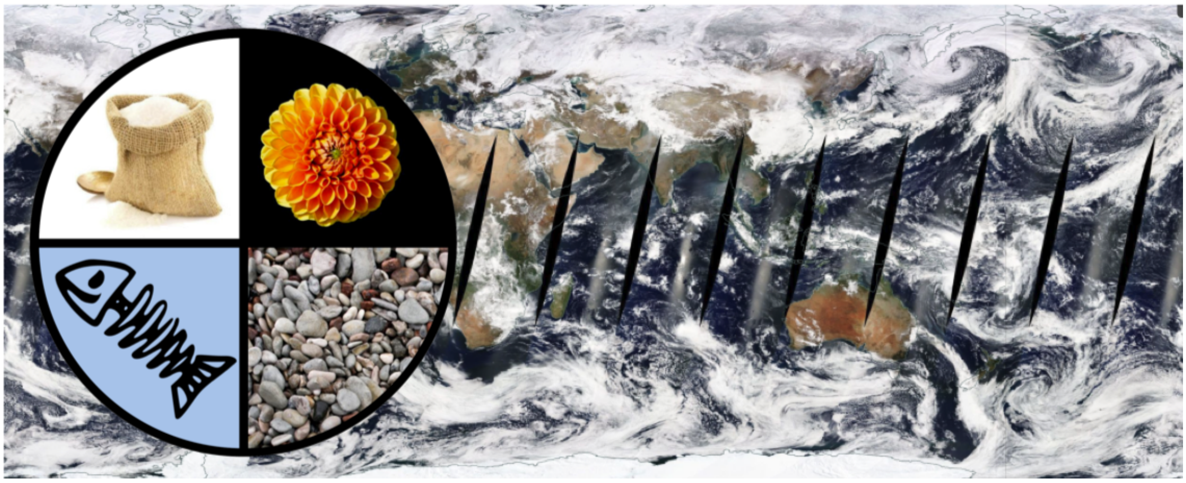


Understanding Clouds from Satellite Images

Can you classify cloud structures from satellites?

In this competition we need to analyze and process cloud images taken from satellites in order to identify cloud formations and help improve the earth's climate understanding.

Dependencies

```
import os
import cv2
import keras
import random
import warnings
import numpy as np
import pandas as pd
import seaborn as sns
import albumentations as albu
import matplotlib.pyplot as plt
from tensorflow import set_random_seed
from sklearn.model_selection import train_test_split
from keras import optimizers
from keras import backend as K
from keras.models import Model
from keras.losses import binary_crossentropy
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
from keras.layers import Input, Conv2D, Conv2DTranspose, MaxPooling2D,
Concatenate

def seed_everything(seed=0):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    set_random_seed(seed)

seed = 0
```

```
seed_everything(seed)
%matplotlib inline
warnings.filterwarnings("ignore")

!pip install segmentation-models==1.0
import segmentation_models as sm
```

```
### Load data
```

```
train = pd.read_csv('../input/understanding_cloud_organization/train.csv')
submission =
pd.read_csv('../input/understanding_cloud_organization/sample_submission.csv')
print('Number of train samples:', train.shape[0])
print('Number of test samples:', submission.shape[0])

# Preprocecss data
train['image'] = train['Image_Label'].apply(lambda x: x.split('_')[0])
train['label'] = train['Image_Label'].apply(lambda x: x.split('_')[1])
submission['image'] = submission['Image_Label'].apply(lambda x: x.split('_')[0])
test = pd.DataFrame(submission['image'].unique(), columns=['image'])

display(train.head())
display(train.describe())
```

```
Number of train samples: 22184
Number of test samples: 14792
```

