x1)

    x2 = conv_block(p1, 256)
    p2 = MaxPooling2D((2, 2))(x2)

    x3 = conv_block(p2, 512)
    p3 = MaxPooling2D((2, 2))(x3)

    x4 = conv_block(p3, 1024)

    # Decoder path with dense skip connections
    x01 = conv_block(Concatenate()([x0, UpSampling2D((2, 2))(x1)]),
                     64)
    x11 = conv_block(Concatenate()([x1, UpSampling2D((2, 2))(x2)]),
                     128)
    x21 = conv_block(Concatenate()([x2, UpSampling2D((2, 2))(x3)]),
                     256)
    x31 = conv_block(Concatenate()([x3, UpSampling2D((2, 2))(x4)]),
                     512)

    x02 = conv_block(Concatenate()([x0, x01, UpSampling2D((2, 2))(x1)]),
                     64)
    x12 = conv_block(Concatenate()([x1, x11, UpSampling2D((2, 2))(x2)]),
                     128)
    x22 = conv_block(Concatenate()([x2, x21, UpSampling2D((2, 2))(x3)]),
                     256)
    x32 = conv_block(Concatenate()([x3, x31, UpSampling2D((2, 2))(x4)]),
                     512)

    x03 = conv_block(Concatenate()([x0, x01, x02, UpSampling2D((2, 2))(x1)]),
                     64)
    x13 = conv_block(Concatenate()([x1, x11, x12, UpSampling2D((2, 2))(x2)]),
                     128)
    x23 = conv_block(Concatenate()([x2, x21, x22, UpSampling2D((2, 2))(x3)]),
                     256)
    x33 = conv_block(Concatenate()([x3, x31, x32, UpSampling2D((2, 2))(x4)]),
                     512)

    x04 = conv_block(Concatenate()([x0, x01, x02, x03, UpSampling2D((2, 2))(x1)]),
                     64)

    # Output layer
    if num_classes == 1:
        output_activation = 'sigmoid' # Binary classification
    else:
        output_activation = 'softmax' # Multi-class classification

    outputs = Conv2D(num_classes, (1, 1), activation=output_activation)(x04)

    model = Model(inputs, outputs)
    return model

# 모델 생성 및 요약 출력
unet_plus_plus_model = unet_plus_plus(input_shape=(224, 224, 3), num_classes=1)
unet_plus_plus_model.summary()
```

Model: "model"

Layer (type) nected to	Output Shape	Param #	Con
input_1 (InputLayer)	[None, 224, 224, 3] 0		
conv2d (Conv2D) ut_1[0][0]	(None, 224, 224, 64) 1792		inp
conv2d_1 (Conv2D) v2d[0][0]	(None, 224, 224, 64) 36928		con
max_pooling2d (MaxPooling2D)	(None, 112, 112, 64) 0		con

```
In [9]: # def dice_coefficient(y_true, y_pred, smooth=1e-6):
#     # Dice Coefficient 계산
#     intersection = tf.reduce_sum(y_true * y_pred)
#     union = tf.reduce_sum(y_true) + tf.reduce_sum(y_pred)
#     dice = (2. * intersection + smooth) / (union + smooth)
#     return dice

# def combined_bce_dice_loss(y_true, y_pred):
#     # Binary Cross-Entropy 계산
#     bce = tf.keras.losses.binary_crossentropy(y_true, y_pred)

#     # Dice Coefficient 계산
#     dice_loss = 1 - dice_coefficient(y_true, y_pred) # Dice Loss는

#     # 손실 함수 결합
#     combined_loss = bce * 0.5 + dice_loss
#     return combined_loss

# unet_plus_plus_model.compile(optimizer='adam',
#                               loss=combined_bce_dice_loss,
#                               metrics=[dice_coefficient])
```

In [11]: `import tensorflow as tf`

