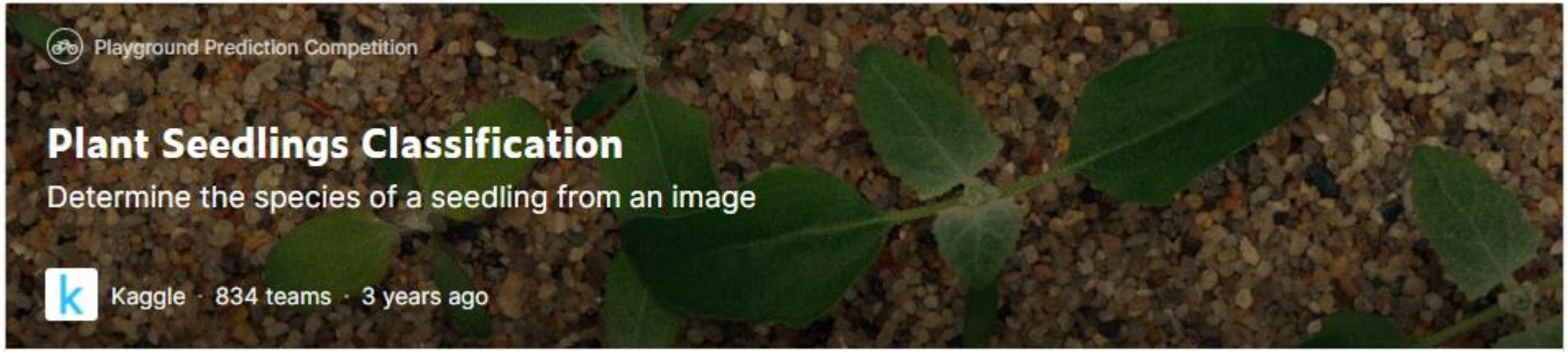
Plant Seedlings Classification

오서영, 허지혜

1. CNN

Plant Seedlings Classification

- The goal of the competition is to create a classifier capable of determining a plant's species from a photo.
- [Plant Seedlings Classification], https://www.kaggle.com/c/plant-seedlings-classification/overview



1. CNN

0. Import Packages

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import os
import sys
import tarfile
import glob
from six.moves import urllib
import random
import shutil
from PIL import Image
import imageio
from PIL import Image, ImageOps
# from scipy.misc import imresize
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer

import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.models import load_model
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
```

1. Load Dataset

- Convert PNG to JPG

```
def convert_jpg(dir, save_dir):

    data = []
    cat_list = os.listdir(dir)
    cat_len = len(os.listdir(dir))

    for cat in cat_list :

        category = os.listdir(dir + cat)
        i = 0
        for name in category :
            png = imageio.imread(dir + cat + '/' + name)
            png = Image.fromarray(png)
            png.load()  # for splitting

            # convert RGBA to RGB -> alpha channel
            if(len(png.split()) == 4):
                img = Image.new('RGB', png.size, (255, 255, 255)) # white
                img.paste(png, mask = png.split()[3])
            else:
                img = png

            img.save(save_dir + cat + '/' + str(i) + '.jpg')
            i += 1
```

1. CNN

```
dir = "train/"
cat_list = os.listdir(dir)
cat_len = len(os.listdir(dir))

print("The number of category :",cat_len)
print(cat_list)
```

```
The number of category : 12
['Black-grass', 'Charlock', 'Cleavers', 'Common Chickweed', 'Common wheat',
 'Fat Hen', 'Loose Silky-bent', 'Maize', 'Scentless Maywee
d', 'Shepherds Purse', 'Small-flowered Cranesbill', 'Sugar beet']
```

```
# Convert png to jpg
dir = "train/"
save_dir = "train_jpg/"
convert_jpg(dir, save_dir)
```

1. CNN

2. Explore Dataset

```
def load_data_files(base_dir):
    folder_name = "train_jpg"
    RAW_DATASET = os.path.join(base_dir, folder_name)

    abs_dir = os.path.join(os.getcwd(), folder_name)
    sub_dir = os.listdir(abs_dir)
    data_dic = {}
    cat_len = []
    for class_name  in sub_dir:
        imgs = glob(os.path.join(RAW_DATASET,class_name,"*.jpg"))

        data_dic[class_name] = imgs
        cat_len.append(len(imgs))
        print("Class: {}".format(class_name))
        print("Number of images: {} ₩n".format(len(imgs)))

    return data_dic, cat_len
```

```
BASE_DIR = os.getcwd()
data_dic, cat_len = load_data_files(BASE_DIR)
```

```
Class: Black-grass
Number of images: 263

Class: Charlock
Number of images: 390

Class: Cleavers
Number of images: 287

Class: Common Chickweed
Number of images: 611

Class: Common wheat
Number of images: 221

Class: Fat Hen
Number of images: 475
```

1. CNN

```
def plot_image_grid(images_files):
    # figure size
    fig = plt.figure(figsize = (8, 8))

    # load images
    images = [tf.keras.preprocessing.image.load_img(img) for img in images_files]

    # plot image grid
    for x in range(4):
        ax = fig.add_subplot(1, 4, x+1)
        plt.imshow(images[x])
        plt.xticks(np.array([]))
        plt.yticks(np.array([]))
    plt.show()
```

