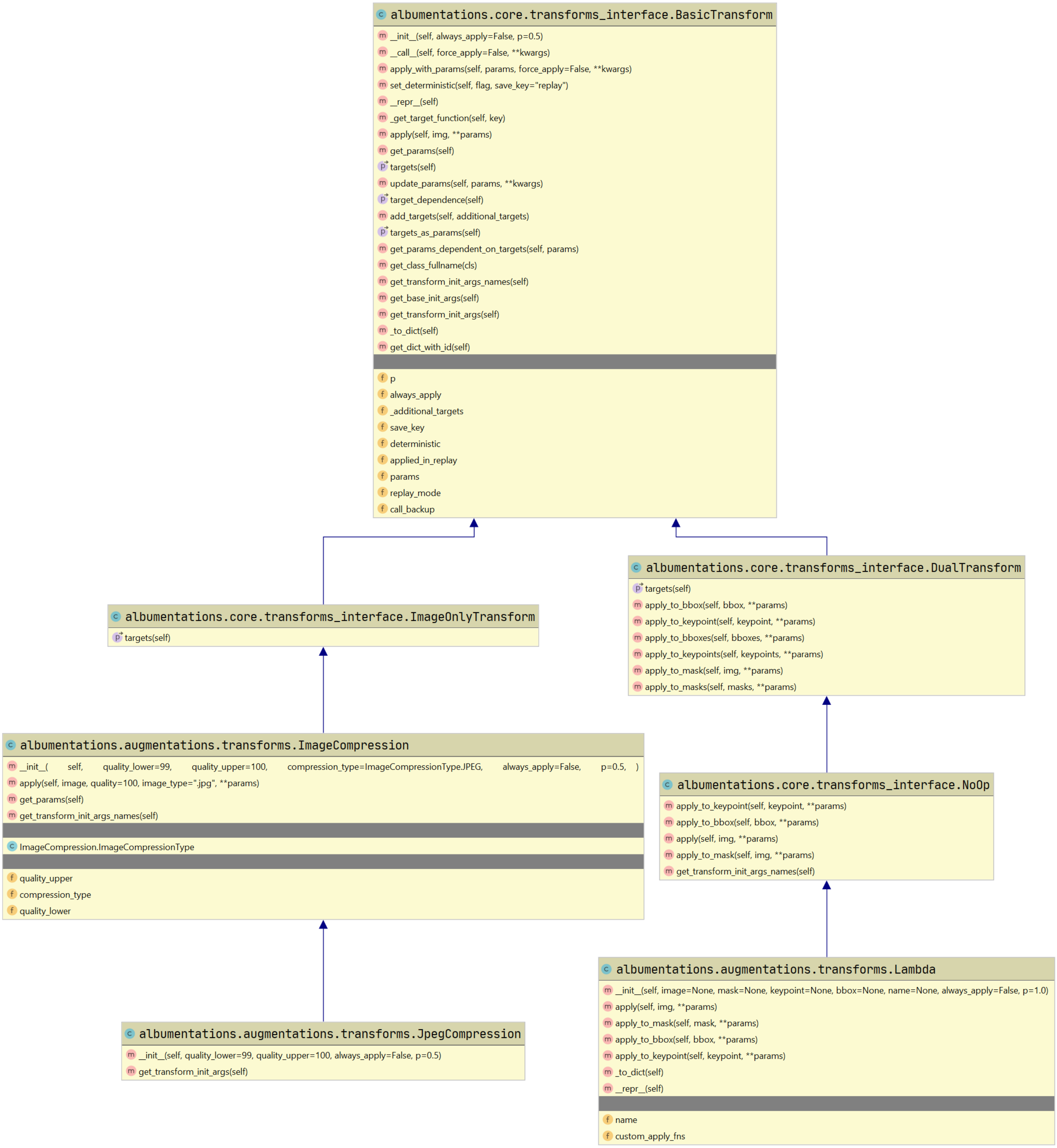
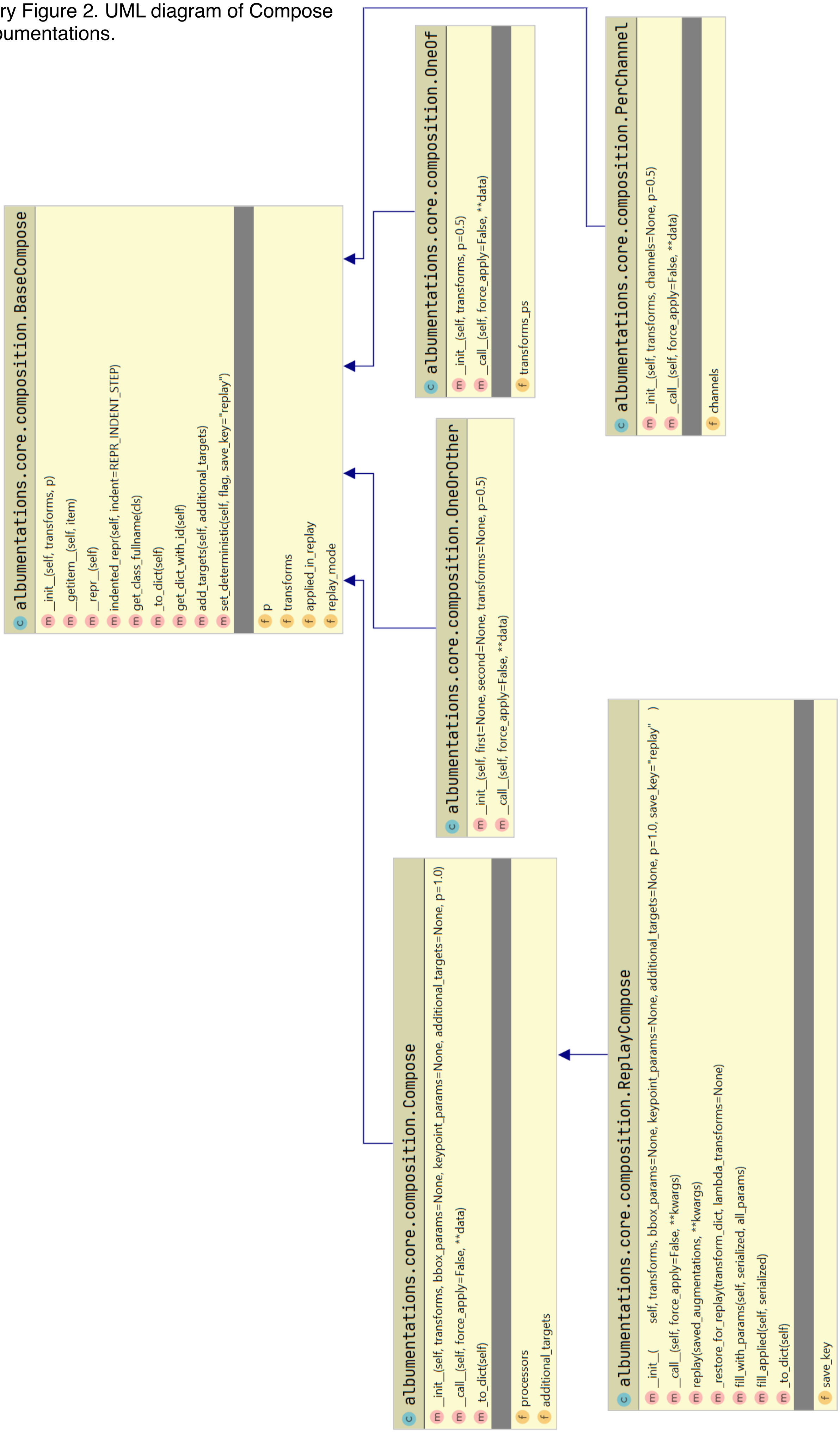
Supplementary Figure 1. UML diagram of Transform classes in Albumentations.



albumentations.core.transforms_interface.BasicTransform
__init__(self, always_apply=False, p=0.5)
__call__(self, force_apply=False, **kwargs)
apply_with_params(self, params, force_apply=False, **kwargs)
set_deterministic(self, flag, save_key="replay")
__repr__(self)
_get_target_function(self, key)
apply(self, img, **params)
get_params(self)
targets(self)
update_params(self, params, **kwargs)
target_dependence(self)
add_targets(self, additional_targets)
targets_as_params(self)
get_params_dependent_on_targets(self, params)
get_class_fullname(cls)
get_transform_init_args_names(self)
get_base_init_args(self)
get_transform_init_args(self)
_to_dict(self)
get_dict_with_id(self)
p
always_apply
_additional_targets
save_key
deterministic
applied_in_replay
params
replay_mode
call_backup
albumentations.core.transforms_interface.DualTransform
targets(self)
apply_to_bbox(self, bbox, **params)
apply_to_keypoint(self, keypoint, **params)
apply_to_bboxes(self, bboxes, **params)
apply_to_keypoints(self, keypoints, **params)
apply_to_mask(self, img, **params)
apply_to_masks(self, masks, **params)
