[Project] 교통 표지판 이미지 분류

프로젝트 목표

- 교통 표지판 이미지 데이터를 분석하고 딥러딩 모델을 통하여 표지판 종류를 예측하는 분류 모델 수행
- 대량의 이미지 데이터를 전 처리하는 과정과 이에 따른 CNN 모델의 성능 변화를 학습

프로젝트 목차

1. **데이터 분석:** 이미지 데이터를 이루고 있는 요소에 대해서 Dataframe를 사용하여 분석 및 확인
   1-1. 이미지 데이터 정보 파악하기 - Meta
   1-2. 이미지 데이터 정보 파악하기 - Train
   1-3. 이미지 데이터 정보 파악하기 - Test

1. **데이터 전 처리:** 이미지 데이터를 읽어오고 딥러닝 모델의 입력으로 전 처리
   2-1. 이미지 데이터 읽기
   2-2. label 데이터 읽기
   2-3. 데이터 분리하기

1. **딥러닝 모델:** CNN 모델을 구현하고 학습, 평가 및 예측을 수행
   3-1. CNN 모델 설정
   3-2. 학습 수행
   3-3. 모델 성능 평가 및 예측

데이터 출처

- https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign

---

프로젝트 개요

차량 운전을 하면서 도로 교통 표지판을 보고 규칙을 지키는 것은 운전자의 및 교통 안전을 위해서 중요한 일입니다. 만일 사람이 아닌 기계가 이를 수행해야 한다면, 어떻게 표지판을 구분할 수 있을까요? 이러한 물음은 자율 주행차 기술이 발전하면서 중요한 이슈가 되었고, 딥러닝 기술 바탕의 분류 모델이 상당한 수준의 정확도를 보이며 적용되고 있습니다.

이번 프로젝트에서는 교통 표지판 분류의 첫 번째 스텝으로 간단하게 교통 표지판 이미지가 입력 되었을 때 이것이 43 종의 표지판 중 어떤 것인가를 분류하는 딥러닝 모델을 구현합니다. 이를 통하여 교통 표지판 이미지 데이터들의 특징과 CNN 모델을 통하여 분류를 수행하는 것을 학습할 수 있습니다.

첫 번째 스텝을 확장하여, 추 후에는 교통 표지판만 있는 이미지를 입력으로 하지 않고 도로 이미지에서 교통 표지판을 분리하여 분류하는 Object detection 모델을 사용할 것이며, 최종적으로는 센서로 입력받는 비디오 이미지에서의 교통 표지판을 분리하고 분류하는 모델을 수행할 수 있습니다.

기본 CNN 분류 모델

Object detection 분류 모델

- Object etection 이미지 참조 논문: V.N. Sichkar, S.A. Kolyubin, "Real time detection and classification of traffic signs based on YOLO version 3 algorithm", Scientific and Technical Journal of Information Technologies Mechanics and Optics 20(3):418-424

In [5]:
```
import os
import pathlib
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_a
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropou
from tensorflow.keras.models import Sequential


%matplotlib inline
```

1. 데이터 분석

1-1. 이미지 데이터 정보 파악하기 - Meta

이미지 데이터를 읽어오기 위해서 `./data` 에 어떠한 파일들이 존재 하는지 확인해 봅시다.

In [6]:
```
file_list = os.listdir('./data')
file_list
```

Out[6]:
```
['Meta', 'Meta.csv', 'Test', 'Test.csv', 'Train', 'Train.csv']
```

3개의 폴더와 3개의 csv 파일이 있습니다. 일반적으로 이미지 데이터의 csv 파일이 제공되는 경우에는, 해당 이미지의 디렉토리 정보가 저장되어 있습니다.

그렇기에 먼저,csv 파일을 dataframe으로 읽어 보겠습니다.

