

농작물 질병 분류

버 이미지 학습을 통한 labeling





목차

1. 배경 소개

2. Data EDA

3. BaseModel 구현

4. mainModel 구현

5. 결론

1. 배경 소개



빅데이터, 인공지능을 사용한 스마트 농업 사업의 중요성이 대두되고 있다.

이러한 상황에 맞게, 경작하는 경작물의 상태를 확인하는데

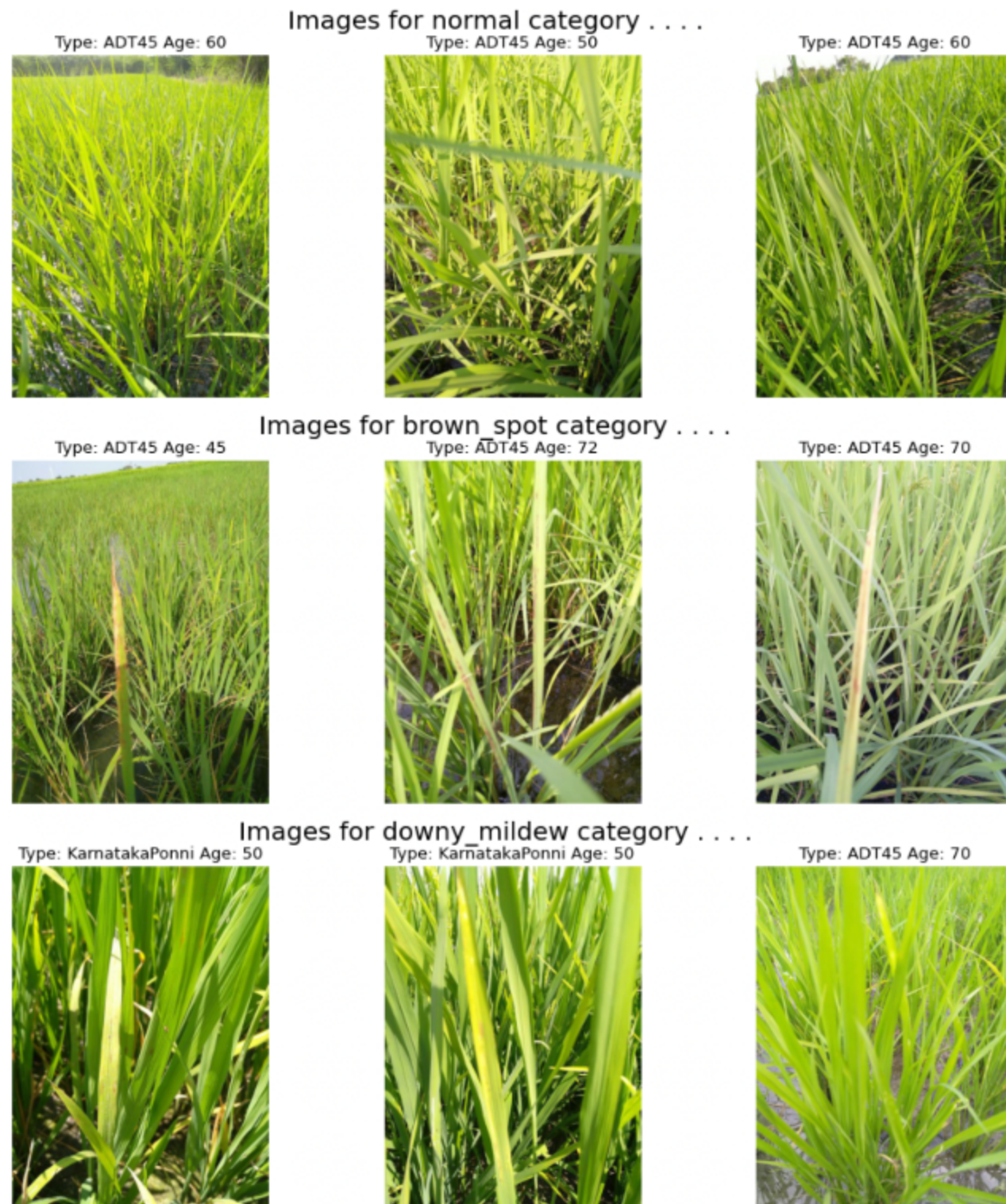
인적/물리적 자원이 소모되던 과거와 달리,

이미지 데이터를 사용한 딥러닝 분류 모델을 사용한

스마트 농업이 도입된다면, 인적/물리적 자원을 효율적으로 사용함에 따라

농작물의 질과 양의 향상이라는 가치 실현을 기대할 수 있다.