albumentations.core.transforms_interface.ImageOnlyTransform
targets(self)
albumentations.augmentations.transforms.ImageCompression
__init__(self, quality_lower=99, quality_upper=100, compression_type=ImageCompressionType.JPEG, always_apply=False, p=0.5,)
apply(self, image, quality=100, image_type=".jpg", **params)
get_params(self)
get_transform_init_args_names(self)
ImageCompression.ImageCompressionType
quality_upper
compression_type
quality_lower
albumentations.core.transforms_interface.NoOp
apply_to_keypoint(self, keypoint, **params)
apply_to_bbox(self, bbox, **params)
apply(self, img, **params)
apply_to_mask(self, img, **params)
get_transform_init_args_names(self)
albumentations.augmentations.transforms.Lambda
__init__(self, image=None, mask=None, keypoint=None, bbox=None, name=None, always_apply=False, p=1.0)
apply(self, img, **params)
apply_to_mask(self, mask, **params)
apply_to_bbox(self, bbox, **params)
apply_to_keypoint(self, keypoint, **params)
_to_dict(self)
__repr__(self)
name
custom_apply_fns
albumentations.augmentations.transforms.JpegCompression
__init__(self, quality_lower=99, quality_upper=100, always_apply=False, p=0.5)
get_transform_init_args(self)

Supplementary Figure 2. UML diagram of Compose classes in Albumentations.