In [7]:
```
import pandas as pd

df_Meta = pd.read_csv('./data/Meta.csv')
df_Meta
```

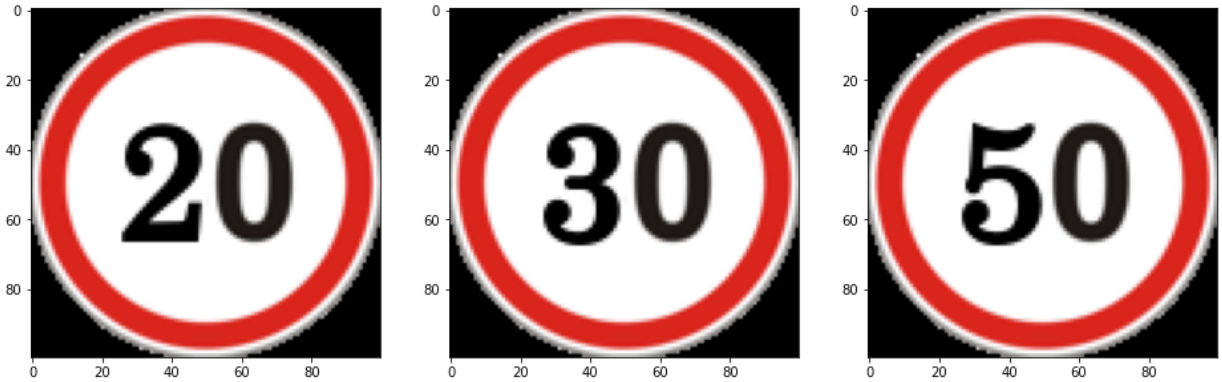
Out[7]:

	Path	ClassId	ShapeId	ColorId	SignId
0	Meta/0.png	0	1	0	3.29

	Path	ClassId	ShapeId	ColorId	SignId
1	Meta/1.png	1	1	0	3.29
2	Meta/2.png	2	1	0	3.29

`Meta.csv` 는 Meta 폴더 내의 이미지에 대한 정보를 담고 있습니다. 위정보를 바탕으로 이미지를 출력해봅시다.

In [8]:
```
Meta_images = []
Meta_labels = []

plt.figure(figsize=(16,16))
for i in range(len(df_Meta)):
    img = load_img('./data/'+df_Meta['Path'][i])
    plt.subplot(1, 3, i+1)
    plt.imshow(img)
    Meta_images.append(img)
    Meta_labels.append(df_Meta['ClassId'][i])
```



1-2. 이미지 데이터 정보 파악하기 - Train

이번엔 `Train.csv` 를 확인해 봅시다.

In [9]:
```
df_Train = pd.read_csv('./data/Train.csv')
df_Train
```

Out[9]:

	Width	Height	Roi.X1	Roi.Y1	Roi.X2	Roi.Y2	ClassId	Path
0	29	30	5	6	24	25	0	Train/0/00000_00000_00000.png
1	30	30	5	5	25	25	0	Train/0/00000_00000_00001.png
2	30	30	5	5	25	25	0	Train/0/00000_00000_00002.png
3	31	31	5	5	26	26	0	Train/0/00000_00000_00003.png
4	30	32	5	6	25	26	0	Train/0/00000_00000_00004.png
...	...	...	...	...	...	...	...	...
2665	50	51	6	6	45	46	2	Train/2/00002_00040_00025.png
2666	52	52	6	6	47	47	2	Train/2/00002_00040_00026.png
2667	55	55	6	6	50	50	2	Train/2/00002_00040_00027.png
2668	59	58	6	5	53	52	2	Train/2/00002_00040_00028.png
2669	62	63	6	6	57	58	2	Train/2/00002_00040_00029.png

