CIFAR-10 training

In Internship program

INDEX

- Introduction
- Dataset
- Architecture
- Experiment
- Conclusion

INTRODUCTION

What did I do a project?

›› Implementing Image Classification Model with CIFAR-10 datasets on CNN

Why did I do a project?

- To use what I have learned about image classification so far
- Previously, I have made simple models using small datasets

So this time, I practice using a large dataset to train the model

DATASET

CIFAR-10

›› Used CIFAR-10 dataset

- Image size

: **60,000 color images** with 32x32 pixels

- Class: **10 classes**

- Label

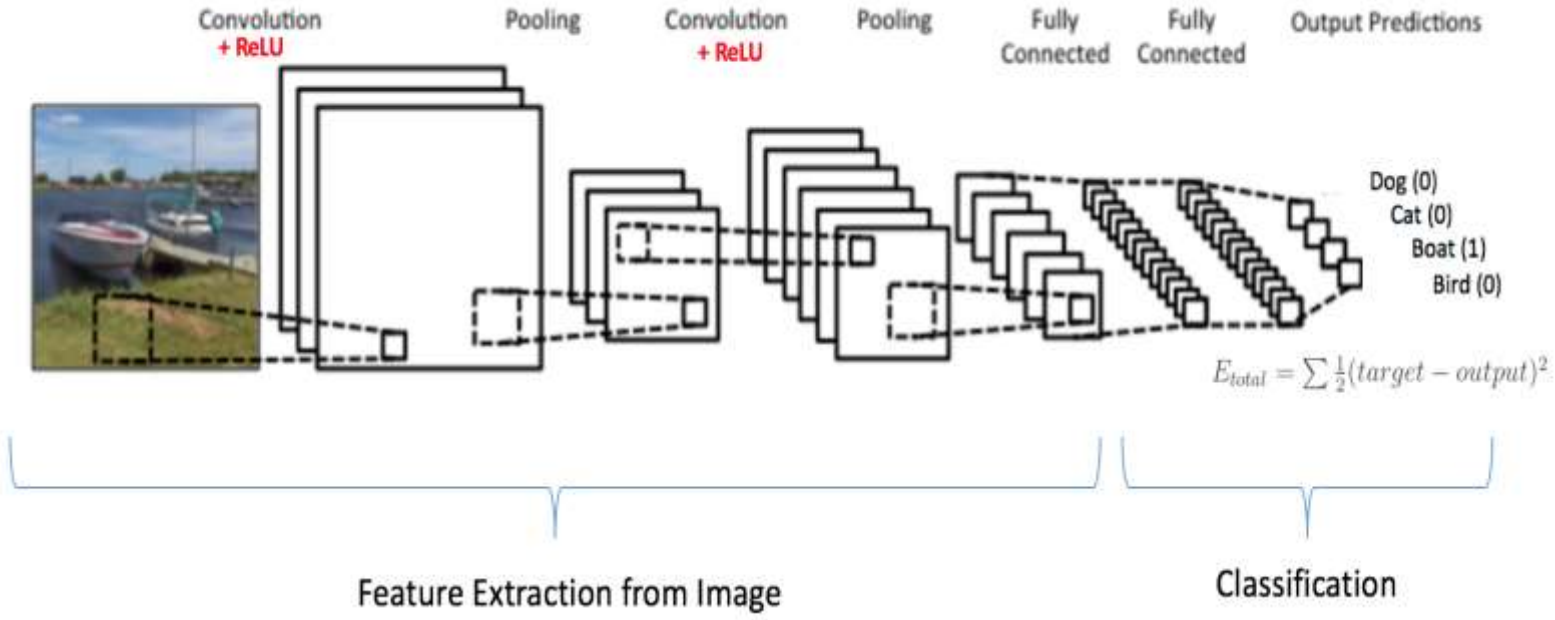
: [airplane, car, bird, cat, dear, dog, frog, horse, ship, truck]

ARCHITECTURE

Architecture

CNN (Convolution Neural Network)

- Useful to **find patterns for image** recognition
- Learn directly from the data and classify images using pattern



EXPERIMENT

Image Classification using CIFAR-10

›› Loading CIFAR-10 dataset and check shape

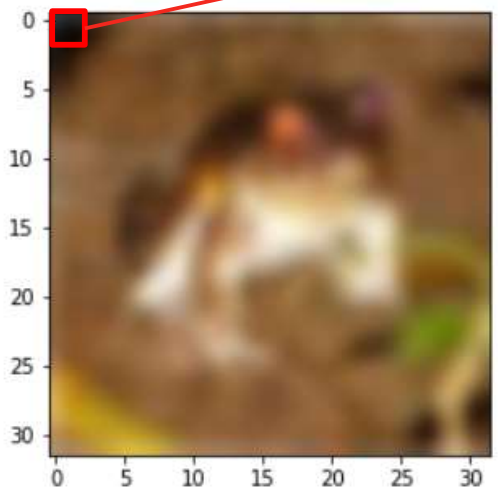
```
In [2]: import tensorflow as tf
        import numpy as np
        import matplotlib.pyplot as plt
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D, BatchNormalization

In [3]: from tensorflow.keras.datasets import cifar10 # CIFAR 데이터 사용

In [4]: (X_train, y_train), (X_test, y_test) = cifar10.load_data() # 데이터 로드

        Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
        170500096/170498071 [==============================] - 11s 0us/step

In [5]: X_train.shape # (50000, 32, 32,3) = (이미지 개수, height, width, channel num)
Out[5]: (50000, 32, 32, 3)

In [6]: X_test.shape
Out[6]: (10000, 32, 32, 3)
```