albumentations.core.composition.BaseCompose
__init__(self, transforms, p)
__getitem__(self, item)
__repr__(self)
indented_repr(self, indent=REPR_INDENT_STEP)
get_class_fullname(cls)
_to_dict(self)
get_dict_with_id(self)
add_targets(self, additional_targets)
set_deterministic(self, flag, save_key="replay")
p
transforms
applied_in_replay
replay_mode
albumentations.core.composition.OneOf
__init__(self, transforms, p=0.5)
__call__(self, force_apply=False, **data)
transforms_ps
albumentations.core.composition.PerChannel
__init__(self, transforms, channels=None, p=0.5)
__call__(self, force_apply=False, **data)
channels
albumentations.core.composition.OneOrOther
__init__(self, first=None, second=None, transforms=None, p=0.5)
__call__(self, force_apply=False, **data)
albumentations.core.composition.Compose
__init__(self, transforms, bbox_params=None, keypoint_params=None, additional_targets=None, p=1.0)
__call__(self, force_apply=False, **data)
_to_dict(self)
processors
additional_targets
albumentations.core.composition.ReplayCompose
__init__(self, transforms, bbox_params=None, keypoint_params=None, additional_targets=None, p=1.0, save_key="replay"   )
__call__(self, force_apply=False, **kwargs)
replay(saved_augmentations, **kwargs)
_restore_for_replay(transform_dict, lambda_transforms=None)
fill_with_params(self, serialized, all_params)
fill_applied(self, serialized)
_to_dict(self)
save_key

Supplementary Listing 1. An example of image augmentation transforms composition in a complex pre-processing pipeline from the APTOS 2019 Blindness Detection Challenge.

```
transform = A.Compose([
    A.OneOf([
        A.ShiftScaleRotate(shift_limit=0.05, scale_limit=0.1,
                           rotate_limit=45,
                           border_mode=cv2.BORDER_CONSTANT, value=0,
                           p=0.5),
        A.ElasticTransform(alpha_affine=0,
                           alpha=35,
                           sigma=5,
                           border_mode=cv2.BORDER_CONSTANT,
                           value=0,
                           p=0.5),
        A.OpticalDistortion(distort_limit=0.11,
                            shift_limit=0.15,
                            border_mode=cv2.BORDER_CONSTANT,
                            value=0,
                            p=0.5),
        A.GridDistortion(border_mode=cv2.BORDER_CONSTANT,
                         value=0,
                         p=0.5),
        A.NoOp()
    ]),
    A.RandomSizedCrop(min_max_height=(int(image_size[0] * 0.75),
                     image_size[0]),
                      height=image_size[0],
                      width=image_size[1],
                      p=0.3),
    A.ISONoise(p=0.5),
    A.OneOf([
        A.RandomBrightnessContrast(brightness_limit=0.5,
                                   contrast_limit=0.4,
                                   p=0.5),
        A.RandomGamma(gamma_limit=(50, 150), p=0.5),
        A.NoOp()
    ]),
    A.OneOf([
        A.FancyPCA(alpha_std=6, p=0.5),
        A.RGBShift(r_shift_limit=40, b_shift_limit=30,
                                g_shift_limit=30, p=0.5),
        A.HueSaturationValue(hue_shift_limit=10,
                                sat_shift_limit=10, p=0.5),
        A.ToGray(p=0.2),
        A.NoOp()
    ]),
    A.ChannelDropout(p=0.5),
    A.RandomGridShuffle(p=0.3),
    A.RandomRotate90(p=0.5),
    A.Transpose(p=0.5)
])
```

Supplementary Listing 2. An example of a custom augmentation that places an “A” logo on the input image and the corresponding binary mask.