```
for class_name, imgs in data_dic.items():
    print("Seed type: {}".format(class_name))
    plot_image_grid(imgs[:16])
```

Seed type: Black-grass

Seed type: Charlock

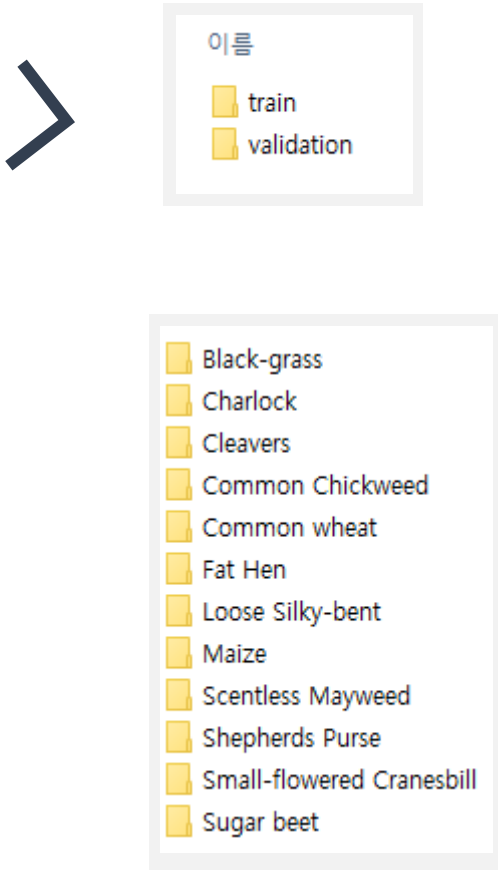
1. CNN

3. Make Dataset

```
# Create new directory and copy files to it
def copy_files_to_directory(files, directory):
    if not os.path.exists(directory):
        os.makedirs(directory)
        print("Created directory: {}".format(directory))

    for f in files:
        shutil.copy(f, directory)
    print("Copied {} files.₩n".format(len(files)))
```

```
def train_validation_split(base_dir, data_dic, split_ratio=0.2):
    FLOWER_DATASET = os.path.join(base_dir,"train_jpg")

    if not os.path.exists(FLOWER_DATASET):
        os.makedirs(FLOWER_DATASET)

    for class_name, imgs in data_dic.items():
        idx_split = int(len(imgs) * split_ratio)
        random.shuffle(imgs)
        validation = imgs[:idx_split]
        train = imgs[idx_split:]

        copy_files_to_directory(train, os.path.join(FLOWER_DATASET,"train",class_name))
        copy_files_to_directory(validation, os.path.join(FLOWER_DATASET,"validation",class_name))
```

```
BASE_DIR = os.getcwd()

train_validation_split(BASE_DIR, data_dic, split_ratio = 0.2)
```



1. CNN

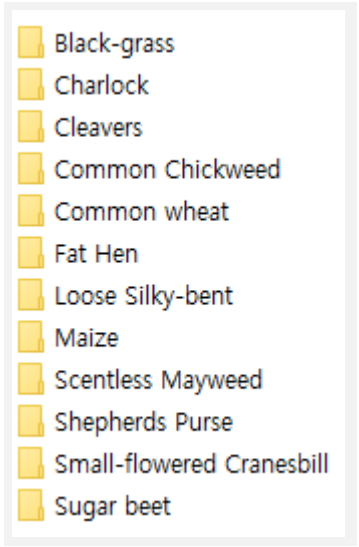
```
# params
batch_size = 32
num_classes = 12
epochs = 50
```

```
preprocessing_image = tf.keras.preprocessing.image

train_datagen = preprocessing_image.ImageDataGenerator(
    rescale=1./255,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True)

val_datagen = preprocessing_image.ImageDataGenerator(rescale=1./255)
```

```
BASE_DIR = os.getcwd()

train_generator = train_datagen.flow_from_directory(
    os.path.join(BASE_DIR, "train_jpg/train"),
    target_size=(32, 32),
    batch_size=batch_size,
    class_mode='categorical')

validation_generator = val_datagen.flow_from_directory(
    os.path.join(BASE_DIR, "train_jpg/validation"),
    target_size=(32, 32),
    batch_size=batch_size,
    class_mode='categorical')
```

```
Found 3803 images belonging to 12 classes.
Found 947  mages belonging to 12 classes.
```

flow_from_directory !



1. CNN

4-1. Baseline CNN