	Image_Label	EncodedPixels	image	label
0	0011165.jpg_Fish	264918 937 266318 937 267718 937 269118 937 27...	0011165.jpg	Fish
1	0011165.jpg_Flower	1355565 1002 1356965 1002 1358365 1002 1359765...	0011165.jpg	Flower
2	0011165.jpg_Gravel	NaN	0011165.jpg	Gravel
3	0011165.jpg_Sugar	NaN	0011165.jpg	Sugar
4	002be4f.jpg_Fish	233813 878 235213 878 236613 878 238010 881 23...	002be4f.jpg	Fish

	Image_Label	EncodedPixels	image	label
count	22184	11836	22184	22184
unique	22184	11836	5546	4
top	8bd81ce.jpg_Gravel	93291 509 94691 509 96091 509 97491 509 98891 ...	b89964a.jpg	Sugar
freq	1	1	4	5546

Split train and validation sets

```
train['Has Mask'] = ~train['EncodedPixels'].isna()
mask_count_df = train.groupby('image').agg(np.sum).reset_index()
mask_count_df.sort_values('Has Mask', ascending=False, inplace=True)
train_idx, val_idx = train_test_split(mask_count_df.index, test_size=0.2,
random_state=seed)
```

hold out을 0.8:0.2로 나눔

```
def np_resize(img, input_shape):
    height, width = input_shape
    return cv2.resize(img, (width, height))

def mask2rle(img):
    pixels= img.T.flatten()
    pixels = np.concatenate([[0], pixels, [0]])
    runs = np.where(pixels[1:] != pixels[:-1])[0] + 1
    runs[1::2] -= runs[::2]
    return ' '.join(str(x) for x in runs)

def rle2mask(rle, input_shape):
    width, height = input_shape[:2]

    mask = np.zeros( width*height ).astype(np.uint8)

    array = np.asarray([int(x) for x in rle.split()])
    starts = array[0::2]
    lengths = array[1::2]

    current_position = 0
    for index, start in enumerate(starts):
        mask[int(start):int(start+lengths[index])] = 1
        current_position += lengths[index]

    return mask.reshape(height, width).T

def build_masks(rles, input_shape, reshape=None):
    depth = len(rles)
    if reshape is None:
        masks = np.zeros((*input_shape, depth))
    else:
        masks = np.zeros((*reshape, depth))

    for i, rle in enumerate(rles):
        if type(rle) is str:
            if reshape is None:
                masks[:, :, i] = rle2mask(rle, input_shape)
            else:
                mask = rle2mask(rle, input_shape)
                reshaped_mask = np_resize(mask, reshape)
                masks[:, :, i] = reshaped_mask

    return masks

def build_rles(masks, reshape=None):
    width, height, depth = masks.shape
    rles = []
    for i in range(depth):
        mask = masks[:, :, i]

        if reshape:
            mask = mask.astype(np.float32)
            mask = np_resize(mask, reshape).astype(np.int64)

        rle = mask2rle(mask)
        rles.append(rle)

    return rles

def dice_coef(y_true, y_pred, smooth=1):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (2. * intersection + smooth) / (K.sum(y_true_f) + K.sum(y_pred_f) +
smooth)

def dice_loss(y_true, y_pred):
    smooth = 1.
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = y_true_f * y_pred_f
    score = (2. * K.sum(intersection) + smooth) / (K.sum(y_true_f) +
K.sum(y_pred_f) + smooth)
    return 1. - score

def bce_dice_loss(y_true, y_pred):
    return binary_crossentropy(y_true, y_pred) + dice_loss(y_true, y_pred)

def post_process(probability, threshold=0.5, min_size=10000):
    mask = cv2.threshold(probability, threshold, 1, cv2.THRESH_BINARY)[1]
    num_component, component = cv2.connectedComponents(mask.astype(np.uint8))
    predictions = np.zeros(probability.shape, np.float32)
    num = 0
    for c in range(1, num_component):
        p = (component == c)
        if p.sum() > min_size:
            predictions[p] = 1
            num += 1
    return predictions, num
```