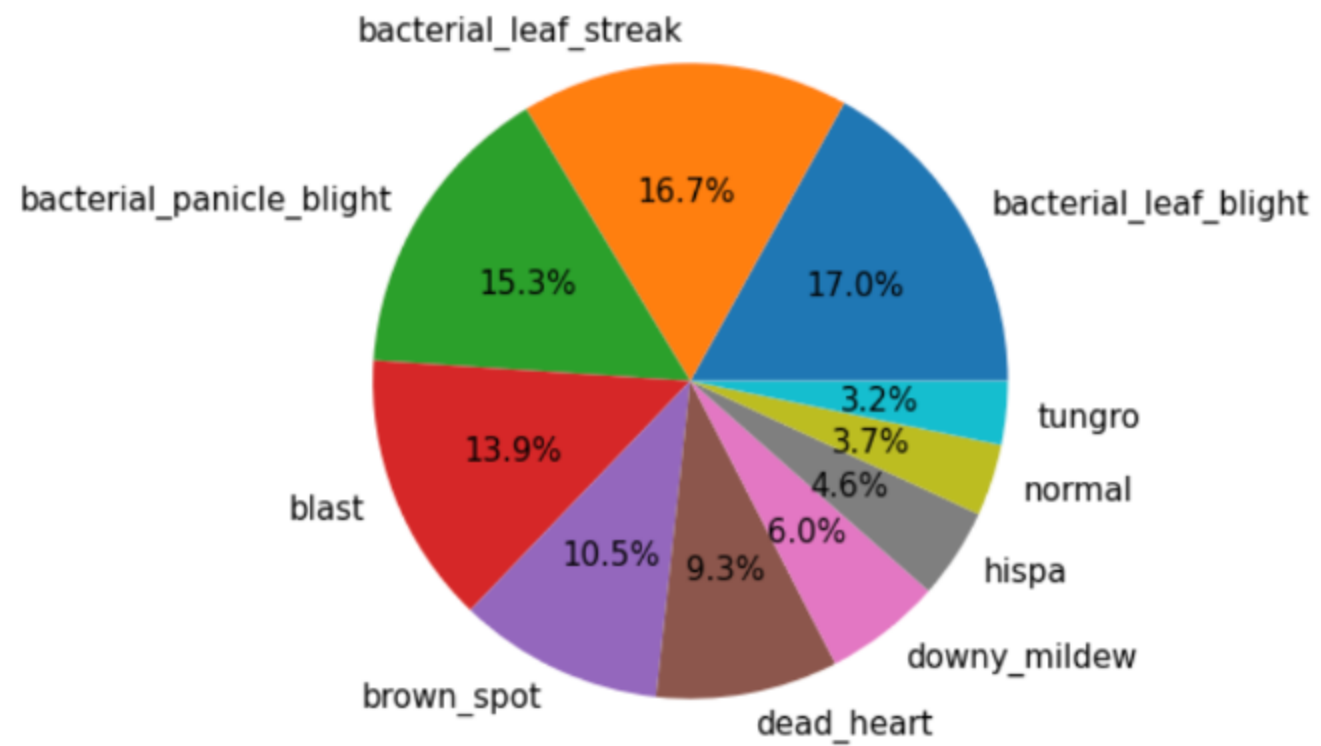
2. Data EDA



실제 labeling 되어 있는 이미지를 출력해보았다.

전문가가 아니면 구분하기 어려워 인적/물리적 자원의 소모가 예상된다.

2. Data EDA



실제 labeling 되어 있는 Data는 다음과 같은 비율로 구성되어 있다.