```
model = Sequential()
model.add(Conv2D(64, kernel_size=(5, 5), strides=(1, 1), padding='same',
                 activation='relu',
                 input_shape=(32, 32, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (5, 5), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_11 (Conv2D)           (None, 32, 32, 64)        4864
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 16, 16, 32)        51232
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 8, 8, 32)          0
_________________________________________________________________
dropout_11 (Dropout)         (None, 8, 8, 32)          0
_________________________________________________________________
flatten_6 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_11 (Dense)             (None, 1024)              2098176
_________________________________________________________________
dropout_12 (Dropout)         (None, 1024)              0
_________________________________________________________________
dense_12 (Dense)             (None, 12)                12300
=================================================================
Total params: 2,166,572
Trainable params: 2,166,572
Non-trainable params: 0
```

1. CNN

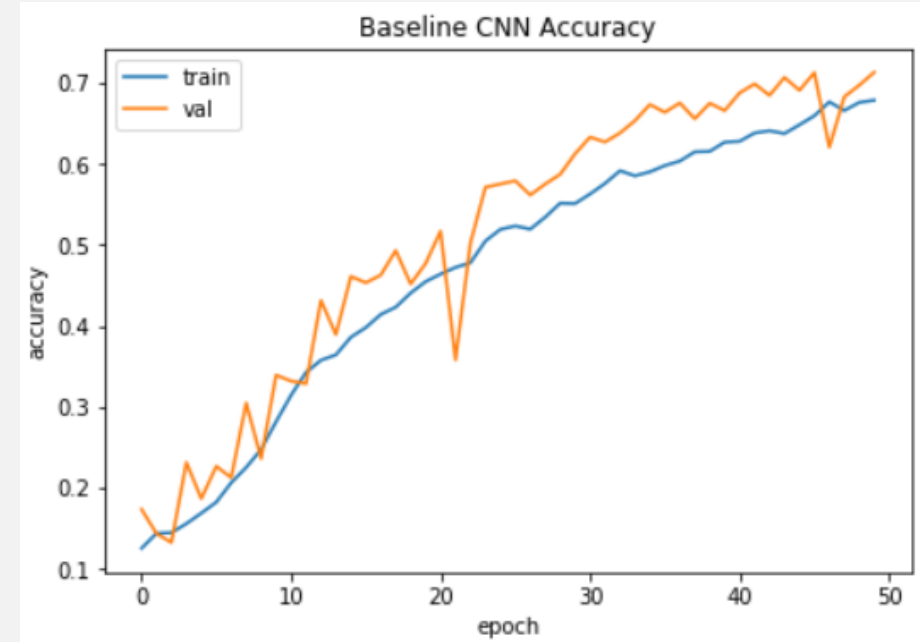
```
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
```

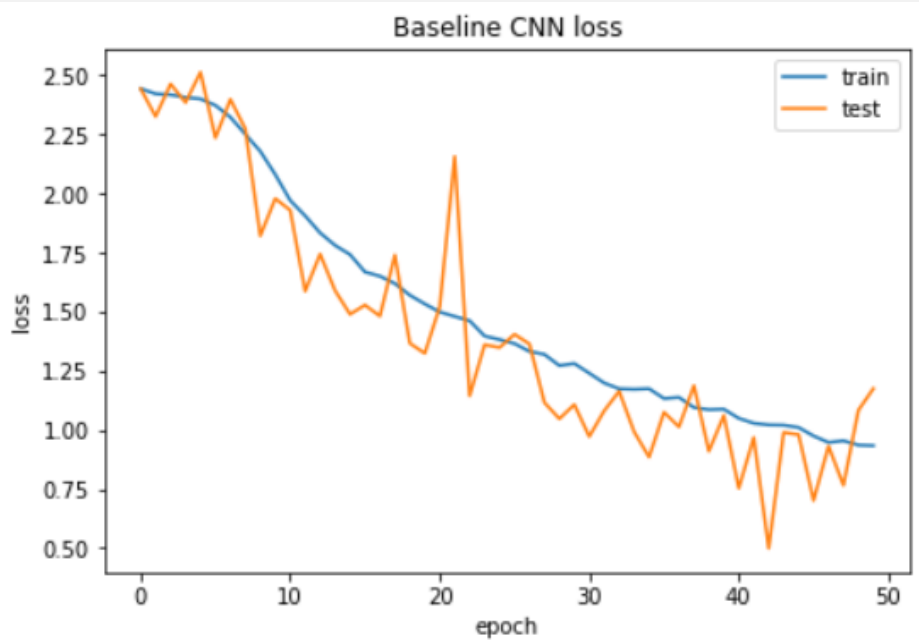
```
%%time
hist50 = model.fit_generator(
    train_generator,
    steps_per_epoch=3803//batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=20)
```

```
def plot_accuracy_and_loss(history):
    plt.figure(1, figsize= (15, 10))

    # plot train and test accuracy
    plt.subplot(221)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Baseline CNN Accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')

    # plot train and test loss
    plt.subplot(222)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Baseline CNN loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper right')

    plt.show()
```

```
plot_accuracy_and_loss(hist50)
```

1. CNN


Baseline CNN Accuracy
train
val
accuracy
epoch
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
10
20
30
40
50


Baseline CNN loss
train
test
loss
epoch
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2.25
2.50
0
10
20
30
40
50

1. CNN

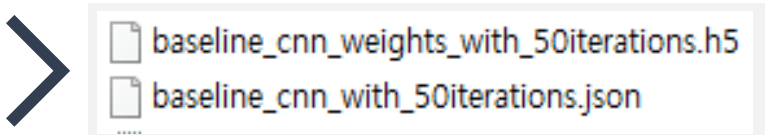
```
print("-- Evaluate --")

scores_train = model2.evaluate_generator(
            train_generator,
            steps = 5)
scores_val = model2.evaluate_generator(
            validation_generator,
            steps = 5)

print("Train " + "%s: %.2f%%" %(model2.metrics_names[1], scores_train[1]*100))
print("Val " + "%s: %.2f%%" %(model2.metrics_names[1], scores_val[1]*100))

print("-- Predict --")
output_train = model2.predict_generator(train_generator, steps=5)
output_val = model2.predict_generator(validation_generator, steps=5)
np.set_printoptions(formatter={'float': lambda x: "{0:0.3f}".format(x)})

print(train_generator.class_indices)
print(output_train)

