

Data Generators

One-hot encoding classes

```
train_df = pd.read_csv('./drive/My Drive/Colab Notebooks/input/train.csv')
train_df.head()
```

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

- 훈련 데이터 로드

```
train_df = train_df[~train_df['EncodedPixels'].isnull()]
train_df['Image'] = train_df['Image_Label'].map(lambda x: x.split('_')[0])
train_df['Class'] = train_df['Image_Label'].map(lambda x: x.split('_')[1])
classes = train_df['Class'].unique()
train_df = train_df.groupby('Image')['Class'].agg(set).reset_index()
for class_name in classes:
    train_df[class_name] = train_df['Class'].map(lambda x: 1 if class_name in x
else 0)
train_df.head()
```

	Image	Class	Fish	Flower	Sugar	Gravel
0	0011165.jpg	{Flower, Fish}	1	1	0	0
1	002be4f.jpg	{Flower, Sugar, Fish}	1	1	1	0
2	0031ae9.jpg	{Flower, Sugar, Fish}	1	1	1	0
3	0035239.jpg	{Gravel, Flower}	0	1	0	1
4	003994e.jpg	{Gravel, Sugar, Fish}	1	0	1	1

```
# dictionary for fast access to ohe vectors
img_2_ohe_vector = {img:vec for img, vec in zip(train_df['Image'],
train_df.iloc[:,
2:].values)}
```

- One-Hot encoding, 해당 이미지에 포함된 클래스 마스크 표시

Stratified split into train / val

```
train_imgs, val_imgs =
    train_test_split(train_df['Image'].values,
                    test_size=0.2,
                    stratify=train_df['Class'].map(lambda x:
str(sorted(list(x)))),
                    # sorting present classes in lexicographical order, just to
be sure
                    random_state=2019)
```

- 훈련 집합을 훈련집합/검증집합으로 나눔

Data Generator

```
class DataGenenerator(Sequence):
    def __init__(self, images_list=None, folder_imgs=train_imgs_folder,
                 batch_size=32, shuffle=True, augmentation=None,
                 resized_height=224, resized_width=224, num_channels=3):
        self.batch_size = batch_size
        self.shuffle = shuffle
        self.augmentation = augmentation
        if images_list is None:
            self.images_list = os.listdir(folder_imgs)
        else:
            self.images_list = deepcopy(images_list)
        self.folder_imgs = folder_imgs
        self.len = len(self.images_list) // self.batch_size
        self.resized_height = resized_height
        self.resized_width = resized_width
        self.num_channels = num_channels
        self.num_classes = 4
        self.is_test = not 'train' in folder_imgs
        if not shuffle and not self.is_test:
            self.labels = [img_2_ohe_vector[img] for img in
self.images_list[:self.len*self.batch_size]]

    def __len__(self):
        return self.len

    def on_epoch_start(self):
        if self.shuffle:
            random.shuffle(self.images_list)

    def __getitem__(self, idx):
        current_batch = self.images_list[idx * self.batch_size: (idx + 1) *
self.batch_size]
        X = np.empty((self.batch_size, self.resized_height, self.resized_width,
self.num_channels))
        y = np.empty((self.batch_size, self.num_classes))

        for i, image_name in enumerate(current_batch):
            path = os.path.join(self.folder_imgs, image_name)
            img = cv2.resize(cv2.imread(path), (self.resized_height,
self.resized_width)).astype(np.float32)
            if not self.augmentation is None:
                augmented = self.augmentation(image=img)
                img = augmented['image']
            X[i, :, :, :] = img/255.0
            if not self.is_test:
                y[i, :] = img_2_ohe_vector[image_name]
        return X, y

    def get_labels(self):
        if self.shuffle:
            images_current = self.images_list[:self.len*self.batch_size]
            labels = [img_2_ohe_vector[img] for img in images_current]
        else:
```

```

labels = self.labels
return np.array(labels)

```

- 이미지 데이터를 batch 단위로 가져오기 위해 `DataGenerator` 클래스 정의
 - resize, augmentation 등 처리 과정 포함

```

augmentations_train = Compose([
    VerticalFlip(), HorizontalFlip(), Rotate(limit=20), GridDistortion()
], p=1)

```

- 사용할 Augmentation 정의

Generator Instances

```

data_generator_train = DataGenerator(train_imgs,
augmentation=augmentations_train)
data_generator_train_eval = DataGenerator(train_imgs, shuffle=False)
data_generator_val = DataGenerator(val_imgs, shuffle=False)

