Cassava Leaf Disease Classification

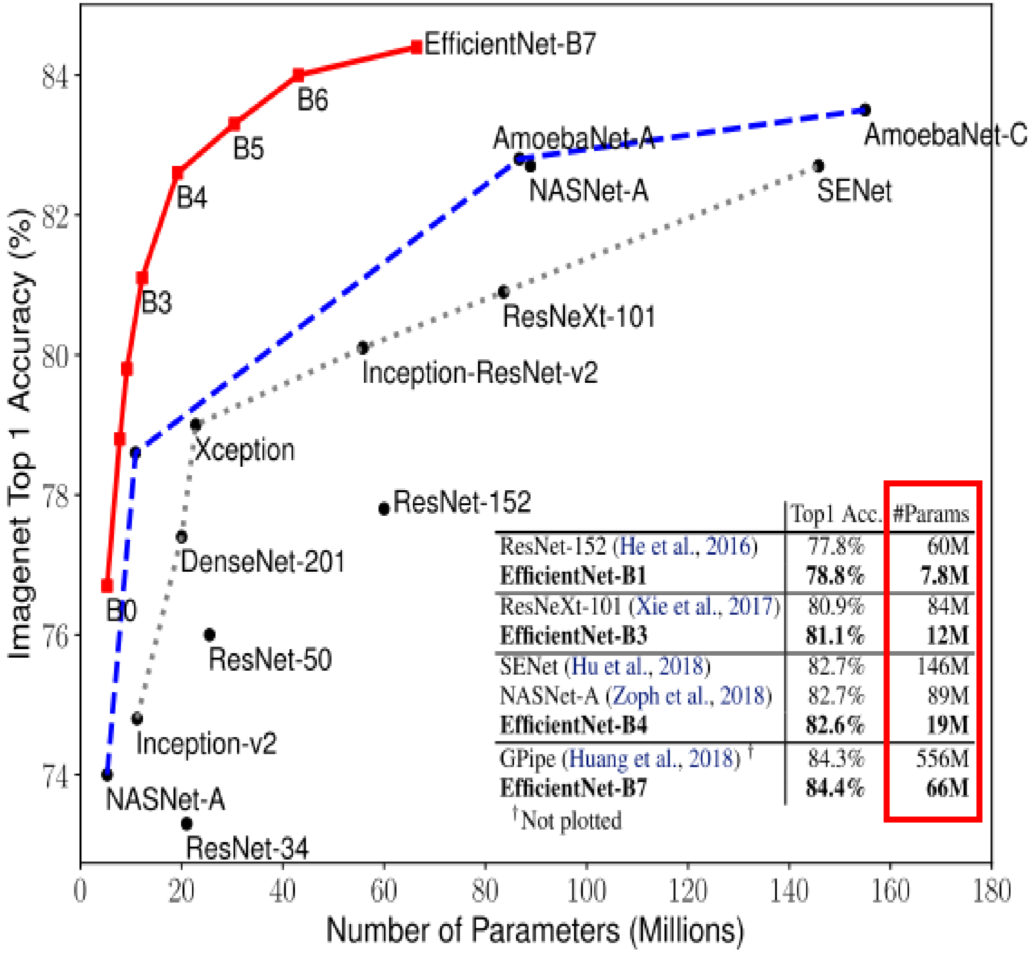
AM20126801 김동현

Contents

Introduction

- Cassava is a key food security crop grown by smallholder farmers in Africa.
- At least 80% of household farms grow this starchy root, but Viral diseases are major sources of poor yields.
- Existing methods of disease detection require farmers to solicit the help of agricultural experts to visually inspect and diagnose the plants.
- In this project, main task is to classify each cassava image into four disease categories or a fifth category indicating a healthy leaf.