2670 rows × 8 columns

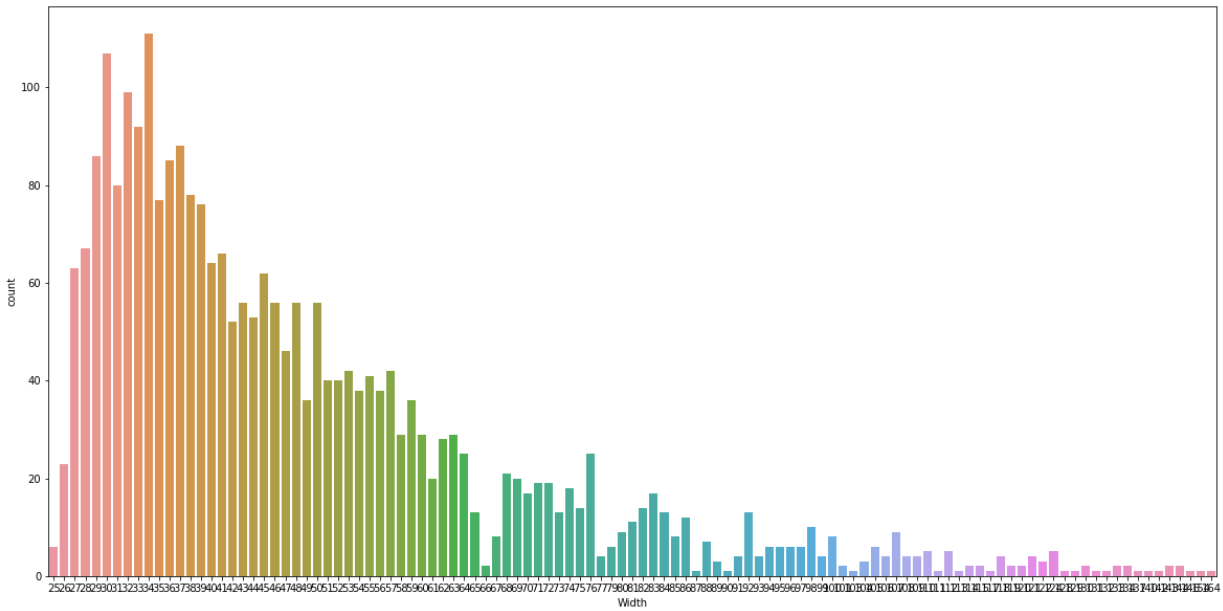
2670개의 학습용 이미지 데이터에 대한 정보가 저장되어 있음을 알 수 있습니다.

이러한 이미지 정보 중에 먼저 `Width` 와 `Height` 에 대해서 알아봅시다.

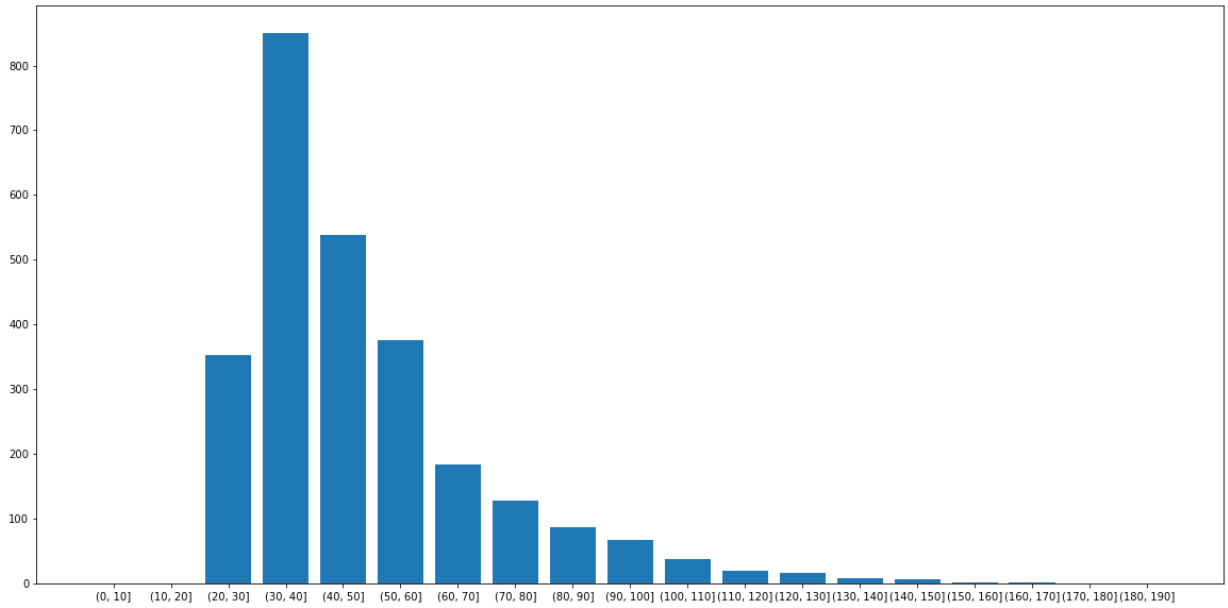
`Width` 와 `Height` 정보는 이미지의 폭과 높이에 대한 정보로 간단히 샘플만 봐도 다양한 크기를 갖는 것을 알 수 있습니다.

이미지 크기가 모두 다르다면 이미지마다 서로 다른 feature의 개수가 있는 것이기에 이를 통일해주는 전 처리가 필요합니다.