이미지 resize, 마스크를 위한 함수, 평가지표, post_process 함수를 정의함

Model parameters

```
BATCH_SIZE = 32
EPOCHS = 15
LEARNING_RATE = 3e-4
HEIGHT = 320
WIDTH = 480
CHANNELS = 3
N_CLASSES = train['label'].nunique()
ES_PATIENCE = 5
RLROP_PATIENCE = 3
DECAY_DROP = 0.5
BACKBONE = 'resnet34'
```

이 부분을 바꿔가면서 진행

Data generator

```
class DataGenerator(keras.utils.Sequence):
    'Generates data for Keras'
    def __init__(self, list_IDs, df, target_df=None, mode='fit',

base_path='../input/understanding_cloud_organization/train_images',
                 batch_size=BATCH_SIZE, dim=(1400, 2100), n_channels=CHANNELS,
reshape=None,
                 n_classes=N_CLASSES, random_state=seed, shuffle=True,
augment=False):
        self.dim = dim
        self.batch_size = batch_size
        self.df = df
        self.mode = mode
        self.base_path = base_path
        self.target_df = target_df
        self.list_IDs = list_IDs
        self.reshape = reshape
        self.n_channels = n_channels
        self.n_classes = n_classes
        self.shuffle = shuffle
        self.augment = augment
        self.random_state = random_state

        self.on_epoch_end()

    def __len__(self):
        'Denotes the number of batches per epoch'
        return int(np.floor(len(self.list_IDs) / self.batch_size))

    def __getitem__(self, index):
        'Generate one batch of data'
        # Generate indexes of the batch
        indexes = self.indexes[index*self.batch_size:(index+1)*self.batch_size]

        # Find list of IDs
        list_IDs_batch = [self.list_IDs[k] for k in indexes]

        X = self.__generate_X(list_IDs_batch)

        if self.mode == 'fit':
            y = self.__generate_y(list_IDs_batch)

            if self.augment:
                X, y = self.__augment_batch(X, y)

            return X, y

        elif self.mode == 'predict':
            return X

        else:
            raise AttributeError('The mode parameter should be set to "fit" or
"predict".')

    def on_epoch_end(self):
        'Updates indexes after each epoch'
        self.indexes = np.arange(len(self.list_IDs))
        if self.shuffle == True:
            np.random.seed(self.random_state)
            np.random.shuffle(self.indexes)

    def __generate_X(self, list_IDs_batch):
        'Generates data containing batch_size samples'
        # Initialization
        if self.reshape is None:
            X = np.empty((self.batch_size, *self.dim, self.n_channels))
        else:
            X = np.empty((self.batch_size, *self.reshape, self.n_channels))

        # Generate data
        for i, ID in enumerate(list_IDs_batch):
            im_name = self.df['image'].iloc[ID]
            img_path = f"{self.base_path}/{im_name}"
            img = cv2.imread(img_path)
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            img = img.astype(np.float32) / 255.

            if self.reshape is not None:
                img = np_resize(img, self.reshape)

            # Store samples
            X[i,] = img

        return X

    def __generate_y(self, list_IDs_batch):
        if self.reshape is None:
            y = np.empty((self.batch_size, *self.dim, self.n_classes),
dtype=int)
        else:
            y = np.empty((self.batch_size, *self.reshape, self.n_classes),
dtype=int)

        for i, ID in enumerate(list_IDs_batch):
            im_name = self.df['image'].iloc[ID]
            image_df = self.target_df[self.target_df['image'] == im_name]

            rles = image_df['EncodedPixels'].values
            if self.reshape is not None:
                masks = build_masks(rles, input_shape=self.dim,
reshape=self.reshape)
            else:
                masks = build_masks(rles, input_shape=self.dim)

            y[i, ] = masks

        return y

    def __augment_batch(self, img_batch, masks_batch):
        for i in range(img_batch.shape[0]):
            img_batch[i, ], masks_batch[i, ] =
self.__random_transform(img_batch[i, ], masks_batch[i, ])

        return img_batch, masks_batch

    def __random_transform(self, img, masks):
        composition = albu.Compose([albu.HorizontalFlip(p=0.5),
                               albu.VerticalFlip(p=0.5),
                               albu.GridDistortion(p=0.2),
                               albu.ElasticTransform(p=0.2)
                              ])

        composed = composition(image=img, mask=masks)
        aug_img = composed['image']
        aug_masks = composed['mask']

        return aug_img, aug_masks

train_generator = DataGenerator(
                  train_idx,
                  df=mask_count_df,
                  target_df=train,
                  batch_size=BATCH_SIZE,
                  reshape=(HEIGHT, WIDTH),
                  n_channels=CHANNELS,
                  n_classes=N_CLASSES,
                  augment=True,
                  random_state=seed)

valid_generator = DataGenerator(
                  val_idx,
                  df=mask_count_df,
                  target_df=train,
                  batch_size=BATCH_SIZE,
                  reshape=(HEIGHT, WIDTH),
                  n_channels=CHANNELS,
                  n_classes=N_CLASSES,
                  random_state=seed)
```