print(validation_generator.class_indices)
print(output_val)
```

```
-- Evaluate --
Train accuracy: 72.50%
Val accuracy: 69.38%
-- Predict --
{'Black-grass': 0, 'Charlock': 1, 'Cleavers': 2,
e': 7, 'Scentless Mayweed': 8, 'Shepherds Purse'
[[0.000 0.000 0.000 ... 0.001 0.995 0.003]
 [0.617 0.000 0.001 ... 0.000 0.000 0.001]
 [0.001 0.064 0.060 ... 0.017 0.002 0.380]
 ...
 [0.000 0.011 0.040 ... 0.016 0.021 0.001]
 [0.395 0.000 0.000 ... 0.000 0.000 0.001]
 [0.000 0.001 0.004 ... 0.001 0.001 0.835]]
{'Black-grass': 0, 'Charlock': 1, 'Cleavers': 2,
e': 7, 'Scentless Mayweed': 8, 'Shepherds Purse'
[[0.000 0.076 0.003 ... 0.543 0.194 0.000]
 [0.011 0.003 0.178 ... 0.002 0.001 0.161]
 [0.000 0.043 0.011 ... 0.142 0.012 0.054]
 ...
 [0.020 0.064 0.598 ... 0.003 0.002 0.038]
 [0.000 0.000 0.000 ... 0.035 0.006 0.000]
 [0.313 0.000 0.000 ... 0.000 0.000 0.000]]
```

1. CNN

```
# save model architecture
model_json = model2.to_json()
open('4layer_cnn_with_50iterations.json', 'w').write(model_json)

# save model's learned weights
model2.save_weights('4layer_cnn_weights_with_50iterations.h5', overwrite=True)
```

```
# Load trained model
from keras.models import model_from_json

json_file = open("baseline_cnn_with_10iterations.json", "r")
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)

# model weight load
loaded_model.load_weights("baseline_cnn_weights_with_10iterations.h5")
print("Loaded model from disk")
```



1. CNN

5. Submission

```
z = glob.glob('test/*.png')
test_imgs = []
names = []
for fn in z:
    if fn[-3:] != 'png':
        continue
    names.append(fn.split('test₩₩')[-1])
#     print(names)
    new_img = Image.open(fn)
    test_img = ImageOps.fit(new_img, (32,32), Image.ANTIALIAS).convert('RGB')
    test_imgs.append(test_img)
```

```
test_img = np.array([np.array(im) for im in test_imgs])
test_x = test_img.reshape(test_img.shape[0], 32, 32, 3) / 255

test_x.shape
```

```
(794, 32, 32, 3)
```

```
lb = LabelBinarizer().fit(names)