EfficientNet



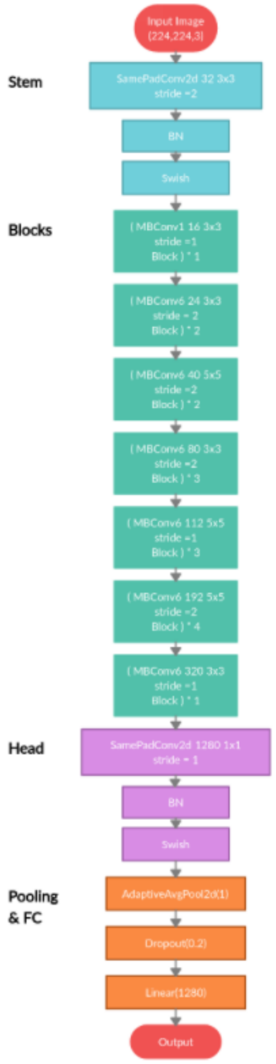
- EfficientNet is a model that achieves state of the art for image classification task by performing better with fewer parameters than before.

- It can be seen that in previous studies, many attempts were made for scaling up to increase the performance of ConvNet.

- For example, Resnet achieved performance improvement by increasing the number of layers.

- EfficientNet is a model that finds the optimal combination for the scale-up method through AutoML.

EfficientNet



Model	Top-1 Acc.	Top-5 Acc.	#Params	Ratio-to-EfficientNet	#FLOPs	Ratio-to-EfficientNet
EfficientNet-B0	**77.1%**	**93.3%**	**5.3M**	**1x**	**0.39B**	**1x**
ResNet-50 (He et al., 2016)	76.0%	93.0%	26M	4.9x	4.1B	11x
DenseNet-169 (Huang et al., 2017)	76.2%	93.2%	14M	2.6x	3.5B	8.9x

Table 1. **EfficientNet-B0 baseline network** – Each row describes a stage i with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$. Notations are adopted from equation 2.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

- EfficientNet uses a mobile inverted bottlenect convolution (MBConV) block as its main
- EfficientNet-B0 is the baseline model with the fewest parameters among the EfficientNet model types.

Dataset

- Cassava image datasets have 5 classes:4 diseases and 1 healthy
  "0": "Cassava Bacterial Blight (CBB)"
  "1": "Cassava Brown Streak Disease (CBSD)"
  "2": "Cassava Green Mottle (CGM)"
  "3": "Cassava Mosaic Disease (CMD)"
  "4": "Healthy"

3. CGM

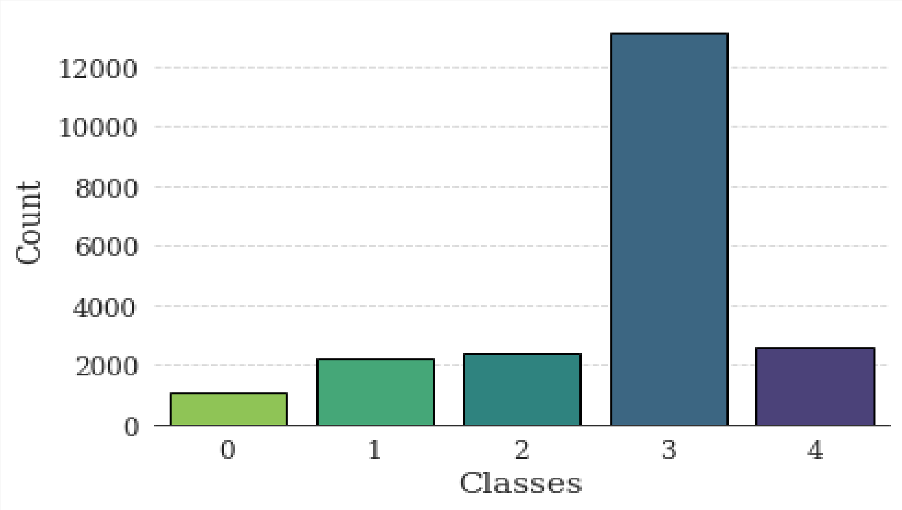
4. Healthy

- Number of train images: 21,397

```
train_labels = pd.read_csv(os.path.join(WORK_DIR, "train.csv"))
train_labels.head()
```

	image_id	label
0	1000015157.jpg	0
1	1000201771.jpg	3
2	100042118.jpg	1
3	1000723321.jpg	1
4	1000812911.jpg	3

- The dataset has a fairly large imbalance.