keras.utils.Sequence의 subclass로 Datagegenrator를 만듬 배치사이즈, 이미지 사이즈, Augmentation을 지정할 수 있음

Model

```
preprocess_input = sm.get_preprocessing(BACKBONE)

model = sm.Unet(BACKBONE, classes=N_CLASSES, activation='sigmoid', input_shape=
(HEIGHT, WIDTH, CHANNELS))


es = EarlyStopping(monitor='val_loss', mode='min', patience=ES_PATIENCE,
restore_best_weights=True, verbose=1)
rlrop = ReduceLROnPlateau(monitor='val_loss', mode='min',
patience=RLROP_PATIENCE, factor=DECAY_DROP, min_lr=1e-6, verbose=1)

metric_list = [dice_coef]
callback_list = [es, rlrop]
optimizer = optimizers.Adam(lr=LEARNING_RATE)
model.compile(optimizer=optimizer, loss=bce_dice_loss, metrics=metric_list)
model.summary()
```

segmentation_models 패키지의 Unet 모듈에서 resnet34 인코더를 이용해서 모델링을 진행

```
history = model.fit_generator(generator=train_generator,
                              validation_data=valid_generator,
                              epochs=EPOCHS,
                              callbacks=callback_list,
                              verbose=1).history
```

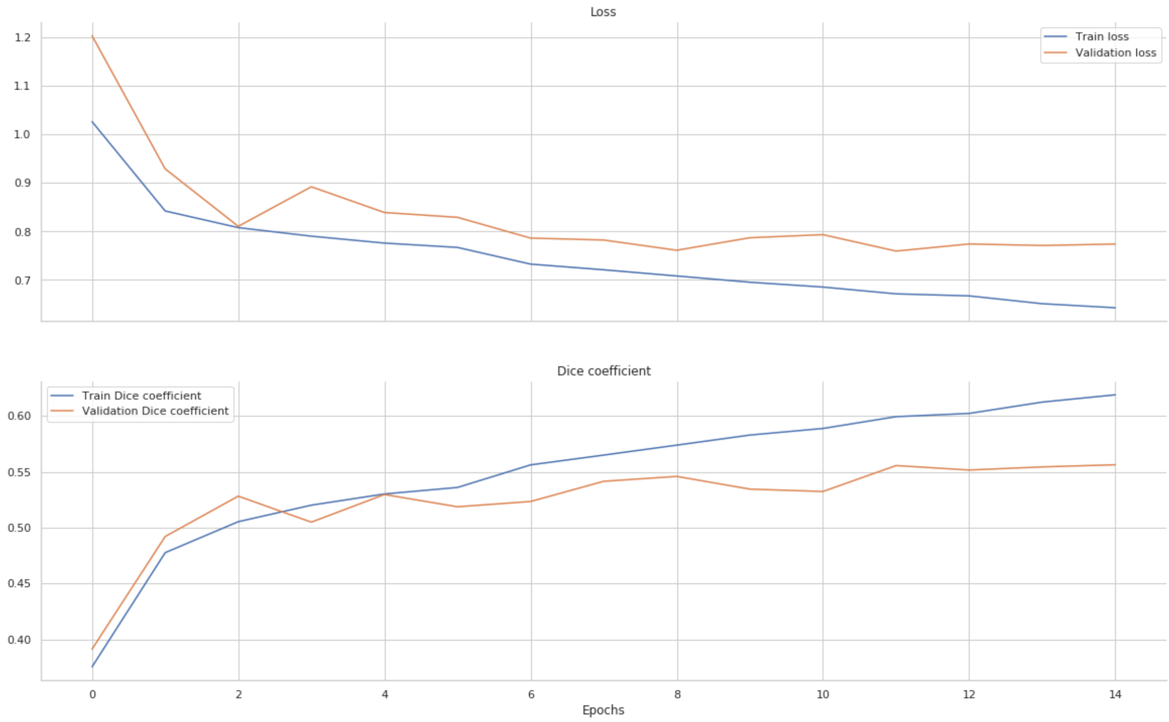
```
Epoch 1/15
138/138 [==============================] - 909s 7s/step - loss: 1.0262 -
dice_coef: 0.3753 - val_loss: 1.2033 - val_dice_coef: 0.3912
Epoch 2/15
138/138 [==============================] - 833s 6s/step - loss: 0.8421 -
dice_coef: 0.4776 - val_loss: 0.9289 - val_dice_coef: 0.4920
Epoch 3/15
138/138 [==============================] - 834s 6s/step - loss: 0.8078 -
dice_coef: 0.5053 - val_loss: 0.8104 - val_dice_coef: 0.5282
Epoch 4/15
138/138 [==============================] - 830s 6s/step - loss: 0.7901 -
dice_coef: 0.5201 - val_loss: 0.8920 - val_dice_coef: 0.5049
Epoch 5/15
138/138 [==============================] - 848s 6s/step - loss: 0.7759 -