pred = loaded_model2.predict(test_x)
# print(pred)
prediction = lb.inverse_transform(pred)
```

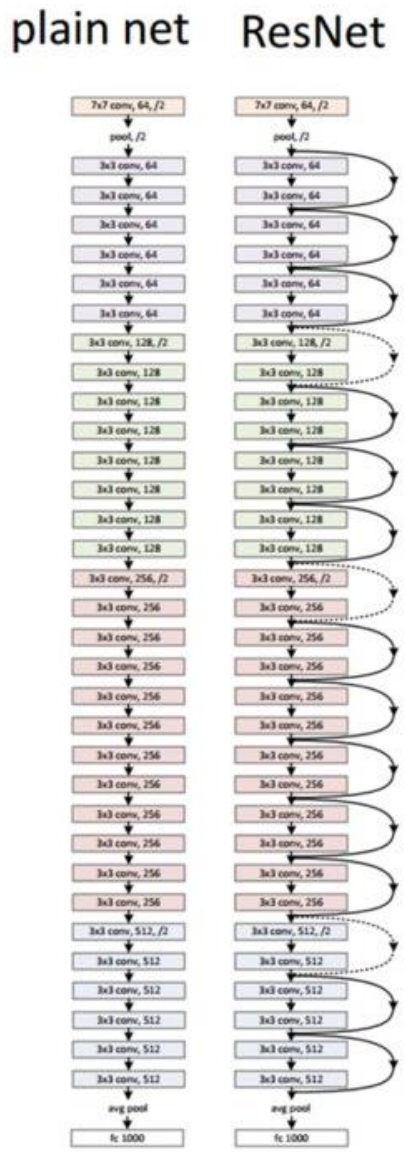
```
df = pd.DataFrame(data={'file': names, 'species': prediction})
df_sort = df.sort_values(by=['file'])
df_sort.to_csv('CNN_with_50iterations.csv', index=False)
```

CNN_with_50iterations.csv — 0.77707

6 days ago by Seoyoung Oh

add submission details

2. Resnet



Resnet

- Ultra deep : 152 layer
- 더 깊은 망이 학습결과를 더 좋게한다

-> but 학습을 시키기가 점점 더 어려워짐
(vanishing gradient, increase error)

-> **Residual Learning**

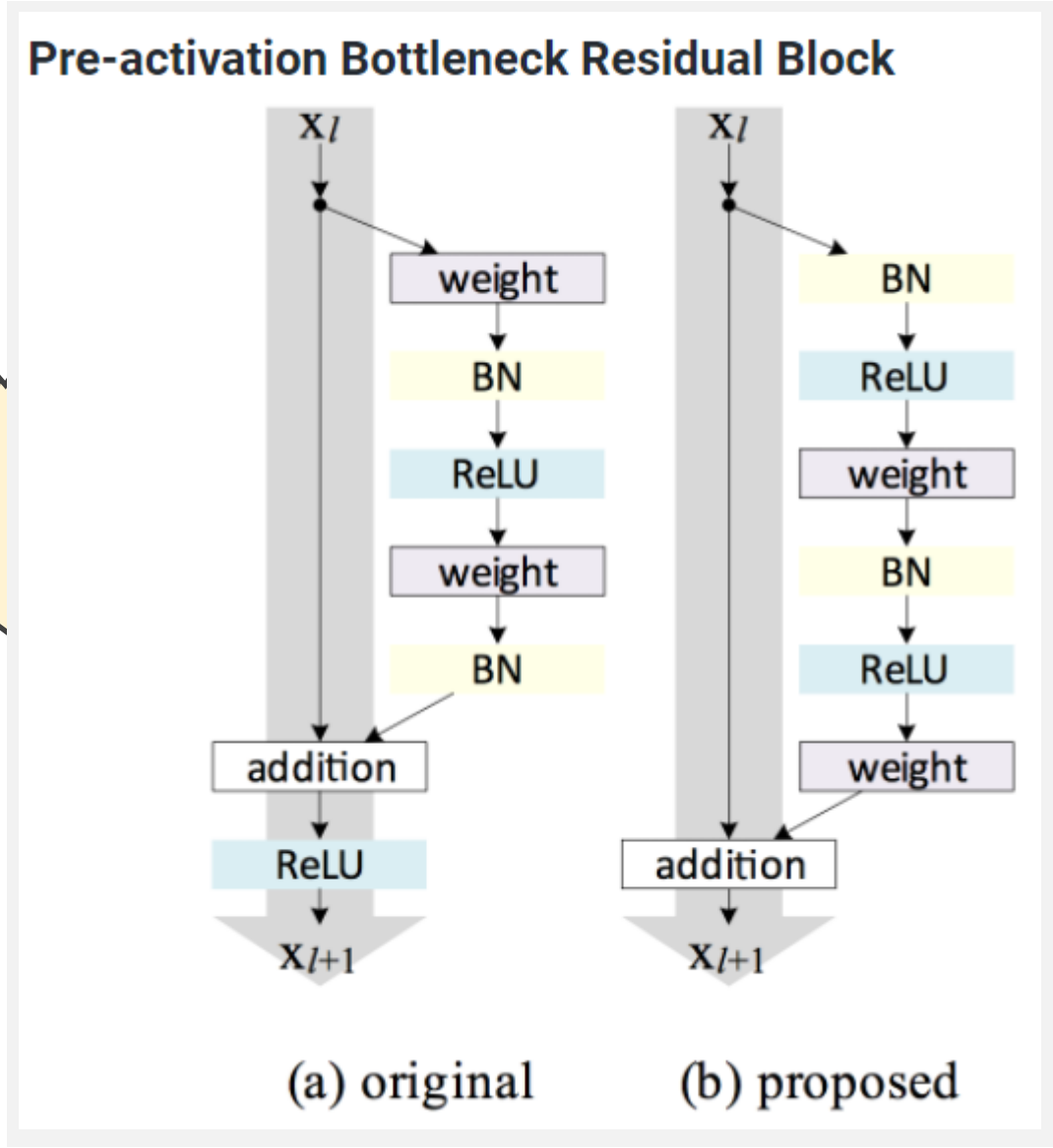
: 입력에서 바로 출력으로 연결되는 shortcut

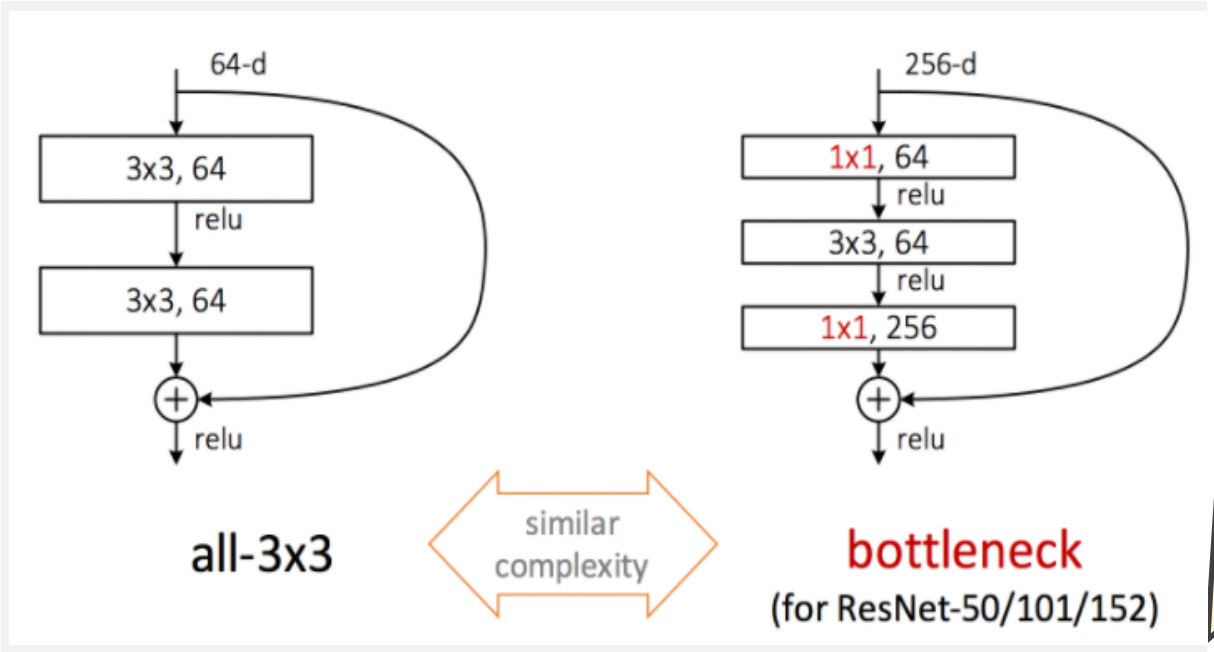
-> 파라미터 없이 바로 연결되는 구조

-> 연산량 관점에서는 덧셈만 추가됨

1. 깊은 망도 쉽게 최적화가 가능하다
2. 늘어난 깊이로 인해 정확도를 개선할 수 있다.

2. Resnet





Deeper Bottleneck Architecture

- Bottleneck

: 차원을 줄였다가 늘리는 모습이 병목처럼 보임'

-> 연산시간을 줄이기 위함

2. Resnet

3. Resnet (Deep Residual Neural Network)

- Pre-activation Bottleneck Residual Block