General Visualization

"0": "Cassava Bacterial Blight (CBB)“

"1": "Cassava Brown Streak Disease (CBSD)"

"2": "Cassava Green Mottle (CGM)“

"3": "Cassava Mosaic Disease (CMD)“

"4": "Healthy“

- All images given are 600x800 size.

Libraries

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import tensorflow as tf
from tensorflow.keras import models, layers
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.optimizers import Adam

# ignoring warnings
import warnings
warnings.simplefilter("ignore")

import os, cv2, json
from PIL import Image
```

- numpy, pandas
- matplotlib, seaborn
- tensorflow
- ReduceLROnPlateau
  - Reduce learning rate when a metric has stopped improving.
- EfficientNetB0
  - This function returns a Keras image classification model, optionally loaded with weights pre-trained on ImageNet.
- cv2
  - Opencv-python is a representative package for image processing and computer vision.

Data augmentation

```
train_labels.label = train_labels.label.astype('str')

train_datagen = ImageDataGenerator(validation_split = 0.2,
                                     preprocessing_function = None,
                                     rotation_range = 45,
                                     zoom_range = 0.2,
                                     horizontal_flip = True,
                                     vertical_flip = True,
                                     fill_mode = 'nearest',
                                     shear_range = 0.1,
                                     height_shift_range = 0.1,
                                     width_shift_range = 0.1)

train_generator = train_datagen.flow_from_dataframe(train_labels,
                         directory = os.path.join(WORK_DIR, "train_images"),
                         subset = "training",
                         x_col = "image_id",
                         y_col = "label",
                         target_size = (TARGET_SIZE, TARGET_SIZE),
                         batch_size = BATCH_SIZE,
                         class_mode = "sparse")


validation_datagen = ImageDataGenerator(validation_split = 0.2)

validation_generator = validation_datagen.flow_from_dataframe(train_labels,
                         directory = os.path.join(WORK_DIR, "train_images"),
                         subset = "validation",
                         x_col = "image_id",
                         y_col = "label",
                         target_size = (TARGET_SIZE, TARGET_SIZE),
                         batch_size = BATCH_SIZE,
                         class_mode = "sparse")
```

- The ImageDataGenerator class in Keras is one of the packages to make learning image data easier.
- It is used to increase the accuracy of the model by increasing the data by giving a slight transformation to the original image.

< ImageDataGenerator Result >

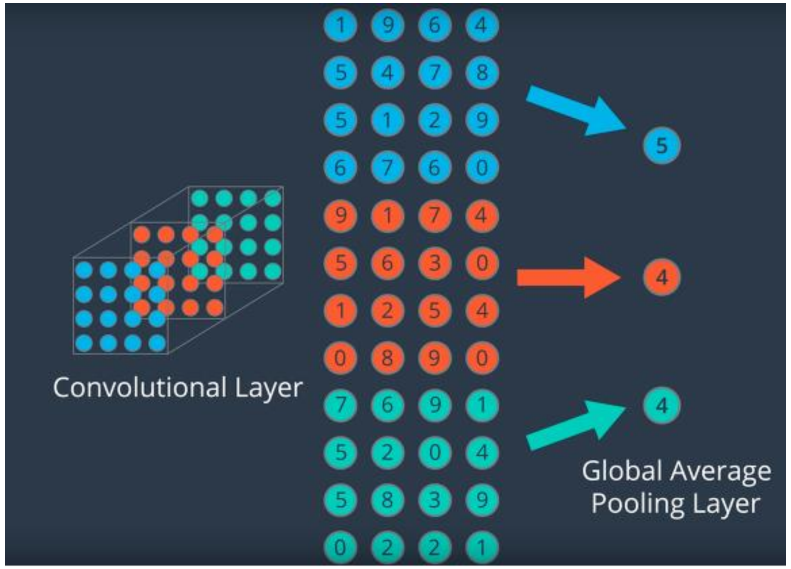
Model