```

- 학습에 사용할 데이터 생성

PR-AUC-based Callback

```

class PrAucCallback(Callback):
    def __init__(self, data_generator, num_workers=num_cores,
                 early_stopping_patience=5,
                 plateau_patience=3, reduction_rate=0.5,
                 stage='train', checkpoints_path='checkpoints/'):
        super(Callback, self).__init__()
        self.data_generator = data_generator
        self.num_workers = num_workers
        self.class_names = ['Fish', 'Flower', 'Sugar', 'Gravel']
        self.history = [[] for _ in range(len(self.class_names) + 1)] # to store
per each class and also mean PR AUC
        self.early_stopping_patience = early_stopping_patience
        self.plateau_patience = plateau_patience
        self.reduction_rate = reduction_rate
        self.stage = stage
        self.best_pr_auc = -float('inf')
        if not os.path.exists(checkpoints_path):
            os.makedirs(checkpoints_path)
        self.checkpoints_path = checkpoints_path

    def compute_pr_auc(self, y_true, y_pred):
        pr_auc_mean = 0
        print(f"\n{'#' * 30}\n")
        for class_i in range(len(self.class_names)):
            precision, recall, _ = precision_recall_curve(y_true[:, class_i],
y_pred[:, class_i])
            pr_auc = auc(recall, precision)
            pr_auc_mean += pr_auc / len(self.class_names)
            print(f"PR AUC {self.class_names[class_i]}, {self.stage}:
{pr_auc:.3f}\n")
            self.history[class_i].append(pr_auc)

```

```

        print(f"\n{'#' * 20}\n PR AUC mean, {self.stage}:
{pr_auc_mean:.3f}\n{'#' * 20}\n")
        self.history[-1].append(pr_auc_mean)
        return pr_auc_mean

    def is_patience_lost(self, patience):
        if len(self.history[-1]) > patience:
            best_performance = max(self.history[-1][-(patience + 1):-1])
            return best_performance == self.history[-1][-(patience + 1)] and
best_performance >= self.history[-1][-1]

    def early_stopping_check(self, pr_auc_mean):
        if self.is_patience_lost(self.early_stopping_patience):
            self.model.stop_training = True

    def model_checkpoint(self, pr_auc_mean, epoch):
        if pr_auc_mean > self.best_pr_auc:
            # remove previous checkpoints to save space
            for checkpoint in glob.glob(os.path.join(self.checkpoints_path,
'classifier_densenet169_epoch_*')):
                os.remove(checkpoint)
            self.best_pr_auc = pr_auc_mean
            self.model.save(os.path.join(self.checkpoints_path,
f'classifier_densenet169_epoch_{epoch}_val_pr_auc_{pr_auc_mean}.h5'))

            print(f"\n{'#' * 20}\nSaved new checkpoint\n{'#' * 20}\n")

    def reduce_lr_on_plateau(self):
        if self.is_patience_lost(self.plateau_patience):
            new_lr = float(keras.backend.get_value(self.model.optimizer.lr)) *
self.reduction_rate
            keras.backend.set_value(self.model.optimizer.lr, new_lr)
            print(f"\n{'#' * 20}\nReduced learning rate to {new_lr}.\n{'#' * 20}\n")

    def on_epoch_end(self, epoch, logs={}):
        y_pred = self.model.predict_generator(self.data_generator,
workers=self.num_workers)
        y_true = self.data_generator.get_labels()
        # estimate AUC under precision recall curve for each class
        pr_auc_mean = self.compute_pr_auc(y_true, y_pred)

        if self.stage == 'val':
            # early stop after early_stopping_patience=4 epochs of no
improvement in mean PR AUC
            self.early_stopping_check(pr_auc_mean)

            # save a model with the best PR AUC in validation
            self.model_checkpoint(pr_auc_mean, epoch)

            # reduce learning rate on PR AUC plateau
            self.reduce_lr_on_plateau()

    def get_pr_auc_history(self):
        return self.history

```

- PR-AUC Callback를 다음과 같은 목적으로 정의
 - Precision-Recall 곡선을 기반으로 각 class의 AUC(Area Under Curve)를 계산

- 5 세대 이상 PR-AUC의 향상이 없을 경우 학습을 조기에 멈춘다
- 검증집합에서 가장 좋은 PR-AUC 값을 보인 모델을 저장
- PR-AUC가 높은 곳에서는 학습률을 줄인다