dice_coef: 0.5301 - val_loss: 0.8390 - val_dice_coef: 0.5296
Epoch 6/15
138/138 [==============================] - 833s 6s/step - loss: 0.7670 -
dice_coef: 0.5360 - val_loss: 0.8290 - val_dice_coef: 0.5187

Epoch 00006: ReduceLROnPlateau reducing learning rate to 0.0001500000071246177.
Epoch 7/15
138/138 [==============================] - 831s 6s/step - loss: 0.7326 -
dice_coef: 0.5562 - val_loss: 0.7862 - val_dice_coef: 0.5234
Epoch 8/15
138/138 [==============================] - 829s 6s/step - loss: 0.7208 -
dice_coef: 0.5649 - val_loss: 0.7822 - val_dice_coef: 0.5415
Epoch 9/15
138/138 [==============================] - 839s 6s/step - loss: 0.7081 -
dice_coef: 0.5738 - val_loss: 0.7611 - val_dice_coef: 0.5459
Epoch 10/15
```

```
138/138 [==============================] - 827s 6s/step - loss: 0.6952 -
dice_coef: 0.5829 - val_loss: 0.7869 - val_dice_coef: 0.5345
Epoch 11/15
138/138 [==============================] - 841s 6s/step - loss: 0.6852 -
dice_coef: 0.5888 - val_loss: 0.7933 - val_dice_coef: 0.5324
Epoch 12/15
138/138 [==============================] - 825s 6s/step - loss: 0.6712 -
dice_coef: 0.5993 - val_loss: 0.7594 - val_dice_coef: 0.5556
Epoch 13/15
138/138 [==============================] - 848s 6s/step - loss: 0.6670 -
dice_coef: 0.6022 - val_loss: 0.7741 - val_dice_coef: 0.5516
Epoch 14/15
138/138 [==============================] - 832s 6s/step - loss: 0.6509 -
dice_coef: 0.6124 - val_loss: 0.7710 - val_dice_coef: 0.5544
Epoch 15/15
138/138 [==============================] - 841s 6s/step - loss: 0.6425 -
dice_coef: 0.6189 - val_loss: 0.7739 - val_dice_coef: 0.5563

Epoch 00015: ReduceLROnPlateau reducing learning rate to 7.500000356230885e-05.
```