```
def dice_coefficient(y_true, y_pred, smooth=1e-6):
    # y_pred를 이진값으로 변환하지 않고 확률로 계산하는 경우
    y_true_f = tf.cast(y_true, tf.float32)
    y_pred_f = tf.cast(y_pred, tf.float32)

    intersection = tf.reduce_sum(y_true_f * y_pred_f)
    union = tf.reduce_sum(y_true_f) + tf.reduce_sum(y_pred_f)

    dice = (2. * intersection + smooth) / (union + smooth)
    return dice

def combined_bce_dice_loss(y_true, y_pred):
    # Binary Cross-Entropy 계산 (y_pred가 확률이라면 from_logits=False)
    bce = tf.keras.losses.binary_crossentropy(y_true, y_pred, from_logits=True)

    # Dice Loss 계산
    dice_loss = 1 - dice_coefficient(y_true, y_pred)

    # 손실 합수 결합 (가중치 비율 조정 가능)
    combined_loss = bce * 0.5 + dice_loss * 0.5
    return combined_loss

# 모델 컴파일
unet_plus_plus_model.compile(optimizer='adam',
                             loss=combined_bce_dice_loss,
                             metrics=[dice_coefficient])
```

```
In [ ]: # from tensorflow.keras.callbacks import CSVLogger, ModelCheckpoint

# # 로그를 저장할 파일 경로 설정
# log_path = "./training_log_1.csv" # 로그 파일을 저장할 경로와 이름을 지정하세요.
# model_checkpoint_path = "./best_model_1.h5" # 가장 성능이 좋은 모델을 저장할 경로와 이름을 지정하세요.

# # 콜백 함수 설정
# csv_logger = CSVLogger(log_path, append=True) # append=True로 하면 같은 파일에 계속 추가합니다.
# model_checkpoint = ModelCheckpoint(
#     model_checkpoint_path,
#     monitor='val_loss',
#     save_best_only=True,
#     mode='min',
#     verbose=1
# )

# # 모델 학습
# unet_plus_plus_model.fit(
#     train_generator,
#     validation_data=test_generator,
#     steps_per_epoch=len(train_generator),
#     epochs=100,
#     callbacks=[csv_logger, model_checkpoint]
# )

# # 최종 학습된 모델을 지정된 경로에 저장
# unet_plus_plus_model.save(model_checkpoint_path)
```

```
Epoch 1/100
11/11 [=====] - 57s 2s/step - loss: 1.0181
- dice_coefficient: 0.2953 - val_loss: 0.8787 - val_dice_coefficient: 0.3583

Epoch 00001: val_loss improved from inf to 0.87871, saving model to
./best_model.h5
Epoch 2/100
11/11 [=====] - 21s 2s/step - loss: 0.7805
- dice_coefficient: 0.4429 - val_loss: 0.6704 - val_dice_coefficient: 0.5966

Epoch 00002: val_loss improved from 0.87871 to 0.67038, saving model
to ./best_model.h5
Epoch 3/100
11/11 [=====] - 21s 2s/step - loss: 0.7290
- dice_coefficient: 0.6041 - val_loss: 0.5933 - val_dice_coefficient: 0.5783

Epoch 00003: val_loss improved from 0.67038 to 0.5922, saving model to
./best_model.h5
```

```
In [24]: from tensorflow.keras.callbacks import CSVLogger, ModelCheckpoint, EarlyStopping, ReduceLROnPlateau

# 로그를 저장할 파일 경로 설정
log_path = "./training_log_1.csv"
model_checkpoint_path = "./best_model_1.h5"

# 콜백 함수 설정
csv_logger = CSVLogger(log_path, append=True)
model_checkpoint = ModelCheckpoint(
    model_checkpoint_path,
    monitor='val_loss',
    save_best_only=True,
    mode='min',
    verbose=1
)

# EarlyStopping: 성능이 개선되지 않으면 학습을 조기에 종료
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=10, # 'val_loss'가 10 epochs 동안 개선되지 않으면 학습 중단
    restore_best_weights=True,
    verbose=1
)

# ReduceLROnPlateau: 'val_loss'가 개선되지 않으면 학습률을 조정
lr_scheduler = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5, # 학습률을 50%로 감소
    patience=5, # 5 epochs 동안 'val_loss' 개선이 없으면 학습률 감소
    min_lr=1e-6, # 최소 학습률을 설정하여 학습률이 너무 작아지는 것을 방지
    verbose=1
)