Callback Instances

```
train_metric_callback = PrAucCallback(data_generator_train_eval)
val_callback = PrAucCallback(data_generator_val, stage='val')
```

Classifier

Defining a Model

```
from keras.losses import binary_crossentropy
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)
```

- competition의 평가방법인 dice coefficient를 기반으로 손실함수 계산법 작성

```
!pip install -U git+https://github.com/qubvel/efficientnet
```

- base model로 사용할 모델 다운로드

```
import efficientnet.keras as efn
def get_model():
    with tf.device('/GPU:0'):
        K.clear_session()
        base_model = efn.EfficientNetB2(weights='imagenet', include_top=False,
pooling='avg', input_shape=(224, 224, 3))
        x = base_model.output
        y_pred = Dense(4, activation='sigmoid')(x)
        return Model(inputs=base_model.input, outputs=y_pred)

model = get_model()
```

- 신경망 모델 정의

```
model.summary()
```

Model: "model_1"

Layer (type)	Output Shape	Param #	Connected to
=====			
input_1 (InputLayer)	(None, 224, 224, 3)	0	
=====			
stem_conv (Conv2D)	(None, 112, 112, 32)	864	input_1[0][0]
=====			
stem_bn (BatchNormalization)	(None, 112, 112, 32)	128	stem_conv[0][0]
=====			
stem_activation (Activation)	(None, 112, 112, 32)	0	stem_bn[0][0]
=====			
...			
(중략)			
...			
=====			
top_bn (BatchNormalization)	(None, 7, 7, 1408)	5632	top_conv[0][0]
=====			
top_activation (Activation)	(None, 7, 7, 1408)	0	top_bn[0][0]
=====			
avg_pool (GlobalAveragePooling2)	(None, 1408)	0	
top_activation[0][0]			
=====			
dense_1 (Dense)	(None, 4)	5636	avg_pool[0][0]
=====			
=====			
Total params: 7,774,198			
Trainable params: 7,706,630			
Non-trainable params: 67,568			
=====			

- 모델의 summary
 - EfficientNetB2를 Imagenet 데이터에 적용한 가중치 사용
 - 마지막 층에 현재 task에 맞춰서 Dense Layer에 노드 수를 4개로 하고 sigmoid를 통과시켜 결과값 출력

```
from keras_radam import RAdam
```

- optimizer로 사용할 RAdam import

Initial tuning of the added fully-connected layer

```
for base_layer in model.layers[:-3]:
    base_layer.trainable = True

model.compile(optimizer=RAdam(warmup_proportion=0.1, min_lr=1e-5),
              loss='categorical_crossentropy', metrics=['accuracy'])
history_1 = model.fit_generator(generator=data_generator_train,
                               validation_data=data_generator_val,
                               epochs=20,
                               callbacks=[train_metric_callback, val_callback],
                               workers=num_cores,
                               verbose=1,
                               initial_epoch=1
                               )
```

```
Epoch 2/20
138/138 [=====] - 335s 2s/step - loss: 2.8027 - acc:
0.3707 - val_loss: 2.7250 - val_acc: 0.4026
```

```
#####
```

```
PR AUC Fish, train: 0.735
```

```
PR AUC Flower, train: 0.866
```

```
PR AUC Sugar, train: 0.839
```

```
PR AUC Gravel, train: 0.774
```

```
#####
```

```
PR AUC mean, train: 0.803
```

```
#####
```

```
#####
```

```
PR AUC Fish, val: 0.686
```

```
PR AUC Flower, val: 0.831
```

```
PR AUC Sugar, val: 0.815
```

```
PR AUC Gravel, val: 0.772
```

```
#####
```

```
PR AUC mean, val: 0.776
```

```
#####
```

```
#####
```

```
Saved new checkpoint
```

```
#####
```

```

...
(종료)
...

Epoch 17/20
138/138 [=====] - 278s 2s/step - loss: 2.1722 - acc:
0.5111 - val_loss: 3.1151 - val_acc: 0.4706

#####

PR AUC Fish, train: 0.947

PR AUC Flower, train: 0.966

PR AUC Sugar, train: 0.953

PR AUC Gravel, train: 0.950

#####
    PR AUC mean, train: 0.954
#####

#####

PR AUC Fish, val: 0.761

PR AUC Flower, val: 0.882

PR AUC Sugar, val: 0.886

PR AUC Gravel, val: 0.799

#####
    PR AUC mean, val: 0.832
#####