15 epoch만큼 학습을 진행, 메트릭으로는 위에서 정의한 dice_coef를 사용

Model loss graph

```
fig, (ax1, ax2) = plt.subplots(2, 1, sharex='col', figsize=(20, 12))

ax1.plot(history['loss'], label='Train loss')
ax1.plot(history['val_loss'], label='Validation loss')
ax1.legend(loc='best')
ax1.set_title('Loss')

ax2.plot(history['dice_coef'], label='Train Dice coefficient')
ax2.plot(history['val_dice_coef'], label='Validation Dice coefficient')
ax2.legend(loc='best')
ax2.set_title('Dice coefficient')

plt.xlabel('Epochs')
sns.despine()
plt.show()
```



그래프를 보면 loss가 내려가면서 학습이 잘되고 있는 것을 알 수 있다.

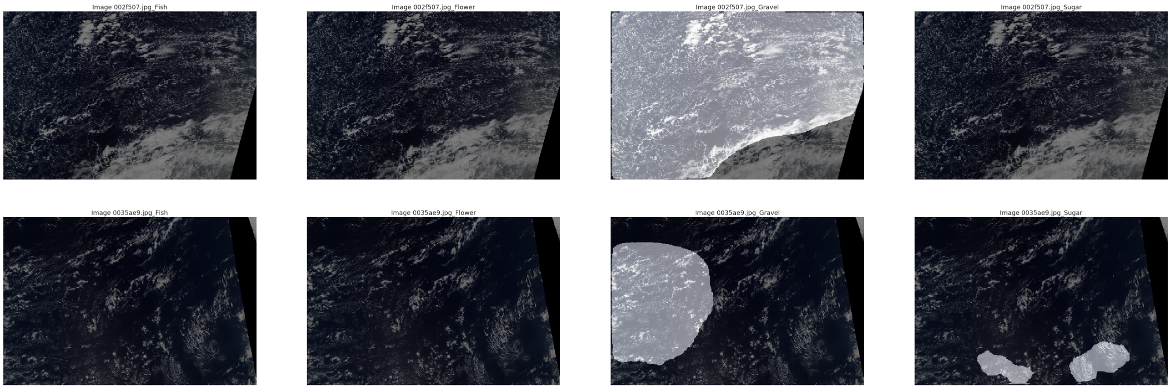
Apply model to test set

```
test_df = []

for i in range(0, test.shape[0], 500):
    batch_idx = list(range(i, min(test.shape[0], i + 500)))

    test_generator = DataGenerator(
                     batch_idx,
                     df=test,
                     target_df=submission,
                     batch_size=1,
                     reshape=(HEIGHT, WIDTH),
                     dim=(350, 525),
                     n_channels=CHANNELS,
                     n_classes=N_CLASSES,
                     random_state=seed,
base_path='../input/understanding_cloud_organization/test_images',
                     mode='predict',
                     shuffle=False)

    batch_pred_masks = model.predict_generator(test_generator)

    for j, b in enumerate(batch_idx):
        filename = test['image'].iloc[b]
        image_df = submission[submission['image'] == filename].copy()
        pred_masks = batch_pred_masks[j, ].round().astype(int)

        ### Post procecssing
        pred_masks_post = batch_pred_masks[j, ].astype('float32')
        for k in range(pred_masks_post.shape[-1]):
            pred_mask = pred_masks_post[...,k]
            pred_mask, num_predict = post_process(pred_mask)
            pred_masks_post[...,k] = pred_mask

        pred_rles_post = build_rles(pred_masks_post, reshape=(350, 525))
        image_df['EncodedPixels_post'] = pred_rles_post
        ###

        pred_rles = build_rles(pred_masks, reshape=(350, 525))
        image_df['EncodedPixels'] = pred_rles
        test_df.append(image_df)

sub_df = pd.concat(test_df)
```