# 모델 학습
unet_plus_plus_model.fit(
    train_generator,
    validation_data=test_generator,
    steps_per_epoch=len(train_generator),
    epochs=100,
    callbacks=[csv_logger, model_checkpoint, early_stopping, lr_scheduler]
)

# 최종 학습된 모델을 지정된 경로에 저장
unet_plus_plus_model.save(model_checkpoint_path)
```

```
Epoch 1/100
11/11 [=====] - 90s 2s/step - loss: 0.8114
- dice_coefficient: 0.2469 - val_loss: 0.6438 - val_dice_coefficient: 0.2370

Epoch 00001: val_loss improved from inf to 0.64377, saving model to
./best_model_1.h5
Epoch 2/100
11/11 [=====] - 20s 2s/step - loss: 0.6055
- dice_coefficient: 0.2900 - val_loss: 0.5405 - val_dice_coefficient: 0.3717

Epoch 00002: val_loss improved from 0.64377 to 0.54050, saving model
to ./best_model_1.h5
Epoch 3/100
11/11 [=====] - 21s 2s/step - loss: 0.5238
- dice_coefficient: 0.3916 - val_loss: 0.4963 - val_dice_coefficient: 0.4559

Epoch 00003: val_loss improved from 0.54050 to 0.49630, saving model to
./best_model_1.h5
```

```
In [25]: # 로그 파일 경로
log_path = "training_log_1.csv"

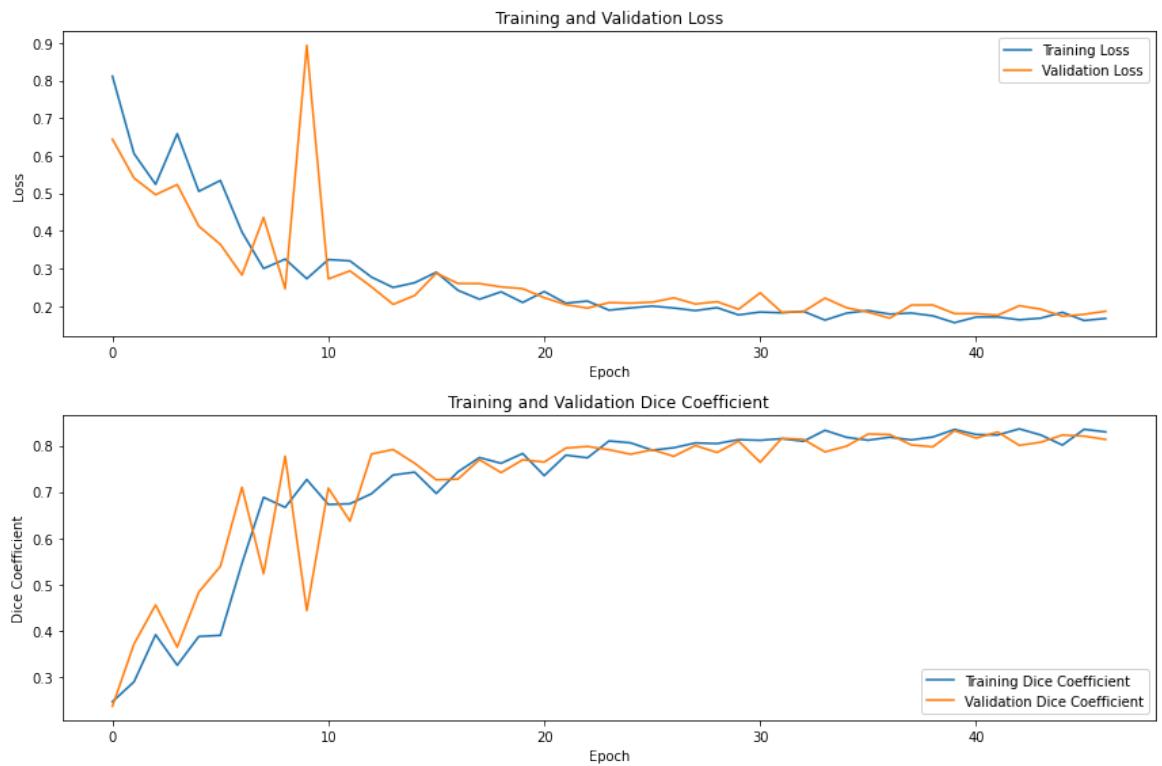
# 로그 파일 읽기
log_data = pd.read_csv(log_path)