```

- 훈련집합으로 최대 20세대까지 먼저 훈련(위 결과에서는 17번째 세대에서 early stop)

Fine-tuning the whole model

- 조금 더 낮은 학습률로 다시 학습(최대 100세대)

Visualizing train and val PR AUC

```

def plot_with_dots(ax, np_array):
    ax.scatter(list(range(1, len(np_array) + 1)), np_array, s=50)
    ax.plot(list(range(1, len(np_array) + 1)), np_array)

pr_auc_history_train = train_metric_callback.get_pr_auc_history()
pr_auc_history_val = val_callback.get_pr_auc_history()

plt.figure(figsize=(10, 7))
plot_with_dots(plt, pr_auc_history_train[-1])

```



```

plot_with_dots(plt, pr_auc_history_val[-1])

plt.xlabel('Epoch', fontsize=15)
plt.ylabel('Mean PR AUC', fontsize=15)
plt.legend(['Train', 'Val'])
plt.title('Training and Validation PR AUC', fontsize=20)
plt.savefig('pr_auc_hist.png')

```



Selecting Postprocessing Thresholds

```

model = load_model('./drive/My
Drive/4classifier_densenet169_epoch_21_val_pr_auc_0.8365921057512743.h5')

```

- submission에서는 pre-trained 모델을 사용

```

class_names = ['Fish', 'Flower', 'Sugar', 'Gravel']
def get_threshold_for_recall(y_true, y_pred, class_i, recall_threshold=0.94,
precision_threshold=0.90, plot=False):
    precision, recall, thresholds = precision_recall_curve(y_true[:, class_i],
y_pred[:, class_i])
    i = len(thresholds) - 1
    best_recall_threshold = None
    while best_recall_threshold is None:
        next_threshold = thresholds[i]
        next_recall = recall[i]
        if next_recall >= recall_threshold:
            best_recall_threshold = next_threshold
        i -= 1

    # concise, even though unnecessary passing through all the values
    best_precision_threshold = [thres for prec, thres in zip(precision,
thresholds) if prec >= precision_threshold][0]

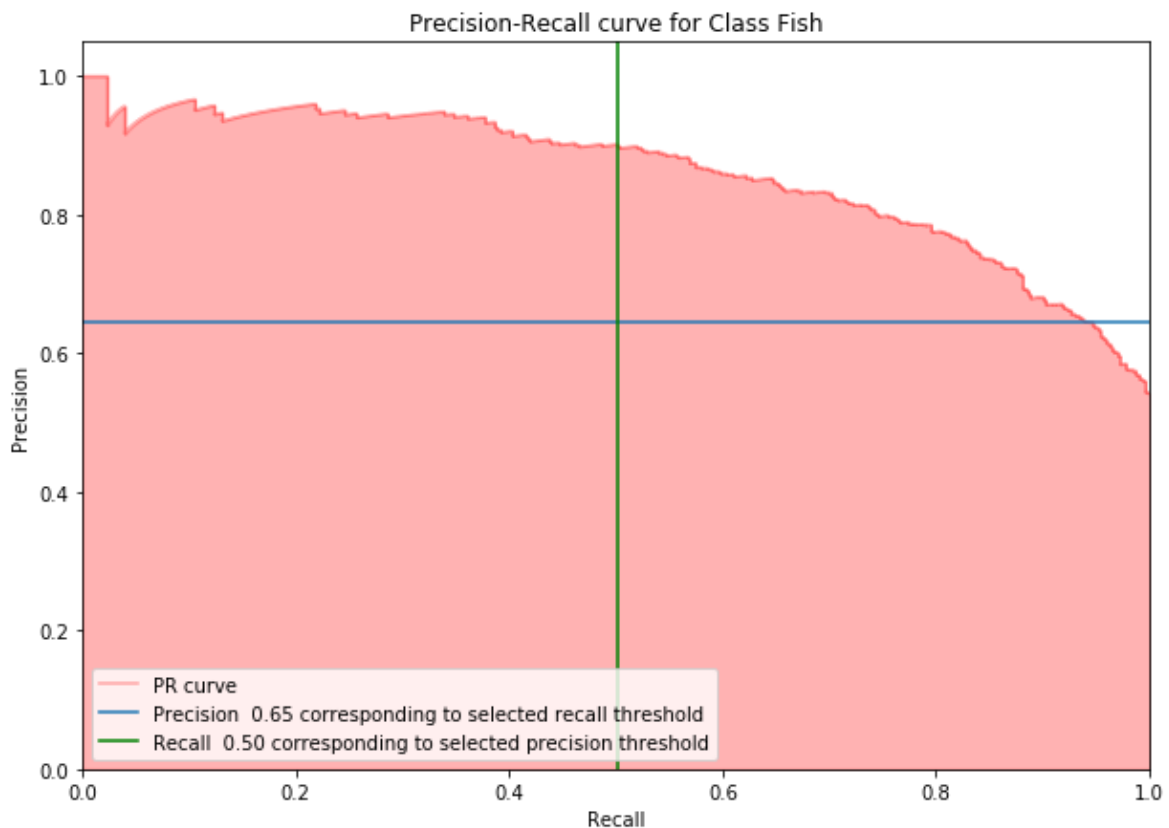
```