Model 학습시에 label 비율에 맞게 train/validation/test set으로 분류할 필요가 있다.

```
df['label'].value_counts()
normal      1764
blast       1738
hispa       1594
dead_heart  1442
tungro      1088
brown_spot   965
downy_mildew 620
bacterial_leaf_blight 479
bacterial_leaf_streak 380
bacterial_panicle_blight 337
Name: label, dtype: int64
```

3. BaseModel 구현

```
# 데이터 종류에 맞게 ImageDataGenerator 객체 생성
tr_gen = ImageDataGenerator(horizontal_flip=True, rescale=1/255., shear_range=0.2, zoom_range=0.2)
val_gen = ImageDataGenerator(rescale=1/255.)
test_gen = ImageDataGenerator(rescale=1/255.)

# 데이터 종류에 맞는 Pandas.DataFrame으로부터 Numpy Array Iterator 생성
tr_flow_gen = tr_gen.flow_from_dataframe(dataframe=train, x_col='path', y_col='label',
                                         target_size=(256, 256), class_mode='sparse',
                                         shuffle=True)

val_flow_gen = val_gen.flow_from_dataframe(dataframe=val, x_col='path', y_col='label',
                                           target_size=(256, 256), class_mode='sparse',
                                           shuffle=False)

test_flow_gen = test_gen.flow_from_dataframe(dataframe=test_df, x_col='path', y_col='label',
                                             target_size=(256, 256), class_mode='sparse',
                                             shuffle=False)

Found 7506 validated image filenames belonging to 10 classes.
Found 1877 validated image filenames belonging to 10 classes.
Found 1043 validated image filenames belonging to 10 classes.
```

1. 실제 labeling 되어 있는 Data는 다음과 같은 비율에 맞게 train/validation/test set으로 데이터를 나눈다.

3. BaseModel 구현

```
# baseline model 정의
def create_bs_model(verbose=False):
    bs_model = Sequential([
        Conv2D(64, (5, 5), padding='same', activation='relu', input_shape=(256, 256, 3)),
        MaxPooling2D(2, 2),
        Conv2D(128, (3, 3), padding='same', activation='relu'),
        Flatten(),
        Dense(256, activation='relu'),
        Dense(10, activation='softmax')
    ])

    if verbose:
        bs_model.summary()

    return bs_model
```

1. 실제 labeling 되어 있는 Data는 다음과 같은 비율에 맞게
train/validation/test set으로 데이터를 나눈다.



2. 3개의 은닉층을 가진 모델을 간단하게 구현하여 학습을 진행하였다.

```
Model: "sequential"
_____
Layer (type)                 Output Shape              Param #
=====
conv2d (Conv2D)              (None, 256, 256, 64)      4864
max_pooling2d (MaxPooling2D) (None, 128, 128, 64)      0
conv2d_1 (Conv2D)            (None, 128, 128, 128)     73856
flatten (Flatten)            (None, 2097152)           0
dense (Dense)                (None, 256)               536871168
dense_1 (Dense)              (None, 10)                2570
=====
Total params: 536,952,458
Trainable params: 536,952,458
Non-trainable params: 0
_____
```

3. BaseModel 구현

```
tqdm_callback = tfa.callbacks.TQDMProgressBar()

bs_model.fit(tr_flow_gen, epochs=1, validation_data=val_flow_gen, callbacks=[tqdm_callback])

Training: 100%  1/1 ETA: 00:00s, 3745.11s/epochs
Epoch 1/1
118/118  ETA: 00:00s - loss: 1.9498 - accuracy: 0.3119 - val_loss: 1.7800 - val_accuracy: 0.3863
118/118 [=====] - 3745s 32s/step - loss: 1.9498 - accuracy: 0.3119 - val_loss: 1.7800 - val_accuracy: 0.3863
<keras.callbacks.History at 0x7fc15e3463d0>

bs_model.evaluate(test_flow_gen)

17/17 [=====] - 459s 29s/step - loss: 1.7540 - accuracy: 0.3941
[1.7539544105529785, 0.3940556049346924]
```

데이터 학습에 많은 시간 소요 낮은 Accuracy

4. MainModel 구현(1) - VGG

```
# 사전학습 모델 사용

pretrained_model = VGG16(weights='imagenet', include_top=False)

Downloading data from https://storage.googleapis.com/tensorflow/keras-
58892288/58889256 [=====] - 0s 0us/step
58900480/58889256 [=====] - 0s 0us/step

# 사전 학습 모델 위에 완전 연결 신경망 추가
# VGG

model_vgg = Sequential()
model_vgg.add(pretrained_model)
model_vgg.add(GlobalAveragePooling2D())
model_vgg.add(Dense(128,activation='relu'))
model_vgg.add(Dense(10,activation='softmax'))

model_vgg.summary()

Model: "sequential_3"