# 시각화
plt.figure(figsize=(12, 8))

# 훈련 및 검증 손실 그래프
plt.subplot(2, 1, 1)
plt.plot(log_data['epoch'], log_data['loss'], label='Training Loss')
plt.plot(log_data['epoch'], log_data['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()

# 훈련 및 검증 Dice Coefficient 그래프
plt.subplot(2, 1, 2)
plt.plot(log_data['epoch'], log_data['dice_coefficient'], label='Training Dice Coefficient')
plt.plot(log_data['epoch'], log_data['val_dice_coefficient'], label='Validation Dice Coefficient')
plt.xlabel('Epoch')
plt.ylabel('Dice Coefficient')
plt.title('Training and Validation Dice Coefficient')
plt.legend()

plt.tight_layout()
plt.show()
```



학습 잘 안됨

1. 과적합 -> Regularization
2. 학습률 문제 -> Learning Rate Scheduler

3. 데이터 부족 및 불균형
4. 모델 구조 및 정규화 기법 부족 -> 모델 구조 간소화, Early Stopping

변경사항

1. EarlyStopping 추가
2. ReduceLROnPlateau 추가
3. 데이터 증강에서 RandomBrightnessContrast 추가

In []:

Step 3. U-Net 과 U-Net++ 모델이 수행한 세그멘테이션 결과 분석

- 두 모델의 정량적, 정성적 성능을 비교
- 시각화, IoU 계산 등을 체계적으로 시도해 보면 차이를 발견

```
In [71]: # def get_output(model, preproc, image_path, output_path):
#     origin_img = imread(image_path)
#     data = {"image":origin_img}
#     processed = preproc(**data)
#     output = model(np.expand_dims(processed["image"] / 255, axis=0))
#     output = (output[0].numpy() > 0.5).astype(np.uint8).squeeze(-1) * 255
#     output = Image.fromarray(output)
#     background = Image.fromarray(origin_img).convert('RGBA')
#     output = output.resize((origin_img.shape[1], origin_img.shape[0]))
#     output = Image.blend(background, output, alpha=0.5)
#     output.show()
#     return output

# 잘 안보여서 수정

from imageio import imread

def get_output(model, preproc, image_path, output_path):
    origin_img = imread(image_path)
    data = {"image": origin_img}
    processed = preproc(**data)

    # 모델 예측
    output = model(np.expand_dims(processed["image"] / 255, axis=0))
    output = (output[0].numpy() > 0.5).astype(np.uint8).squeeze(-1) * 255

    # 마스크 색상을 명확히 설정
    mask_color = [255, 0, 0, 128] # 빨간색, 반투명 (RGBA 형식으로 설정)
    output_colored = np.zeros((output.shape[0], output.shape[1], 4), dtype=np.uint8)
    output_colored[output == 255] = mask_color # 마스킹된 부분에 색상을 지정

    # 이미지를 RGBA 형식으로 변환
    background = Image.fromarray(origin_img).convert('RGBA')
    mask = Image.fromarray(output_colored, 'RGBA')