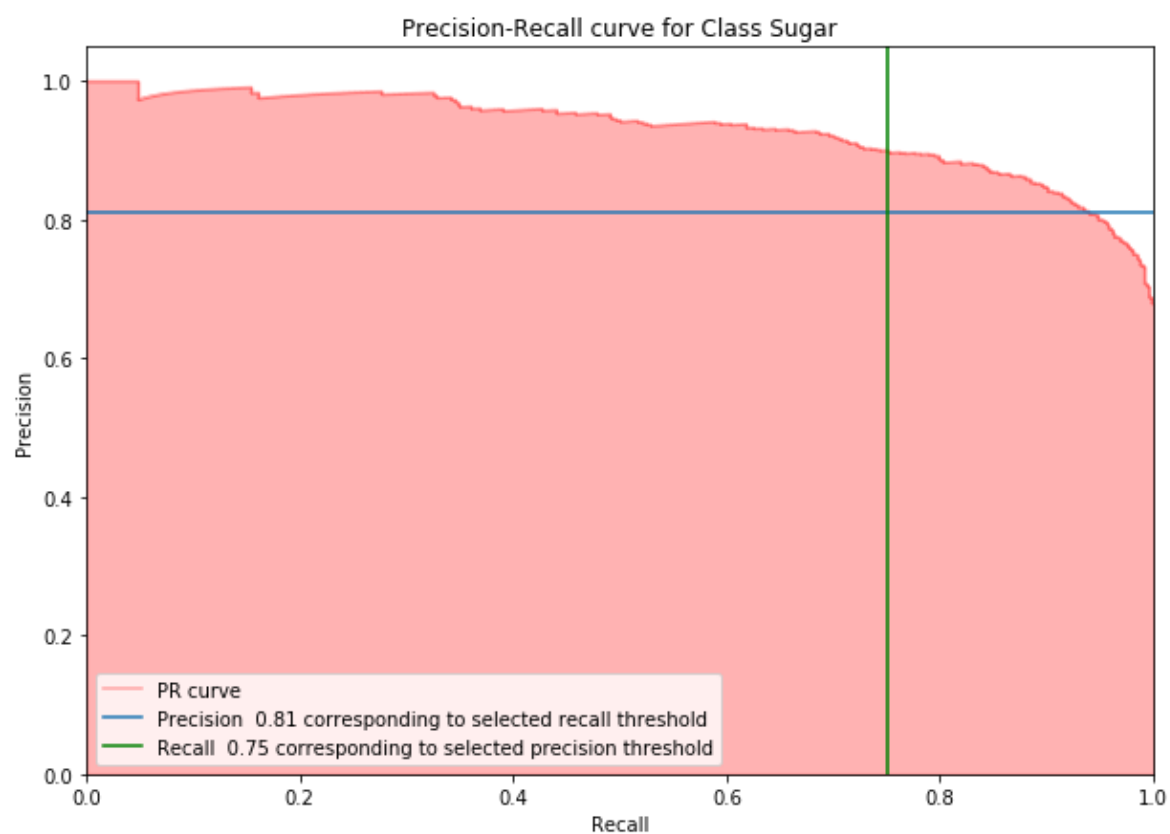
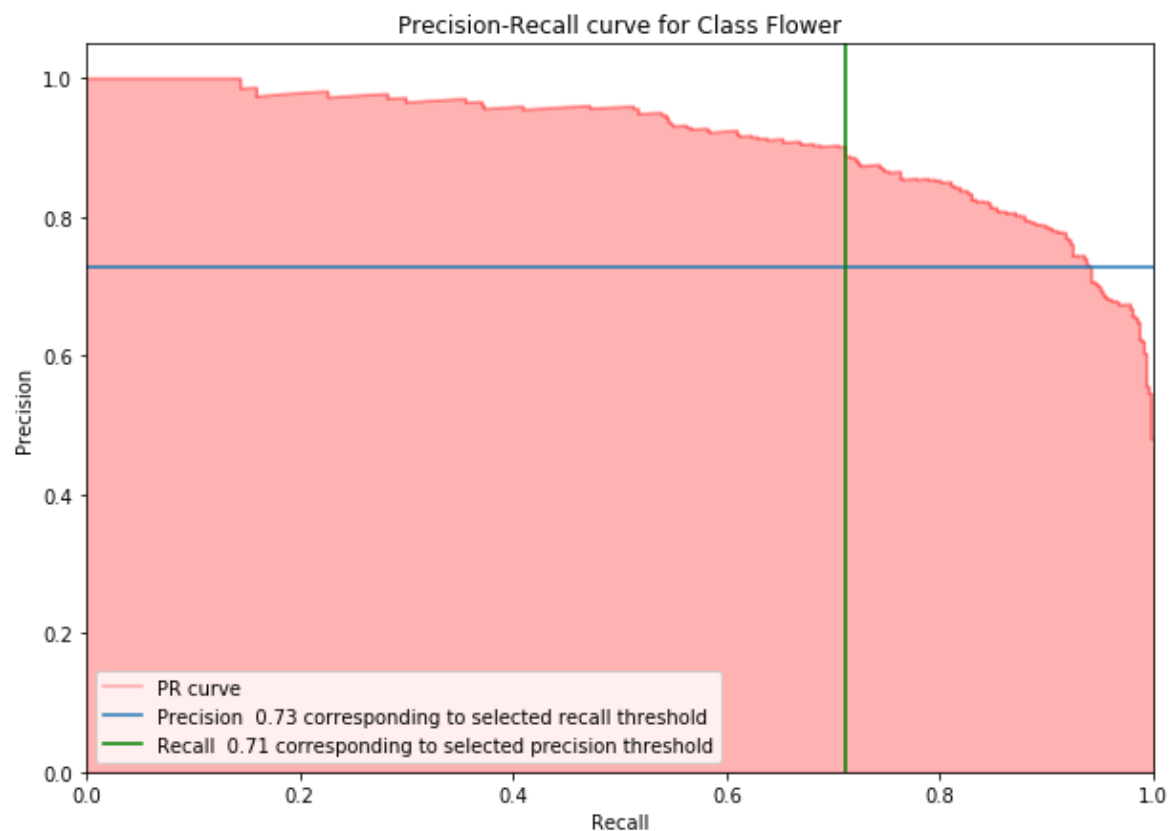
```

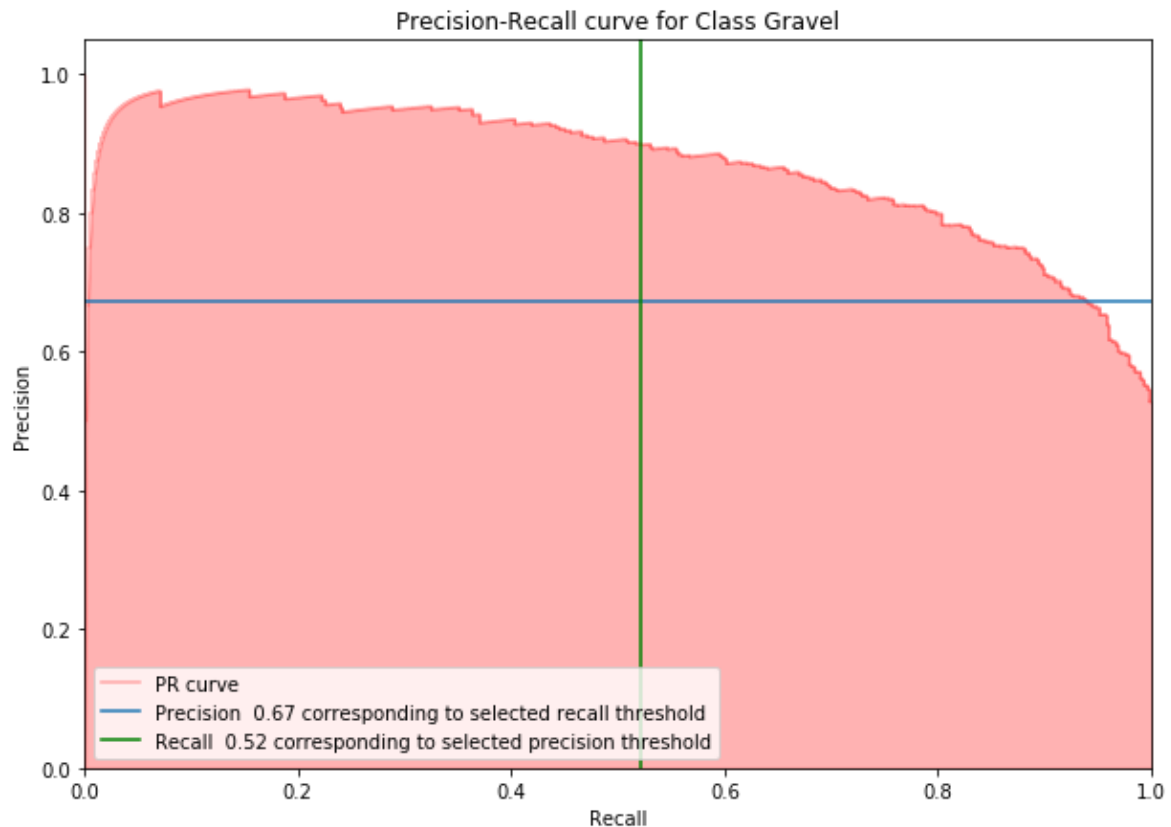
if plot:
    plt.figure(figsize=(10, 7))
    plt.step(recall, precision, color='r', alpha=0.3, where='post')
    plt.fill_between(recall, precision, alpha=0.3, color='r')
    plt.axhline(y=precision[i + 1])
    recall_for_prec_thres = [rec for rec, thres in zip(recall, thresholds)
                             if thres == best_precision_threshold][0]
    plt.axvline(x=recall_for_prec_thres, color='g')
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])
    plt.legend(['PR curve',
               f'Precision {precision[i + 1]: .2f} corresponding to '
               f'selected recall threshold',
               f'Recall {recall_for_prec_thres: .2f} corresponding to '
               f'selected precision threshold'])
    plt.title(f'Precision-Recall curve for Class {class_names[class_i]}')
    return best_recall_threshold, best_precision_threshold

y_pred = model.predict_generator(data_generator_val, workers=num_cores)
y_true = data_generator_val.get_labels()
recall_thresholds = dict()
precision_thresholds = dict()
for i, class_name in tqdm(enumerate(class_names)):
    recall_thresholds[class_name], precision_thresholds[class_name] =
    get_threshold_for_recall(y_true, y_pred, i, plot=True)