In [10]:
```
import seaborn as sns

plt.figure(figsize=(20,10))
ax = sns.countplot(x="Width", data=df_Train)
```



In [11]:
```
df_cutWidth = pd.cut(df_Train['Width'], np.arange(0,200,10)).value_counts(sor

fig, ax = plt.subplots(figsize=(20,10))
ax.bar(range(len(df_cutWidth)),df_cutWidth.values)
ax.set_xticks(range(len(df_cutWidth)))
ax.set_xticklabels(df_cutWidth.index)
fig.show()
```



위 분포를 보게 되면 30~35의 폭 또는 높이를 갖는 이미지가 제일 많음을 확인했습니다.

이미지 크기를 통일하는 데 있어서 너무 작은 이미지는 큰 이미지의 정보 손실을 발생하며, 너무 큰 이미지는 작은 이미지의 정보 부족한 정보량을 부각할 것입니다.

따라서 적절한 이미지 크기를 잡는 것은 하나의 파라미터 조정이 되며, 이번 프로젝트에서는 이미지 분포 기반으로 대다수를 차지하는 크기인 33x33 크기로 통일하겠습니다.

In [12]:
```
image_height = 33
image_width = 33
image_channel = 3 # 컬러 이미지이기에 3채널
```

이번엔 Roi 데이터에 대해서 살펴봅시다. 이미지 데이터에서 Roi는 Region of interest의 약자로 지금 데이터에서는 표지판이 있는 부분을 의미합니다.

`Train.csv` , `Test.csv` 파일에 있는 Roi 데이터는 아래 실행된 이미지에서의 좌측 상단 좌표와 우측 하단 좌표를 의미합니다.

In [13]:
```
from PIL import Image
from PIL import ImageDraw

img_sample = Image.open('./data/'+df_Train['Path'][0])

draw = ImageDraw.Draw(img_sample)
draw.rectangle([df_Train['Roi.X1'][0], df_Train['Roi.Y1'][0], df_Train['Roi.X
img_sample_resized = img_sample.resize((300,300))
img_sample_resized
```

Out[13]:

Roi 데이터를 사용하면 보다 명확하게 표지판 부분만을 crop 할 수 있으며, 이러한 데이터 전 처리를 통하여 분류의 성능을 높일 수 있습니다.

In [14]:
```
img_sample_crop = img_sample.crop((df_Train['Roi.X1'][0], df_Train['Roi.Y1'][

# Shows the image in image viewer
img_sample_crop_resized = img_sample_crop.resize((300,300))
img_sample_crop_resized
```

Out[14]:

1-3. 이미지 데이터 정보 파악하기 - Test

이번엔 `Test.csv` 를 살펴보겠습니다.

In [15]:
```
df_Test = pd.read_csv('./data/Test.csv')
df_Test
```

Out[15]:

	Width	Height	Roi.X1	Roi.Y1	Roi.X2	Roi.Y2	ClassId	Path
0	42	45	5	5	36	40	1	Test/00001.png
1	59	65	5	6	54	60	1	Test/00024.png

	Width	Height	Roi.X1	Roi.Y1	Roi.X2	Roi.Y2	ClassId	Path
2	58	59	6	6	53	54	2	Test/00034.png
3	37	37	5	6	31	32	2	Test/00067.png
4	45	51	6	6	40	46	1	Test/00076.png
...	...	...	...	...	...	...	...	...
695	34	33	5	5	28	28	1	Test/05537.png
696	58	56	6	6	52	50	1	Test/05539.png
697	37	38	5	5	32	33	1	Test/05551.png
698	28	30	5	6	23	25	2	Test/05579.png
699	48	47	5	5	43	42	1	Test/05580.png

700 rows × 8 columns

Train.csv와 같은 형태로 구성되어 있는 것을 알 수 있습니다.

2. 데이터 전 처리

2-1. 이미지 데이터 읽기

통일된 이미지 크기를 바탕으로 이미지를 읽어 보겠습니다.

먼저 학습용 이미지를 불러와 `Train_images` 에 array 형태로 저장합니다.