```
# Main parameters
BATCH_SIZE = 8
STEPS_PER_EPOCH = len(train_labels)*0.8 / BATCH_SIZE
VALIDATION_STEPS = len(train_labels)*0.2 / BATCH_SIZE
EPOCHS = 20
TARGET_SIZE = 512
```

```
def create_model():
    conv_base = EfficientNetB0(include_top = False, weights = None,
                               input_shape = (TARGET_SIZE, TARGET_SIZE, 3))
    model = conv_base.output
    model = layers.GlobalAveragePooling2D()(model)
    model = layers.Dense(5, activation = "softmax")(model)
    model = models.Model(conv_base.input, model)

    model.compile(optimizer = Adam(lr = 0.001),
                  loss = "sparse_categorical_crossentropy",
                  metrics = ["acc"])
    return model
```

Global average pooling layer



- The global average pooling layer does not specify window size or stride.
- This pooling layer reduces the dimensionality of the CNN in a more abrupt way.
- Extract the average of the node values on each feature map.

Model

- Model summary

```
Model: "functional_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 512, 512, 3) 0
__________________________________________________________________________________________________
rescaling (Rescaling)           (None, 512, 512, 3)  0           input_1[0][0]
__________________________________________________________________________________________________
normalization (Normalization)   (None, 512, 512, 3)  7           rescaling[0][0]
__________________________________________________________________________________________________
stem_conv_pad (ZeroPadding2D)   (None, 513, 513, 3)  0           normalization[0][0]
__________________________________________________________________________________________________
stem_conv (Conv2D)              (None, 256, 256, 32) 864         stem_conv_pad[0][0]
__________________________________________________________________________________________________
stem_bn (BatchNormalization)    (None, 256, 256, 32) 128         stem_conv[0][0]
__________________________________________________________________________________________________
stem_activation (Activation)    (None, 256, 256, 32) 0           stem_bn[0][0]
__________________________________________________________________________________________________
block1a_dwconv (DepthwiseConv2D (None, 256, 256, 32) 288         stem_activation[0][0]
__________________________________________________________________________________________________
block1a_bn (BatchNormalization) (None, 256, 256, 32) 128         block1a_dwconv[0][0]
__________________________________________________________________________________________________
```

. . .

```
__________________________________________________________________________________________________
block7a_se_reduce (Conv2D)      (None, 1, 1, 48)     55344       block7a_se_reshape[0][0]
__________________________________________________________________________________________________
block7a_se_expand (Conv2D)      (None, 1, 1, 1152)   56448       block7a_se_reduce[0][0]
__________________________________________________________________________________________________
block7a_se_excite (Multiply)    (None, 16, 16, 1152) 0           block7a_activation[0][0]
                                                                 block7a_se_expand[0][0]
__________________________________________________________________________________________________
block7a_project_conv (Conv2D)   (None, 16, 16, 320)  368640      block7a_se_excite[0][0]
__________________________________________________________________________________________________
block7a_project_bn (BatchNormal (None, 16, 16, 320)  1280        block7a_project_conv[0][0]
__________________________________________________________________________________________________
top_conv (Conv2D)               (None, 16, 16, 1280) 409600      block7a_project_bn[0][0]
__________________________________________________________________________________________________
top_bn (BatchNormalization)     (None, 16, 16, 1280) 5120        top_conv[0][0]
__________________________________________________________________________________________________
top_activation (Activation)     (None, 16, 16, 1280) 0           top_bn[0][0]
__________________________________________________________________________________________________
global_average_pooling2d (Globa (None, 1280)         0           top_activation[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 5)            6405        global_average_pooling2d[0][0]
==================================================================================================
Total params: 4,055,976
Trainable params: 4,013,953
Non-trainable params: 42,023
```

- EfficientNetB0 CNN model consists of a total of 239 layers

- Model training