```







- PR Curve를 기반으로 각 Class의 최적 Threshold 도출

Post-processing segmentation submission

```
data_generator_test = DataGenerator(folder_imgs=test_imgs_folder,
shuffle=False)
y_pred_test = model.predict_generator(data_generator_test, workers=num_cores)
```

- test image 전처리

```
image_labels_empty = set()
for i, (img, predictions) in enumerate(zip(os.listdir(test_imgs_folder),
y_pred_test)):
    for class_i, class_name in enumerate(class_names):
        if predictions[class_i] < recall_thresholds[class_name]:
            image_labels_empty.add(f'{img}_{class_name}')
```

- 각각 Image_Label들 정리해서 추가

```
recall_thresholds
```

```
{'Fish': 0.2712186,
'Flower': 0.22801664,
'Gravel': 0.2691737,
'Sugar': 0.45162806}
```

- 위에서 도출한 class 별 최적 threshold

```
submission = pd.read_csv('./drive/My Drive/Colab
Notebooks/input/submission_segmentation_and_classifier.csv')
```

- Threshold 안했던 submission 파일 불러오기

```
predictions_nonempty = set(submission.loc[
    ~submission['EncodedPixels'].isnull(),
    'Image_Label'].values)
```

```
#removing masks
submission.loc[submission['Image_Label'].isin(image_labels_empty),
    'EncodedPixels'] = np.nan
submission.to_csv('new3.csv', index=None)
```

- class 별 threshold값에 따라 해당하는 image_label의 세그멘테이션 데이터를 제거하고, csv 파일로 재생성