```

| Layer (type) | Output Shape | Param # |
|--------------------------------------------------------|-------------------------|----------|
| vgg16 (Functional) | (None, None, None, 512) | 14714688 |
| global_average_pooling2d_1 (GlobalAveragePooling2D) | (None, 512) | 0 |
| dense_6 (Dense) | (None, 128) | 65664 |
| dense_7 (Dense) | (None, 10) | 1290 |

```

=====
Total params: 14,781,642
Trainable params: 14,781,642
Non-trainable params: 0
=====
```

VGG 모델로 사전학습 모델을 정의하고
간단한 완전신경망을 연결하여 학습을 진행하였다.

4. MainModel 구현(1) - VGG

```
Epoch 1/10
235/235 [=====] - 205s 869ms/step - loss: 2.1059 - accuracy: 0.2225 - val_loss: 1.9944 - val_accuracy: 0.2552
Epoch 2/10
235/235 [=====] - 204s 868ms/step - loss: 1.9312 - accuracy: 0.3002 - val_loss: 1.7249 - val_accuracy: 0.3980
Epoch 3/10
235/235 [=====] - 203s 864ms/step - loss: 1.6492 - accuracy: 0.4334 - val_loss: 1.5683 - val_accuracy: 0.4566
Epoch 4/10
235/235 [=====] - 204s 866ms/step - loss: 1.4996 - accuracy: 0.4840 - val_loss: 1.4372 - val_accuracy: 0.5147
Epoch 5/10
235/235 [=====] - 205s 870ms/step - loss: 1.4024 - accuracy: 0.5147 - val_loss: 1.3316 - val_accuracy: 0.5466
Epoch 6/10
235/235 [=====] - 205s 873ms/step - loss: 1.3459 - accuracy: 0.5334 - val_loss: 1.3234 - val_accuracy: 0.5445
Epoch 7/10
235/235 [=====] - 206s 874ms/step - loss: 1.2743 - accuracy: 0.5549 - val_loss: 1.2722 - val_accuracy: 0.5743
Epoch 8/10
235/235 [=====] - 208s 882ms/step - loss: 1.2788 - accuracy: 0.5596 - val_loss: 1.4669 - val_accuracy: 0.5535
Epoch 9/10
235/235 [=====] - 207s 879ms/step - loss: 1.2561 - accuracy: 0.5581 - val_loss: 1.2461 - val_accuracy: 0.5850
Epoch 10/10
235/235 [=====] - 204s 865ms/step - loss: 1.1854 - accuracy: 0.5889 - val_loss: 1.1485 - val_accuracy: 0.6201
<keras.callbacks.History at 0x7fbf47175f10>
```

BaseModel 에 비해 속도가 빨라져 epoch을 증가시켜 학습

Accuracy 또한 증가하였지만, 여전히 낮은 Accuracy

4. MainModel 구현(2) - mobileNetV2

```
AUTOTUNE = tf.data.experimental.AUTOTUNE
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)

def get_model(base, preprocessor, img_size):
    inputs = tf.keras.Input(shape=(img_size, img_size, 3))
    x = RandomFlip('horizontal')(inputs)
    x = preprocessor(x)
    x = base(x)

    x = tfl.Flatten()(x)
    x = tfl.Dense(1024, activation='relu')(x)
    x = tfl.BatchNormalization()(x)
    x = tfl.Dense(512, activation='relu')(x)
    x = tfl.BatchNormalization()(x)
    x = tfl.Dense(128, activation='relu')(x)
    x = tfl.Dropout(0.15)(x)
    x = tfl.BatchNormalization()(x)
    x = tfl.Dense(64, activation='relu')(x)
    x = tfl.Dropout(0.3)(x)
    x = tfl.BatchNormalization()(x)
    outputs = tfl.Dense(10, activation='softmax')(x)

    model = tf.keras.Model(inputs, outputs)

    return model

preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input

pre_trained_model = tf.keras.applications.MobileNetV2(input_shape=(img_size, img_size, 3),
                                                        include_top=False,
                                                        weights='imagenet')

pre_trained_model.trainable = False

WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 128,
```