평가데이터에 학습한 모델을 적용함

Inspecting some of the predictions

```
sns.set_style("white")
plt.figure(figsize=[60, 20])
for index, row in sub_df[:8].iterrows():
    img = cv2.imread("../input/understanding_cloud_organization/test_images/%s"
% row['image'])[...,[2, 1, 0]]
    img = cv2.resize(img, (525, 350))
    mask_rle = row['EncodedPixels']
    try: # label might not be there!
        mask = rle_decode(mask_rle)
    except:
        mask = np.zeros((1400, 2100))
    plt.subplot(2, 4, index+1)
    plt.imshow(img)
    plt.imshow(rle2mask(mask_rle, img.shape), alpha=0.5, cmap='gray')
    plt.title("Image %s" % (row['Image_Label']), fontsize=18)
    plt.axis('off')

plt.show()
```

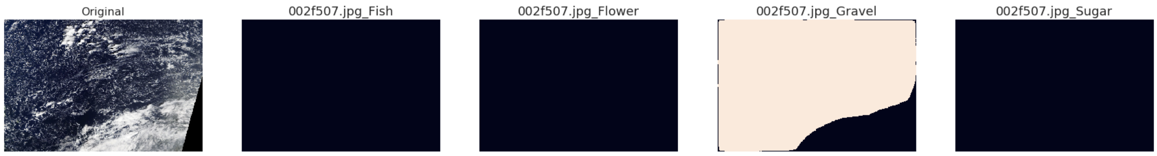


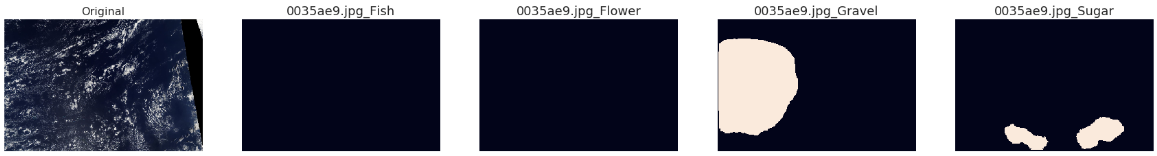
전반적으로 Mask가 잘 씌워졌지만 간혹 이상한 것이 보임

Without background

```
fig, axs = plt.subplots(1, 5, figsize=(30, 30))
```

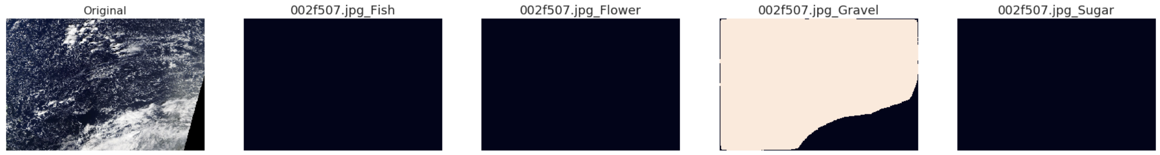
```
axs[0].imshow(cv2.resize(plt.imread('../input/understanding_cloud_organization/t
est_images/%s' % sub_df['image'][0]),(525, 350)))
axs[0].set_title('Original', fontsize=16)
axs[0].axis('off')
for i in range(4):
    axs[i+1].imshow(rle2mask(sub_df['EncodedPixels'][i], img.shape))
    axs[i+1].set_title(sub_df['Image_Label'][i], fontsize=18)
    axs[i+1].axis('off')

fig, axs = plt.subplots(1, 5, figsize=(30, 30))
axs[0].imshow(cv2.resize(plt.imread('../input/understanding_cloud_organization/t
est_images/%s' % sub_df['image'][4]),(525, 350)))
axs[0].set_title('Original', fontsize=16)
axs[0].axis('off')
for i in range(4):
    axs[i+1].imshow(rle2mask(sub_df['EncodedPixels'][4 + i], img.shape))
    axs[i+1].set_title(sub_df['Image_Label'][4 + i], fontsize=18)
    axs[i+1].axis('off')
```