In [16]:
```
image_height = 33
image_width = 33
image_channel = 3

Train_images = []
Train_labels = []

for i in tqdm(range(len(df_Train))):
    img = load_img('./data/'+df_Train['Path'][i], target_size = (image_height
    img = img_to_array(img)
    Train_images.append(img)
```

```
100%|■■■■■■■■■■| 2670/2670 [01:30<00:00, 29.48it/s]
```

같은 방식으로 평가용 이미지를 불러와 `Test_images` 에 array 형태로 저장합니다.

In [17]:
```
Test_images = []
Test_labels = []

for i in tqdm(range(len(df_Test))):
    img = load_img('./data/'+df_Test['Path'][i], target_size = (image_height,
    img = img_to_array(img)
    Test_images.append(img)
```

```
100%|■■■■■■■■■■| 700/700 [00:22<00:00, 31.39it/s]
```

2-2. label 데이터 읽기

학습용, 평가용 데이터에 대한 label은 csv 파일에 `ClassId` 열로 저장되어 있기에 이를 불러와 array로

저장합니다.

In [18]:
```
Train_labels = df_Train['ClassId'].values
Train_labels
```

Out[18]:
```
array([0, 0, 0, ..., 2, 2, 2])
```

In [19]:
```
Test_labels = df_Test['ClassId'].values
Test_labels
```

Out[19]:
```
array([1, 1, 2, 2, 1, 1, 2, 1, 1, 1, 1, 2, 1, 2, 2, 1, 2, 1, 2, 1, 2, 2,
       1, 0, 0, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 2, 2, 2, 2, 1, 1, 2, 1, 0,
       2, 2, 2, 1, 2, 1, 2, 1, 2, 2, 1, 2, 1, 1, 1, 1, 1, 1, 2, 0, 1, 2,
       2, 2, 1, 1, 1, 2, 2, 1, 2, 2, 0, 1, 2, 2, 2, 1, 2, 1, 1, 2, 2, 1,
       0, 2, 2, 2, 2, 1, 1, 2, 0, 0, 1, 1, 2, 2, 1, 1, 2, 1, 1, 2, 1, 1,
       1, 1, 1, 2, 1, 1, 2, 1, 2, 0, 1, 1, 2, 1, 0, 1, 2, 1, 1, 2, 2, 2,
       2, 2, 2, 2, 1, 2, 1, 1, 1, 2, 1, 2, 2, 1, 2, 1, 2, 2, 2, 0, 2, 1,
       2, 1, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 1, 2, 2, 2,
       2, 2, 1, 1, 0, 2, 1, 2, 1, 2, 2, 1, 2, 1, 2, 2, 1, 2, 2, 2, 1, 2,
       1, 1, 1, 1, 2, 1, 2, 2, 2, 1, 1, 2, 2, 0, 1, 2, 1, 1, 1, 2, 2, 2,
       1, 1, 2, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 1, 0, 2, 1, 1, 2, 2, 1, 2,
       1, 1, 2, 2, 2, 2, 1, 2, 1, 0, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1,
       2, 2, 2, 1, 1, 2, 2, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 2, 2, 1, 2, 1,
       2, 1, 2, 1, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 1, 1, 1, 2,
       2, 2, 1, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 2, 2,
       2, 1, 2, 1, 1, 0, 2, 2, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 2, 2,
       2, 1, 1, 1, 0, 1, 2, 2, 2, 1, 2, 1, 2, 1, 0, 2, 1, 1, 1, 2, 1, 1,
       2, 2, 2, 2, 1, 2, 2, 1, 2, 1, 1, 1, 1, 2, 2, 2, 1, 1, 2, 1, 1, 1,
       2, 1, 2, 2, 1, 1, 1, 2, 2, 2, 1, 0, 2, 1, 2, 1, 1, 2, 1, 1, 1, 1,
       2, 0, 2, 1, 2, 2, 1, 1, 1, 2, 0, 2, 2, 1, 2, 1, 2, 1, 1, 1, 0, 1,
       1, 2, 2, 2, 2, 2, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 1, 1, 1,
       1, 2, 1, 2, 0, 2, 1, 2, 2, 2, 2, 2, 1, 1, 1, 2, 0, 0, 1, 1, 2, 1,
       1, 1, 1, 2, 2, 1, 1, 0, 1, 1, 1, 1, 1, 2, 0, 2, 1, 1, 1, 1, 2, 1,
       0, 2, 2, 1, 2, 2, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 0, 1, 2,
       2, 2, 1, 1, 1, 2, 2, 1, 1, 2, 2, 1, 0, 0, 1, 1, 2, 1, 1, 2, 1, 2,
       1, 1, 2, 2, 1, 1, 1, 0, 2, 2, 2, 2, 2, 1, 2, 1, 2, 2, 1, 0, 2, 2,
       0, 1, 2, 1, 1, 2, 2, 2, 2, 1, 0, 2, 2, 1, 2, 1, 1, 0, 2, 1, 2, 2,
       1, 1, 1, 1, 2, 2, 1, 2, 0, 1, 1, 2, 1, 1, 2, 0, 1, 2, 2, 2, 1, 1,
       2, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, 1, 2, 2,
       1, 2, 2, 1, 2, 1, 2, 2, 1, 2, 1, 1, 1, 1, 2, 2, 2, 1, 1, 2, 1, 2,
       2, 1, 1, 2, 1, 2, 1, 2, 1, 1, 2, 1, 1, 1, 2, 2, 1, 2, 2, 1, 1, 1,
       1, 1, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1, 2, 1])
```