```
models = tf.keras.models
layers = tf.keras.layers
initializers = tf.keras.initializers
regularizers = tf.keras.regularizers
losses = tf.keras.losses
optimizers = tf.keras.optimizers
metrics = tf.keras.metrics
```

```
def residual_block(input_tensor, filters, stage, reg=0.0, use_shortcuts=True):

    bn_name = 'bn' + str(stage)
    conv_name = 'conv' + str(stage)
    relu_name = 'relu' + str(stage)
    merge_name = 'merge' + str(stage)

    # 1x1 conv
    # batchnorm-relu-conv
    # from input_filters to bottleneck_filters

    if stage>1: # first activation is just after conv1
        x = layers.BatchNormalization(name=bn_name+'a')(input_tensor)
        x = layers.Activation('relu', name=relu_name+'a')(x)
    else:
        x = input_tensor

    x = layers.Convolution2D(
            filters[0], (1,1),
            kernel_regularizer=regularizers.l2(reg),
            use_bias=False,
            name=conv_name+'a'
        )(x)
```

2. Resnet

```
# 3x3 conv
# batchnorm-relu-conv
# from bottleneck_filters to bottleneck_filters

x = layers.BatchNormalization(name=bn_name+'b')(x)
x = layers.Activation('relu', name=relu_name+'b')(x)
x = layers.Convolution2D(
        filters[1], (3,3),
        padding='same',
        kernel_regularizer=regularizers.l2(reg),
        use_bias = False,
        name=conv_name+'b'
    )(x)

# 1x1 conv
# batchnorm-relu-conv
# from bottleneck_filters  to input_filters

x = layers.BatchNormalization(name=bn_name+'c')(x)
x = layers.Activation('relu', name=relu_name+'c')(x)
x = layers.Convolution2D(
        filters[2], (1,1),
        kernel_regularizer=regularizers.l2(reg),
        name=conv_name+'c'
    )(x)

# merge output with input layer (residual connection)

if use_shortcuts:
    x = layers.add([x, input_tensor], name=merge_name)

return x
```

- Full Residual Network

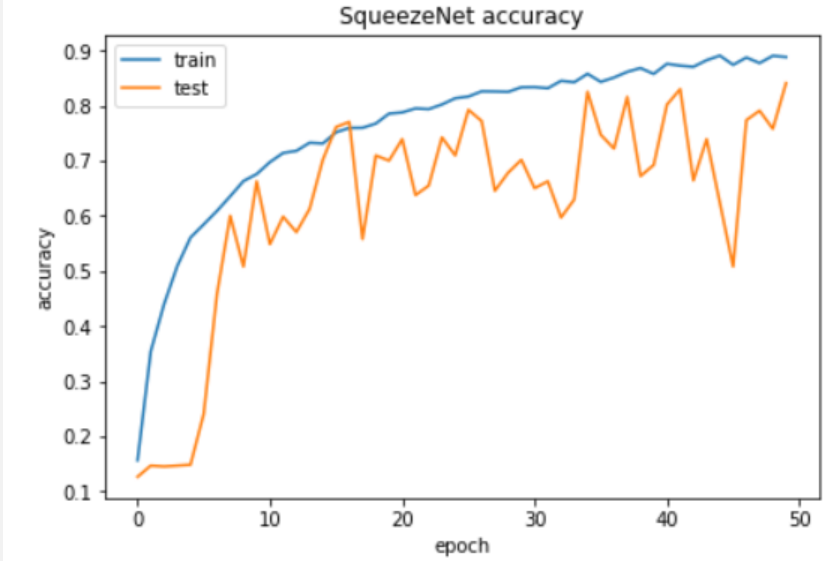
```
def ResNetPreAct(input_shape=(32,32,3), nb_classes=5, num_stages=5,
                 use_final_conv=False, reg=0.0):


    # Input
    img_input = layers.Input(shape=input_shape)

    #### Input stream ####
    # conv-BN-relu-(pool)

    x = layers.Convolution2D(
            128, (3,3), strides=(2, 2),
            padding='same',
            kernel_regularizer=regularizers.l2(reg),
            use_bias=False,
            name='conv0'
        )(img_input)
    x = layers.BatchNormalization(name='bn0')(x)
    x = layers.Activation('relu', name='relu0')(x)
#     x = layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same', name='pool0')(x)