```
Epoch 1/20
2140/2139 [==============================] - ETA: 0s - loss: 0.6477 - acc: 0.7721
Epoch 00001: val_loss improved from inf to 0.65709, saving model to ./EffNetB0_512_8_best_weights.h5
2140/2139 [==============================] - 1730s 808ms/step - loss: 0.6477 - acc: 0.7721 - val_loss: 0.6571 - val_acc: 0.7871
Epoch 2/20
2140/2139 [==============================] - ETA: 0s - loss: 0.5273 - acc: 0.8182
Epoch 00002: val_loss improved from 0.65709 to 0.49969, saving model to ./EffNetB0_512_8_best_weights.h5
2140/2139 [==============================] - 1666s 778ms/step - loss: 0.5273 - acc: 0.8182 - val_loss: 0.4997 - val_acc: 0.8364
Epoch 3/20
2140/2139 [==============================] - ETA: 0s - loss: 0.4861 - acc: 0.8317
Epoch 00003: val_loss improved from 0.49969 to 0.49330, saving model to ./EffNetB0_512_8_best_weights.h5
2140/2139 [==============================] - 1647s 769ms/step - loss: 0.4861 - acc: 0.8317 - val_loss: 0.4933 - val_acc: 0.8387
Epoch 4/20
2140/2139 [==============================] - ETA: 0s - loss: 0.4604 - acc: 0.8403
Epoch 00004: val_loss improved from 0.49330 to 0.46575, saving model to ./EffNetB0_512_8_best_weights.h5
2140/2139 [==============================] - 1690s 790ms/step - loss: 0.4604 - acc: 0.8403 - val_loss: 0.4658 - val_acc: 0.8444
Epoch 5/20
2140/2139 [==============================] - ETA: 0s - loss: 0.4442 - acc: 0.8465
Epoch 00005: val_loss did not improve from 0.46575
```

. . .

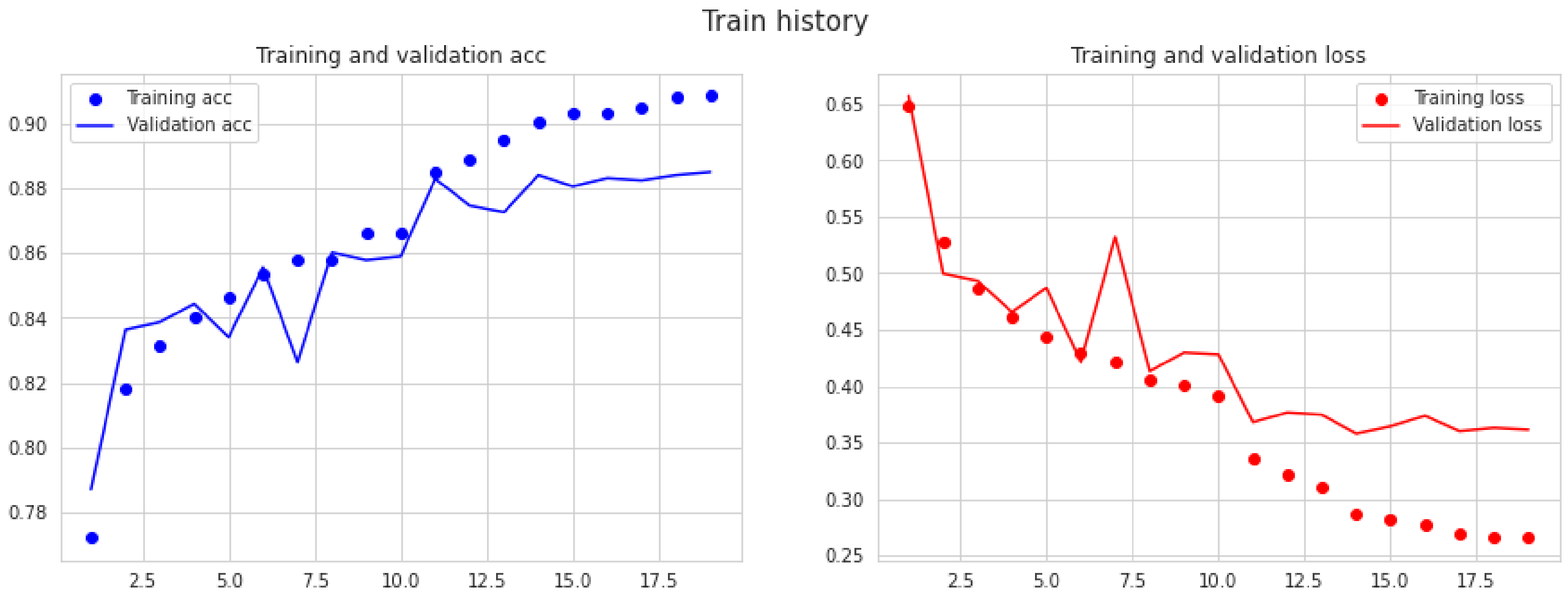
```
Epoch 16/20
2140/2139 [==============================] - ETA: 0s - loss: 0.2781 - acc: 0.9031
Epoch 00016: val_loss did not improve from 0.35809