    # 배경 이미지 크기에 맞춰 마스크 이미지 크기 조정
    mask = mask.resize(background.size, Image.ANTIALIAS)

    # 배경 이미지와 마스크를 결합 (alpha 값 조정 가능)
    blended = Image.alpha_composite(background, mask)

    # 결과 이미지 저장 및 표시
    blended.show()
#     blended.save(output_path)
return blended
```

```
In [72]: def get_output_1(model, preproc, image_path, output_path, label_path):
    origin_img = imread(image_path)
    data = {"image":origin_img}
    processed = preproc(**data)
    output = model(np.expand_dims(processed["image"] / 255, axis=0))
    output = (output[0].numpy() >= 0.5).astype(np.uint8).squeeze(-1) * 255
    prediction = output / 255 # 도로로 판단한 영역

    output = Image.fromarray(output)
    background = Image.fromarray(origin_img).convert('RGBA')
    output = output.resize((origin_img.shape[1], origin_img.shape[0]))
    output = Image.blend(background, output, alpha=0.5)
    output.show() # 도로로 판단한 영역을 시각화!

    if label_path:
        label_img = imread(label_path)
        label_data = {"image":label_img}
        label_processed = preproc(**label_data)
        label_processed = label_processed["image"]
        target = (label_processed == 7).astype(np.uint8)*1 # 라벨에서

        return output, prediction, target
    else:
        return output, prediction, _
```

평가문항

상세기준

1. U-Net을 통한 세그멘테이션 작업이 정상적으로 진행되었는가?

KITTI 데이터셋 구성, U-Net 모델 훈련, 결과물 시각화의 한 사이클이 정상 수행되어 세그멘테이션 결과 이미지를 제출하였다.

2. U-Net++ 모델이 성공적으로 구현되었는가?

U-Net++ 모델을 스스로 구현하여 학습 진행 후 세그멘테이션 결과까지 정상 진행되었다.

3. U-Net과 U-Net++ 두 모델의 성능이 정량적/정성적으로 잘 비교되었는가?

U-Net++ 의 세그멘테이션 결과 사진과 IoU 계산치를 U-Net과 비교하여 우월함을 확인하였다.

```
In [84]: def binarize_mask(mask, threshold=0.5):
    # 마스크가 확률일 경우 이진화 (0과 1로 변환)
    return (mask > threshold).astype(np.uint8)

def calculate_iou_score(target, prediction):
    # 마스크 이진화
    target_binary = binarize_mask(target)
    prediction_binary = binarize_mask(prediction)

    # 교집합 영역 계산
    intersection = (target_binary & prediction_binary).sum()

    # 합집합 영역 계산
    union = (target_binary | prediction_binary).sum()

    # IoU 스코어 계산
    iou_score = float(intersection) / float(union) if union != 0 else
    print('IoU : %f' % iou_score)
    return iou_score

def evaluate_model_on_test_set(model, test_preproc, test_image_paths,
iou_scores = []

for i, image_path in enumerate(test_image_paths):
    label_path = label_paths[i]

    # 이미지와 레이블을 모델에 적용하여 예측
    _, prediction, target = get_output_1(
        model,
        test_preproc,
        image_path=image_path,
        output_path=None, # 결과 이미지를 저장하지 않음
        label_path=label_path
    )

    # IoU 점수 계산 및 저장
    iou_score = calculate_iou_score(target, prediction)
    iou_scores.append(iou_score)
    print(f"Image {i + 1}/{len(test_image_paths)} - IoU: {iou_score}")

    # IoU 평균 계산
    mean_iou = np.mean(iou_scores)
    print(f"\nMean IoU for the test set: {mean_iou:.4f}")
return mean_iou
```

```
In [74]: # 테스트 이미지 및 레이블 경로 리스트 생성
test_image_paths = glob(os.path.join(dir_path, "image_2", "*.png"))
label_paths = glob(os.path.join(dir_path, "semantic", "*.png"))

# 정렬하여 이미지와 레이블이 일치하도록 보장
test_image_paths.sort()
label_paths.sort()
```

In []:

U-Net

```
In [37]: # 모델 로드  
model_path = './seg_model_unet.h5'  
unet_model = tf.keras.models.load_model(  
    model_path,  
    custom_objects={'combined_bce_dice_loss': combined_bce_dice_loss,  
                   'dice_coefficient': dice_coefficient}  
)  
unet_model.summary()
```

Model: "model_1"

Layer (type) nected to	Output Shape	Param #	Con
input_2 (InputLayer)	[None, 224, 224, 3] 0		
conv2d_23 (Conv2D) ut_2[0][0]	(None, 224, 224, 64) 1792		inp
conv2d_24 (Conv2D) v2d_23[0][0]	(None, 224, 224, 64) 36928		con
max_pooling2d_4 (MaxPooling2D) conv2d_24[0][0]	(None, 112, 112, 64) 0		con

In [75]: # 완성한 뒤에는 시각화한 결과를 눈으로 확인해봅시다!