    #### Residual Blocks ####
    # 1x1 conv: batchnorm-relu-conv
    # 3x3 conv: batchnorm-relu-conv
    # 1x1 conv: batchnorm-relu-conv

    for stage in range(1,num_stages+1):
        x = residual_block(x, [32,32,128], stage=stage, reg=reg)
```

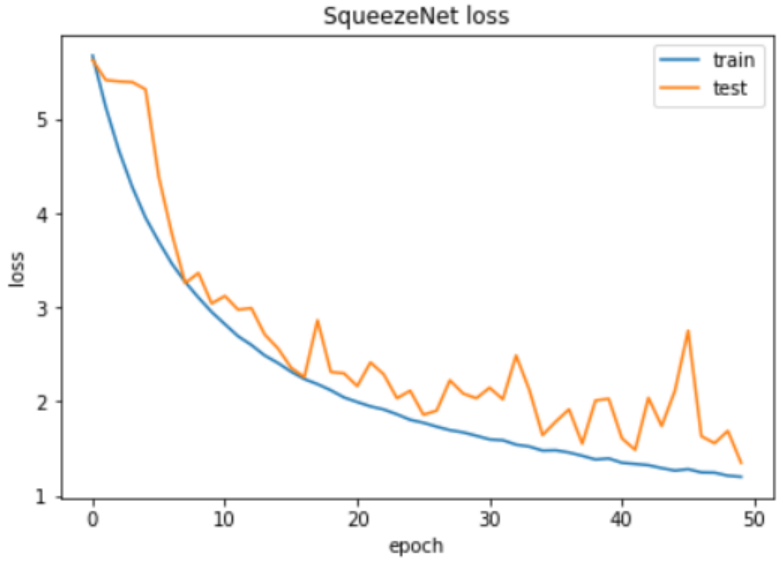
2. Resnet

```
#### Output stream ####
# BN-relu-(conv)-avgPool-softmax

x = layers.BatchNormalization(name='bnF')(x)
x = layers.Activation('relu', name='reluF')(x)

# Optional final conv layer
if use_final_conv:
    x = layers.Convolution2D(
            64, (3,3),
            padding='same',
            kernel_regularizer=regularizers.l2(reg),
            name='convF'
        )(x)

pool_size = input_shape[0] / 2
x = layers.AveragePooling2D((pool_size,pool_size),name='avg_pool')(x)

x = layers.Flatten(name='flat')(x)
x = layers.Dense(nb_classes, activation='softmax', name='fc10')(x)

return models.Model(img_input, x, name='rnpa')
```

```
# params
batch_size = 32
num_classes = 12
epochs = 50
```

2. Resnet






```
-- Evaluate --
Train categorical_accuracy: 86.25%
Val categorical_accuracy: 80.00%
```

Resnet_with_50iterations.csv

14 hours ago by Seoyoung Oh

final

0.86020

3. Background remove

```
path_to_images = '../input/plant-seedlings-classification/train/*/*.png'
images = glob(path_to_images)
trainingset = []
traininglabels = []
num = len(images)
count = 1
#READING IMAGES AND RESIZING THEM
for i in images:
    print(str(count)+'/'+str(num),end='\r')
    trainingset.append(cv2.resize(cv2.imread(i),(scale,scale)))
    traininglabels.append(i.split('/')[-2])
    count=count+1
trainingset = np.asarray(trainingset)
traininglabels = pd.DataFrame(traininglabels)
```

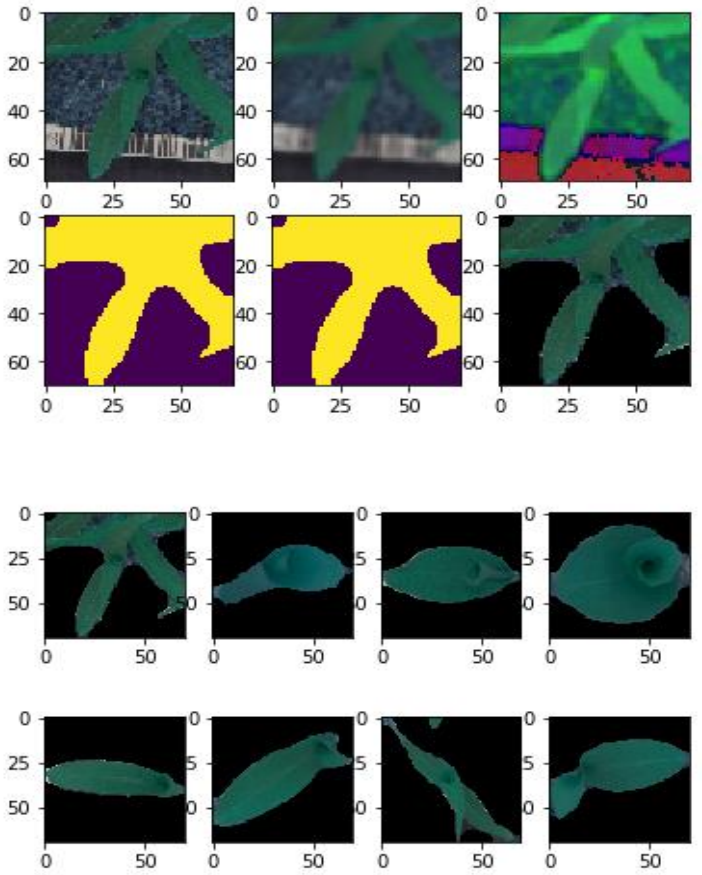
```
4750/4750
```