Epoch 00016: ReduceLROnPlateau reducing learning rate to 2.700000040931627e-05.
2140/2139 [==============================] - 1593s 744ms/step - loss: 0.2781 - acc: 0.9031 - val_loss: 0.3738 - val_acc: 0.8832
Epoch 17/20
2140/2139 [==============================] - ETA: 0s - loss: 0.2697 - acc: 0.9051
Epoch 00017: val_loss did not improve from 0.35809
2140/2139 [==============================] - 1576s 736ms/step - loss: 0.2697 - acc: 0.9051 - val_loss: 0.3603 - val_acc: 0.8824
Epoch 18/20
2140/2139 [==============================] - ETA: 0s - loss: 0.2658 - acc: 0.9082
Epoch 00018: val_loss did not improve from 0.35809

Epoch 00018: ReduceLROnPlateau reducing learning rate to 8.100000013655517e-06.
2140/2139 [==============================] - 1581s 739ms/step - loss: 0.2658 - acc: 0.9082 - val_loss: 0.3632 - val_acc: 0.8841
Epoch 19/20
2140/2139 [==============================] - ETA: 0s - loss: 0.2655 - acc: 0.9085
Epoch 00019: val_loss did not improve from 0.35809
Restoring model weights from the end of the best epoch.
2140/2139 [==============================] - 1609s 752ms/step - loss: 0.2655 - acc: 0.9085 - val_loss: 0.3616 - val_acc: 0.8850
Epoch 00019: early stopping
```

- If the model performance did not improve 5 times in a row, training was stopped
- As a result of training, an accuracy of 90% was obtained.

Result



- It can be seen that the performance did not improve after 15 epochs by expressing the results of the previous training as a graph.

Conclusion

- As a result of training using the EfficientNetB0 model, I got good performance with fewer parameters than other CNN models.

- Ideas for future improvements:
  - try various image sizes
  - try various batch sizes
  - experiments with learning rate
  - experiments with data augmentation
  - other

Reference

Kaggle

- https://www.kaggle.com/c/cassava-leaf-disease-classification
- https://www.kaggle.com/maksymshkliarevskyi/cassava-leaf-disease-best-keras-cnn/notebook
- https://www.kaggle.com/vkehfdl1/for-korean-cassava

EfficientNet Reference

- https://lynnshin.tistory.com/13
- https://lynnshin.tistory.com/53

Global average pooling layer

- https://kevinthegrey.tistory.com/142

EfficientNet thesis

- https://arxiv.org/pdf/1905.11946.pdf

ImageDataGenerator

- https://3months.tistory.com/199


Thank you