2-3. 데이터 분리하기

딥러닝 학습 시, 과적합을 막기 위하여 validation 데이터를 학습용 데이터에서 분리합니다.

모든 데이터는 numpy array로 저장합니다.

In [20]:
```
x_train, x_val, y_train, y_val = train_test_split(np.array(Train_images), np.
```

평가용 데이터도 적용합니다.

In [21]:
```
x_test = np.array(Test_images)
y_test = np.array(Test_labels)
```

3. 딥러닝 모델

3-1. CNN 모델 설정

CNN을 사용하여 간단하게 모델을 구현해 보겠습니다. filters, kernel 등의 사이즈는 하이퍼 파리미터로 자신만의 모델로 튜닝이 가능합니다.

In [22]:
```
model = Sequential([
    Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=(ima
    MaxPool2D(pool_size=(2, 2)),
    Dropout(rate=0.25),

    Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
    MaxPool2D(pool_size=(2, 2)),
    Dropout(rate=0.25),

    Flatten(),
    Dense(512, activation='relu'),
    Dropout(rate=0.25),
    Dense(3, activation='softmax')
])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 31, 31, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 2304)              0
_________________________________________________________________
dense (Dense)                (None, 512)               1180160
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 1539
=================================================================
Total params: 1,201,091
Trainable params: 1,201,091
Non-trainable params: 0
_________________________________________________________________
```

3-2. 학습 수행

3개의 class를 갖는 데이터이기에 loss 함수로 `sparse_categorical_crossentropy` 을 설정하고 optimizer로는 adam을 사용하여 학습하여 보겠습니다.

In [23]:
```
model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
```

In [24]:
```
EPOCHS = 30
```

```
# EPOCHS에 따른 성능을 보기 위하여 history 사용
history = model.fit(x_train,
                    y_train,
                    validation_data = (x_val, y_val), # validation 데이터 사용
                    epochs=EPOCHS,
                   )
```