| Model: "model_2" | | |
|-----------------------------|-----------------------|-----------|
| Layer (type) | Output Shape | Param # |
| ===== | | |
| input_7 (InputLayer) | [(None, 400, 400, 3)] | 0 |
| random_flip_2 (RandomFlip) | (None, 400, 400, 3) | 0 |
| tf.math.truediv_2 (TFOpLamb | (None, 400, 400, 3) | 0 |
| da) | | |
| tf.math.subtract_2 (TFOpLam | (None, 400, 400, 3) | 0 |
| bda) | | |
| mobilenetv2_1.00_224 (Funct | (None, 13, 13, 1280) | 2257984 |
| ional) | | |
| flatten_2 (Flatten) | (None, 216320) | 0 |
| dense_12 (Dense) | (None, 1024) | 221512704 |
| batch_normalization_8 (Batc | (None, 1024) | 4096 |
| hNormalization) | | |
| dense_13 (Dense) | (None, 512) | 524800 |
| batch_normalization_9 (Batc | (None, 512) | 2048 |
| hNormalization) | | |
| dense_14 (Dense) | (None, 128) | 65664 |
| dropout_4 (Dropout) | (None, 128) | 0 |
| batch_normalization_10 (Bat | (None, 128) | 512 |
| chNormalization) | | |
| dense_15 (Dense) | (None, 64) | 8256 |
| dropout_5 (Dropout) | (None, 64) | 0 |
| batch_normalization_11 (Bat | (None, 64) | 256 |
| chNormalization) | | |
| dense_16 (Dense) | (None, 10) | 650 |
| ===== | | |

다음과 같은 수정사항을 반영해 새롭게 MainModel을 정의하였다.

- 1. mobileNet-V2 를 사전학습 모델로 사용
- 2. 더 깊은 신경망을 구축
- 3. 기울기 소실을 방지하기 위해 batchNomalization 사용
- 4. 과적합 방지를 위해 Dropout 적용

4. MainModel 구현(2) - mobileNetV2

```
history = model_v2.fit(train_dataset,  
                        validation_data=val_dataset,  
                        epochs=5, verbose=1)
```

```
Epoch 1/5  
147/147 [=====] - 605s 4s/step - loss: 1.2682 - accuracy: 0.5921 - val_loss: 0.7675 - val_accuracy: 0.7620  
Epoch 2/5  
147/147 [=====] - 33s 220ms/step - loss: 0.5056 - accuracy: 0.8497 - val_loss: 0.3239 - val_accuracy: 0.9117  
Epoch 3/5  
147/147 [=====] - 33s 220ms/step - loss: 0.2697 - accuracy: 0.9268 - val_loss: 0.1782 - val_accuracy: 0.9530  
Epoch 4/5  
147/147 [=====] - 33s 220ms/step - loss: 0.1733 - accuracy: 0.9547 - val_loss: 0.2513 - val_accuracy: 0.9290  
Epoch 5/5  
147/147 [=====] - 34s 222ms/step - loss: 0.1186 - accuracy: 0.9695 - val_loss: 0.0855 - val_accuracy: 0.9760
```

데이터 학습 시간 절약
높은 Accuracy 도출

Result

딥러닝 모델을 구현할 때는

데이터의 특징을 확인하고,

학습률을 높이기 위해 데이터와 상황에 맞는

하이퍼파라미터 설정이 중요하다는 것을 알 수 있다.

```
history = model_v2.fit(train_data, validation_data=val_data, epochs=5)

Epoch 1/5
147/147 [=====]
Epoch 2/5
147/147 [=====]
Epoch 3/5
147/147 [=====]
Epoch 4/5
147/147 [=====]
Epoch 5/5
147/147 [=====]
```

```
175 - val_accuracy: 0.7620
3239 - val_accuracy: 0.9117
1782 - val_accuracy: 0.9530
2513 - val_accuracy: 0.9290
0855 - val_accuracy: 0.9760
```


감사합니다