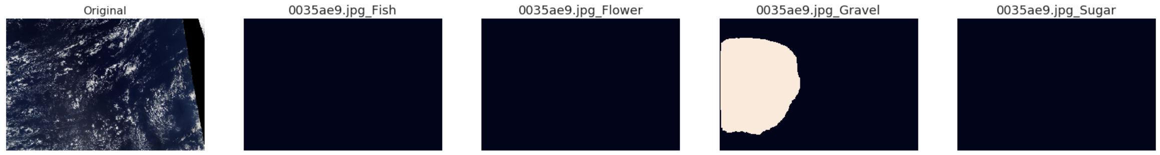




Without background and with post processing

```
fig, axs = plt.subplots(1, 5, figsize=(30, 30))
axs[0].imshow(cv2.resize(plt.imread('../input/understanding_cloud_organization/t
est_images/%s' % sub_df['image'][0]),(525, 350)))
axs[0].set_title('Original', fontsize=16)
axs[0].axis('off')
for i in range(4):
    axs[i+1].imshow(rle2mask(sub_df['EncodedPixels_post'][i], img.shape))
    axs[i+1].set_title(sub_df['Image_Label'][i], fontsize=18)
    axs[i+1].axis('off')

fig, axs = plt.subplots(1, 5, figsize=(30, 30))
axs[0].imshow(cv2.resize(plt.imread('../input/understanding_cloud_organization/t
est_images/%s' % sub_df['image'][4]),(525, 350)))
axs[0].set_title('Original', fontsize=16)
axs[0].axis('off')
for i in range(4):
    axs[i+1].imshow(rle2mask(sub_df['EncodedPixels_post'][4 + i], img.shape))
    axs[i+1].set_title(sub_df['Image_Label'][4 + i], fontsize=18)
    axs[i+1].axis('off')
```





post-processing을 적용하면 한결 나아진 결과를 볼 수 있음

Regular submission

```
submission_df = sub_df[['Image_Label' ,'EncodedPixels']]
submission_df.to_csv('submission.csv', index=False)
display(submission_df.head())
```

	Image_Label	EncodedPixels
0	002f507.jpg_Fish	
1	002f507.jpg_Flower	
2	002f507.jpg_Gravel	109 11 172 8 459 11 522 8 707 330 1056 337 140...
3	002f507.jpg_Sugar	
4	0035ae9.jpg_Fish	

threshold를 통해 post processing하지 않은 submission

Submission with post processing

```
submission_df_post = sub_df[['Image_Label' ,'EncodedPixels_post']]
submission_df_post.columns = ['Image_Label' ,'EncodedPixels']
submission_df_post.to_csv('submission_post.csv', index=False)
display(submission_df_post.head())
```

	Image_Label	EncodedPixels
0	002f507.jpg_Fish	
1	002f507.jpg_Flower	
2	002f507.jpg_Gravel	109 11 172 8 459 11 522 8 707 330 1056 337 140...
3	002f507.jpg_Sugar	
4	0035ae9.jpg_Fish	

threshold를 통해 post processing을 한 submission (threshold를 0.5로 설정하거나 직접 최적값을 찾아야하는 단점이 있음)