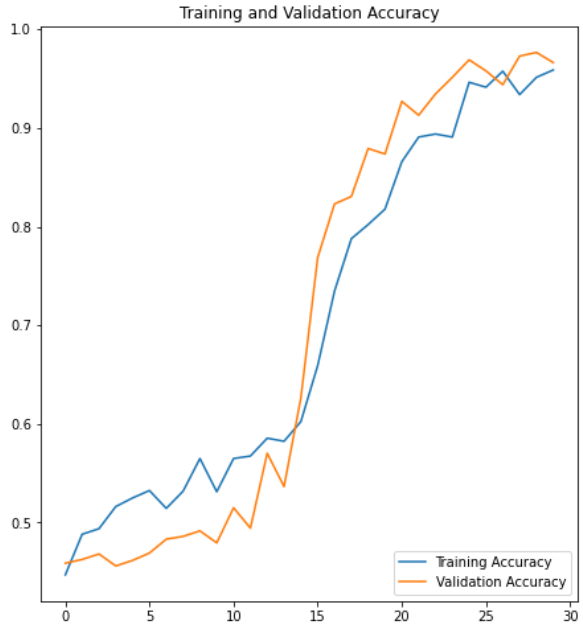
```
Epoch 1/30
51/51 [==============================] - 4s 68ms/step - loss: 52.3156 - accura
cy: 0.4518 - val_loss: 1.0509 - val_accuracy: 0.4588
Epoch 2/30
51/51 [==============================] - 3s 60ms/step - loss: 1.0035 - accurac
y: 0.4887 - val_loss: 0.9820 - val_accuracy: 0.4625
Epoch 3/30
51/51 [==============================] - 3s 58ms/step - loss: 0.9390 - accurac
y: 0.4951 - val_loss: 0.9459 - val_accuracy: 0.4682
Epoch 4/30
51/51 [==============================] - 3s 58ms/step - loss: 0.9401 - accurac
y: 0.5059 - val_loss: 0.9320 - val_accuracy: 0.4560
Epoch 5/30
51/51 [==============================] - 3s 59ms/step - loss: 0.8885 - accurac
y: 0.5337 - val_loss: 0.9167 - val_accuracy: 0.4616
Epoch 6/30
51/51 [==============================] - 3s 58ms/step - loss: 0.8878 - accurac
y: 0.5315 - val_loss: 0.9090 - val_accuracy: 0.4691
Epoch 7/30
51/51 [==============================] - 3s 58ms/step - loss: 0.8964 - accurac
y: 0.5183 - val_loss: 0.9082 - val_accuracy: 0.4831
Epoch 8/30
51/51 [==============================] - 3s 58ms/step - loss: 0.8610 - accurac
y: 0.5445 - val_loss: 0.8892 - val_accuracy: 0.4860
Epoch 9/30
51/51 [==============================] - 3s 58ms/step - loss: 0.8740 - accurac
y: 0.5570 - val_loss: 0.8948 - val_accuracy: 0.4916
Epoch 10/30
51/51 [==============================] - 3s 58ms/step - loss: 0.8896 - accurac
y: 0.5327 - val_loss: 0.9020 - val_accuracy: 0.4794
Epoch 11/30
51/51 [==============================] - 3s 58ms/step - loss: 0.8297 - accurac
y: 0.5669 - val_loss: 0.8788 - val_accuracy: 0.5150
Epoch 12/30
51/51 [==============================] - 3s 58ms/step - loss: 0.8547 - accurac
y: 0.5813 - val_loss: 0.8921 - val_accuracy: 0.4944
Epoch 13/30
51/51 [==============================] - 3s 58ms/step - loss: 0.8379 - accurac
y: 0.5840 - val_loss: 0.8203 - val_accuracy: 0.5702
Epoch 14/30
51/51 [==============================] - 3s 58ms/step - loss: 0.8841 - accurac
y: 0.5850 - val_loss: 0.8389 - val_accuracy: 0.5365
Epoch 15/30
51/51 [==============================] - 3s 58ms/step - loss: 0.7725 - accurac
y: 0.6066 - val_loss: 0.7184 - val_accuracy: 0.6264
Epoch 16/30
51/51 [==============================] - 3s 59ms/step - loss: 0.7198 - accurac
y: 0.6442 - val_loss: 0.5247 - val_accuracy: 0.7687
Epoch 17/30
51/51 [==============================] - 3s 58ms/step - loss: 0.6695 - accurac
y: 0.7094 - val_loss: 0.4285 - val_accuracy: 0.8230
Epoch 18/30
51/51 [==============================] - 3s 58ms/step - loss: 0.4868 - accurac
y: 0.7875 - val_loss: 0.4006 - val_accuracy: 0.8305
Epoch 19/30
51/51 [==============================] - 3s 59ms/step - loss: 0.4745 - accurac
y: 0.7987 - val_loss: 0.3125 - val_accuracy: 0.8792
Epoch 20/30
51/51 [==============================] - 3s 58ms/step - loss: 0.4281 - accurac
y: 0.8406 - val_loss: 0.3289 - val_accuracy: 0.8736
Epoch 21/30
51/51 [==============================] - 3s 57ms/step - loss: 0.3839 - accurac
```