```
i = 1      # i값을 바꾸면 테스트용 파일이 달라집니다.
get_output(
    unet_model,
    test_preproc,
    image_path=dir_path + f'/image_2/00{str(i).zfill(4)}_10.png',
    output_path=dir_path + f'./result_{str(i).zfill(3)}.png'
)
```



Out[75]:



In [76]: # 완성한 뒤에는 시각화한 결과를 눈으로 확인해봅시다!

```
i = 1      # i값을 바꾸면 테스트용 파일이 달라집니다.
output, prediction, target = get_output_1(
    unet_model,
    test_preproc,
    image_path=dir_path + f'/image_2/00{str(i).zfill(4)}_10.png',
    output_path=dir_path + f'./result_{str(i).zfill(3)}.png',
    label_path=dir_path + f'/semantic/00{str(i).zfill(4)}_10.png'
)

# IoU 계산
calculate_iou_score(target, prediction)
```



IoU : 0.877921

Out[76]: 0.8779211719567492

```
In [86]: unet_mean_iou = evaluate_model_on_test_set(unet_model,
                                                test_prepoc,
                                                test_image_paths,
                                                label_paths)
```



IoU : 0.930644
Image 1/200 – IoU: 0.9306



U-Net ++

```
In [46]: # 모델 로드
model_path = './best_model_1.h5'
unet_plus_plus_model = tf.keras.models.load_model(
    model_path,
    custom_objects={'combined_bce_dice_loss': combined_bce_dice_loss,
                    'dice_coefficient': dice_coefficient})
unet_plus_plus_model.summary()
```

Model: "model"

Layer (type) nected to	Output Shape	Param #	Con
input_1 (InputLayer)	[None, 224, 224, 3] 0		
conv2d_1 (Conv2D) ut_1[0][0]	(None, 224, 224, 64) 1792		inp
conv2d_1 (Conv2D) v2d[0][0]	(None, 224, 224, 64) 36928		con
max_pooling2d (MaxPooling2D) od_1[0][0]	(None, 112, 112, 64) 0		con

In [80]: # 완성한 뒤에는 시각화한 결과를 눈으로 확인해봅시다!

```
i = 1      # i값을 바꾸면 테스트용 파일이 달라집니다.
get_output(
    unet_plus_plus_model,
    test_preproc,
    image_path=dir_path + f'/image_2/00{str(i).zfill(4)}_10.png',
    output_path=dir_path + f'./result_{str(i).zfill(3)}.png'
)
```



Out[80]:



In [81]: # 완성한 뒤에는 시각화한 결과를 눈으로 확인해봅시다!

```
i = 1      # i값을 바꾸면 테스트용 파일이 달라집니다.
output, prediction, target = get_output_1(
    unet_plus_plus_model,
    test_preproc,
    image_path=dir_path + f'/image_2/00{str(i).zfill(4)}_10.png',
    output_path=dir_path + f'./result_{str(i).zfill(3)}.png',
    label_path=dir_path + f'/semantic/00{str(i).zfill(4)}_10.png'
)

# IoU 계산
calculate_iou_score(target, prediction)
```



IoU : 0.839623

Out[81]: 0.839622641509434

```
In [87]: unet_plus_plus_model_mean_iou = evaluate_model_on_test_set(unet_plus_plus_model,
                                                               test_prepoc,
                                                               test_image_paths,
                                                               label_paths)
```



IoU : 0.830895

Image 1/200 - IoU: 0.8309



```
In [ ]:
```