```
def custom_augment_image(image, **kwargs):
    image_orig = image.copy()
    cv2.putText(image, "A",
        (150, 350), cv2.FONT_HERSHEY_SCRIPT_COMPLEX, 10,
        (0, 0, 200), thickness=14)
    cv2.circle(
        image,
        (image.shape[1] // 2, image.shape[0] // 2),
        image.shape[0] // 3,
        color=(0, 0, 200),
        thickness=14,
        lineType=cv2.LINE_AA,
    )
    return cv2.addWeighted(image_orig, 0.5, image, 0.5, 0)


def custom_augment_mask(mask, **kwargs):
    cv2.putText(mask, "A",
        (150, 350), cv2.FONT_HERSHEY_SCRIPT_COMPLEX, 10,
        (127, 127, 127), thickness=14)
    cv2.circle(
        mask,
        (mask.shape[1] // 2, mask.shape[0] // 2),
        mask.shape[0] // 3,
        color=(127, 127, 127),
        thickness=14,
        lineType=cv2.LINE_AA,
    )
    return mask
```

```
custom_aug = A.Lambda(image=custom_augment_image,
                     mask=custom_augment_mask)]
```

Supplementary Listing 3. Definitions of four different image augmentation modes for the ablation study on the Inria Aerial Image Labeling dataset.

```
def crop_transform(image_size: Tuple[int, int], min_scale=0.75,
                     max_scale=1.25, input_size=5000):
    return A.OneOrOther(
        A.RandomSizedCrop(
            (int(image_size[0] * min_scale), int(min(input_size,
image_size[0] * max_scale))),
            image_size[0],
            image_size[1],
        ),
        A.CropNonEmptyMaskIfExists(image_size[0], image_size[1]),
    )
def none_augmentations():
    return A.Compose(
        [
            A.Normalize()
        ]
    )
def light_augmentations():
    return A.Compose([
        A.HorizontalFlip(),
        A.RandomBrightnessContrast(),
        A.OneOf([
            A.ShiftScaleRotate(scale_limit=0.05, rotate_limit=15,
border_mode=cv2.BORDER_CONSTANT),
            A.IAAAffine(),
            A.IAAPerspective(),
            A.NoOp()
        ]),
        A.HueSaturationValue(),
        A.Normalize()
    ])
def medium_augmentations():
    return A.Compose(
        [
            A.HorizontalFlip(),
            A.ShiftScaleRotate(scale_limit=0.1, rotate_limit=15,
border_mode=cv2.BORDER_CONSTANT),
            # Add occasion blur/sharpening
            A.OneOf([A.GaussianBlur(), A.IAASharpen(), A.NoOp()]),
            # Spatial-preserving augmentations:
            A.OneOf([A.CoarseDropout(), A.MaskDropout(max_objects=5),
A.NoOp()]),
            A.GaussNoise(),
            A.OneOf([A.RandomBrightnessContrast(), A.CLAHE(),
A.HueSaturationValue(), A.RGBShift(), A.RandomGamma()]),
            # Weather effects
            A.RandomFog(fog_coef_lower=0.01, fog_coef_upper=0.3,
p=0.1),
            A.Normalize(),
        ]
    )
def hard_augmentations():
    return A.Compose(
        [
            A.RandomRotate90(),
            A.Transpose(),
            A.RandomGridShuffle(),
            A.ShiftScaleRotate(
                scale_limit=0.1, rotate_limit=45,
border_mode=cv2.BORDER_CONSTANT, mask_value=0, value=0
            ),
            A.ElasticTransform(border_mode=cv2.BORDER_CONSTANT,
alpha_affine=5, mask_value=0, value=0),
            # Add occasion blur
            A.OneOf([A.GaussianBlur(), A.GaussNoise(),
A.IAAAdditiveGaussianNoise(), A.NoOp()]),
            # D4 Augmentations
            A.OneOf([A.CoarseDropout(),
A.MaskDropout(max_objects=10), A.NoOp()]),
            # Spatial-preserving augmentations:
            A.OneOf(
                [

A.RandomBrightnessContrast(brightness_by_max=True),
                    A.CLAHE(),
                    A.HueSaturationValue(),
                    A.RGBShift(),
                    A.RandomGamma(),
                    A.NoOp(),
                ]
            ),
            # Weather effects
            A.OneOf([A.RandomFog(fog_coef_lower=0.01,
fog_coef_upper=0.3, p=0.1), A.NoOp()]),
            A.Normalize(),
        ]
    )
```