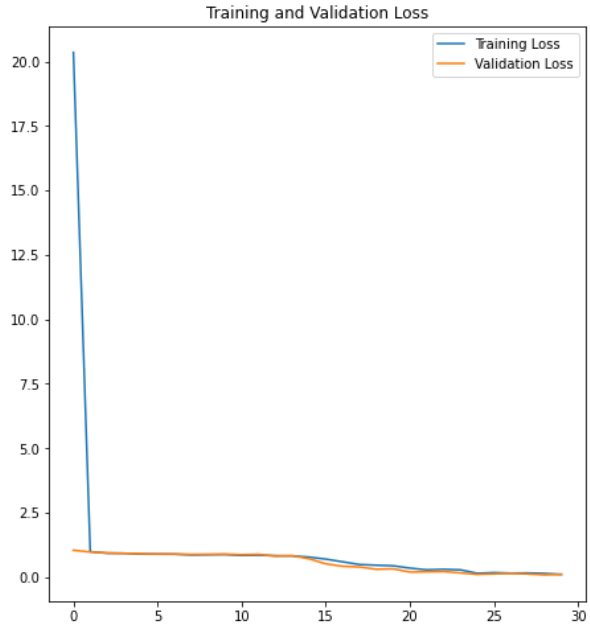
```
y: 0.8590 - val_loss: 0.2053 - val_accuracy: 0.9270
Epoch 22/30
51/51 [==============================] - 3s 58ms/step - loss: 0.2675 - accurac
y: 0.8946 - val_loss: 0.2152 - val_accuracy: 0.9129
Epoch 23/30
51/51 [==============================] - 3s 60ms/step - loss: 0.3193 - accurac
y: 0.8868 - val_loss: 0.2230 - val_accuracy: 0.9345
Epoch 24/30
51/51 [==============================] - 3s 58ms/step - loss: 0.2814 - accurac
y: 0.8974 - val_loss: 0.1725 - val_accuracy: 0.9513
Epoch 25/30
51/51 [==============================] - 3s 58ms/step - loss: 0.1582 - accurac
y: 0.9390 - val_loss: 0.1132 - val_accuracy: 0.9691
Epoch 26/30
51/51 [==============================] - 3s 58ms/step - loss: 0.1582 - accurac
y: 0.9534 - val_loss: 0.1345 - val_accuracy: 0.9579
Epoch 27/30
51/51 [==============================] - 3s 58ms/step - loss: 0.1463 - accurac
y: 0.9602 - val_loss: 0.1564 - val_accuracy: 0.9438
Epoch 28/30
51/51 [==============================] - 3s 58ms/step - loss: 0.1648 - accurac
y: 0.9346 - val_loss: 0.1286 - val_accuracy: 0.9728
Epoch 29/30
51/51 [==============================] - 3s 58ms/step - loss: 0.1389 - accurac
y: 0.9509 - val_loss: 0.0949 - val_accuracy: 0.9766
Epoch 30/30
51/51 [==============================] - 3s 58ms/step - loss: 0.1128 - accurac
y: 0.9584 - val_loss: 0.1145 - val_accuracy: 0.9663
```

학습을 수행하면서 Accuracy와 Loss의 변화를 그래프로 출력하면 다음과 같습니다.

In [25]:

```
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

loss=history.history['loss']
val_loss=history.history['val_loss']

epochs_range = range(EPOCHS)

plt.figure(figsize=(16, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, accuracy, label='Training Accuracy')
plt.plot(epochs_range, val_accuracy, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```





3-3. 모델 성능 평가 및 예측

학습이 잘 수행 되었으면 평가용 데이터에서 성능이 잘 나오는 확인합니다.

In [26]:
```
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=0)

print('test set accuracy: ', test_accuracy)
```

```
test set accuracy:  0.8928571343421936
```

이번엔 테스트 데이터를 입력하여 예측된 결과를 비교해 보겠습니다.

25개의 테스트 데이터를 불러와 실제 class와 예측 class를 출력하면 다음과 같습니다.

In [27]:
```
test_prediction = np.argmax(model.predict(x_test), axis=-1)
```

In [28]:
```
plt.figure(figsize = (13, 13))

start_index = 0
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    prediction = test_prediction[start_index + i]
    actual = y_test[start_index + i]
    col = 'g'
    if prediction != actual:
        col = 'r'
    plt.xlabel('Actual={} || Pred={}'.format(actual, prediction), color = col
    plt.imshow(array_to_img(x_test[start_index + i]))
plt.show()
```

마지막으로 confusion matrix를 시각화 하여 분류 학습 결과를 확인합니다.

In [29]:
```
import seaborn as sns
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, test_prediction)
plt.figure(figsize = (20, 20))
sns.heatmap(cm, annot = True)
```

Out[29]: <AxesSubplot:>