```
new_train = []
sets = []; getEx = True
for i in trainingset:
    blurr = cv2.GaussianBlur(i,(5,5),0)
    hsv = cv2.cvtColor(blurr,cv2.COLOR_BGR2HSV)
    #GREEN PARAMETERS
    lower = (25,40,50)
    upper = (75,255,255)
    mask = cv2.inRange(hsv,lower,upper)
    struc = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(11,11))
    mask = cv2.morphologyEx(mask,cv2.MORPH_CLOSE,struc)
    boolean = mask>0
    new = np.zeros_like(i,np.uint8)
    new[boolean] = i[boolean]
    new_train.append(new)

    if getEx:
        plt.subplot(2,3,1);plt.imshow(i) # ORIGINAL
        plt.subplot(2,3,2);plt.imshow(blurr) # BLURRED
        plt.subplot(2,3,3);plt.imshow(hsv) # HSV CONVERTED
        plt.subplot(2,3,4);plt.imshow(mask) # MASKED
        plt.subplot(2,3,5);plt.imshow(boolean) # BOOLEAN MASKED
        plt.subplot(2,3,6);plt.imshow(new) # NEW PROCESSED IMAGE
        plt.show()
        getEx = False
new_train = np.asarray(new_train)
```

3. Background remove

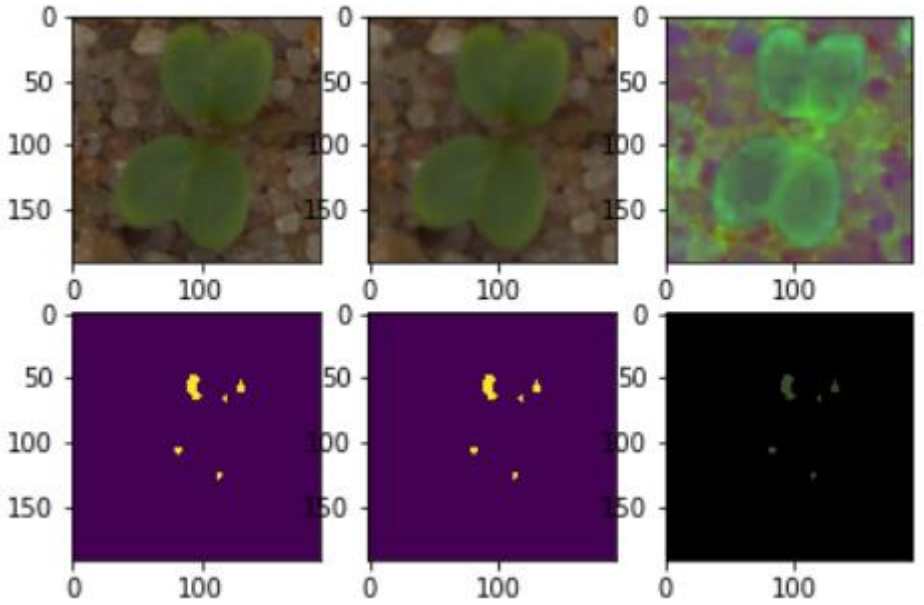
```
new_train = []
sets = []; getEx = True
for i in trainingset:
    blurr = cv2.GaussianBlur(i,(5,5),0)
    hsv = cv2.cvtColor(blurr,cv2.COLOR_BGR2HSV)
    #GREEN PARAMETERS
    lower = (25,40,50)
    upper = (75,255,255)
    mask = cv2.inRange(hsv,lower,upper)
    struc = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(11,11))
    mask = cv2.morphologyEx(mask,cv2.MORPH_CLOSE,struc)
    boolean = mask>0
    new = np.zeros_like(i,np.uint8)
    new[boolean] = i[boolean]
    new_train.append(new)

    if getEx:
        plt.subplot(2,3,1);plt.imshow(i) # ORIGINAL
        plt.subplot(2,3,2);plt.imshow(blurr) # BLURRED
        plt.subplot(2,3,3);plt.imshow(hsv) # HSV CONVERTED
        plt.subplot(2,3,4);plt.imshow(mask) # MASKED
        plt.subplot(2,3,5);plt.imshow(boolean) # BOOLEAN MASKED
        plt.subplot(2,3,6);plt.imshow(new) # NEW PROCESSED IMAGE
        plt.show()
        getEx = False
new_train = np.asarray(new_train)
```

```
# CLEANED IMAGES
for i in range(8):
    plt.subplot(2,4,i+1)
    plt.imshow(new_train[i])
```



3. Background remove

```
import cv2
from glob import glob
import numpy as np
from matplotlib import pyplot as plt
import math
import pandas as pd

new_train = []
sets = [] ; getEx = True
#path = "C:/Users/HOME/Desktop/수DA쟁이/TEAMPROJECT-M/잇/train_jpg/train/Black-grass/0.jpg"
#i = glob(path)
blurr = cv2.GaussianBlur(a,(5,5),0)
hsv = cv2.cvtColor(blurr,cv2.COLOR_BGR2HSV)
    #GREEN PARAMETERS
lower = (20,40,40)
upper = (75,255,255)
mask = cv2.inRange(hsv,lower,upper)
struc = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(11,11))
mask = cv2.morphologyEx(mask,cv2.MORPH_CLOSE,struc)
boolean = mask>0
new = np.zeros_like(a,np.uint8)
new[boolean] = a[boolean]
new_train.append(new)

if getEx:
    plt.subplot(2,3,1);plt.imshow(a) # ORIGINAL
    plt.subplot(2,3,2);plt.imshow(blurr) # BLURRED
    plt.subplot(2,3,3);plt.imshow(hsv) # HSV CONVERTED
    plt.subplot(2,3,4);plt.imshow(mask) # MASKED
    plt.subplot(2,3,5);plt.imshow(boolean) # BOOLEAN MASKED
    plt.subplot(2,3,6);plt.imshow(new) # NEW PROCESSED IMAGE
    plt.show()
    getEx = False
```



Reference

[1] [Advanced Computer Vision with TensorFlow],
https://stephan-osterburg.gitbook.io/coding/coding/ml-dl/tensorfow

[2] [Seeding Classification using CNN(V13 – 0.95)],
https://www.kaggle.com/omkarsabnis/seedling-classification-using-cnn-v13-0-95