60,000 image data divided into
50,000 training data and 10,000 test data

Image Classification using CIFAR-10

›› Check Image structure

```
In [8]: plt.imshow(X_train[0], interpolation="bicubic")
Out[8]: <matplotlib.image.AxesImage at 0x7f18731893c8>
```



The values for each pixel
of the **image are stored in an array**

```
In [10]: X_train[0] # 이미지를 array로 나타낸 결과 값
Out[10]: array([[[ 59,  62,  63],
                 [ 43,  46,  45],
                 [ 50,  48,  43],
                 ...,
                 [158, 132, 108],
                 [152, 125, 102],
                 [148, 124, 103]],

                [[ 16,  20,  20],
                 [  0,   0,   0],
                 [ 18,   8,   0],
                 ...,
                 [123,  88,  55],
                 [119,  83,  50],
                 [122,  87,  57]],

                [[ 25,  24,  21],
                 [ 16,   7,   0],
                 [ 49,  27,   8],
                 ...,
                 [118,  84,  50],
                 [120,  84,  50],
                 [109,  73,  42]],

                ...,

                [[208, 170,  96],
                 [201, 153,  34],
                 [198, 161,  26],
                 ...,
                 [160, 133,  70],
                 [ 56,  31,   7],
                 [ 53,  34,  20]],

                [[180, 139,  96],
                 [173, 123,  42],
                 [186, 144,  30],
```

Normalization is required
for optimal results

Image Classification using CIFAR-10

›› Normalization Image array

```
# 0~255 사이에 위치한 픽셀값을 255.로 나누어 0~1 사이에 존재하도록 함.(정규화)
X_train = X_train.astype('float32')/255.
X_test = X_test.astype('float32')/255.
```

```
In [10]: X_train[0] # 이미지를 array로 나타낸 결과
Out[10]: array([[[ 59,  62,  63],
                 [ 43,  46,  45],
                 [ 50,  48,  43],
                 ...,
                 [158, 132, 108],
                 [152, 125, 102],
                 [148, 124, 103]],

                [[ 16,  20,  20],
                 [  0,   0,   0],
                 [ 18,   8,   0],
                 ...,
                 [123,  88,  55],
                 [119,  83,  50],
                 [122,  87,  57]],

                [[ 25,  24,  21],
                 [ 16,   7,   0],
                 [ 49,  27,   8],
                 ...,
                 [118,  84,  50],
                 [120,  84,  50],
                 [109,  73,  42]],
```

Normalization makes that all values exist **between 0 and 1**

```
1 X_train[0]

array([[[0.23137255, 0.24313726, 0.24705882],
        [0.16862746, 0.18039216, 0.1764706 ],
        [0.19607843, 0.1882353 , 0.16862746],
        ...,
        [0.61960787, 0.5176471 , 0.42352942],
        [0.59607846, 0.49019608, 0.4       ],
        [0.5803922 , 0.4862745 , 0.40392157]],

       [[0.0627451 , 0.07843138, 0.07843138],
        [0.        , 0.        , 0.        ],
        [0.07058824, 0.03137255, 0.        ],
        ...,
        [0.48235294, 0.34509805, 0.21568628],
        [0.46666667, 0.3254902 , 0.19607843],
        [0.47843137, 0.34117648, 0.22352941]],

       [[0.09803922, 0.09411765, 0.08235294],
        [0.0627451 , 0.02745098, 0.        ],
        [0.19215687, 0.10588235, 0.03137255],
        ...,
        [0.4627451 , 0.32941177, 0.19607843],
        [0.47058824, 0.32941177, 0.19607843],
        [0.42745098, 0.28627452, 0.16470589]],
```

Image Classification using CIFAR-10

›› Model define and compile

```
# 모델 설계
model = Sequential()

model.add(Conv2D(128, kernel_size=(5,5), activation='relu', padding="same", input_shape = X_train.shape[1:]))
model.add(BatchNormalization())
model.add(MaxPooling2D())


model.add(Conv2D(128, kernel_size=(5,5), padding="same", activation='relu')) # 64->128
model.add(BatchNormalization())
model.add(Dropout(0.15)) # 0.25 -> 0.15

model.add(Conv2D(64, kernel_size=(3,3), padding="same", activation='relu')) ## 32->64
model.add(BatchNormalization())
model.add(Dropout(0.15))

model.add(Conv2D(64, kernel_size=(3,3),  padding="same", activation='relu')) ## 추가  ## 32-> 64
model.add(BatchNormalization())
model.add(Dropout(0.15))

model.add(Conv2D(64, kernel_size=(3,3), padding="same", activation='relu')) ## 추가  ## 32-> 64
model.add(BatchNormalization())
model.add(Dropout(0.15))

model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D())
model.add(Dropout(0.15))

model.add(Flatten())
# 덴스를 줄다 넓는게 좋을 수도

model.add(Dense(10, activation = 'softmax'))
```

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

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 32, 32, 128)       9728
_________________________________________________________________
batch_normalization_6 (Batch (None, 32, 32, 128)       512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 16, 16, 128)       409728
_________________________________________________________________
batch_normalization_7 (Batch (None, 16, 16, 128)       512
_________________________________________________________________
dropout_5 (Dropout)          (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 16, 16, 64)        73792
_________________________________________________________________
batch_normalization_8 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
dropout_6 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
batch_normalization_9 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
dropout_7 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 16, 16, 64)        36928
_________________________________________________________________
batch_normalization_10 (Batc (None, 16, 16, 64)        256
_________________________________________________________________
dropout_8 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 14, 14, 32)        18464
_________________________________________________________________
batch_normalization_11 (Batc (None, 14, 14, 32)        128
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
dropout_9 (Dropout)          (None, 7, 7, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1568)              0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                15690
=================================================================
Total params: 603,178
Trainable params: 602,218
Non-trainable params: 960
_________________________________________________________________
```

Image Classification using CIFAR-10

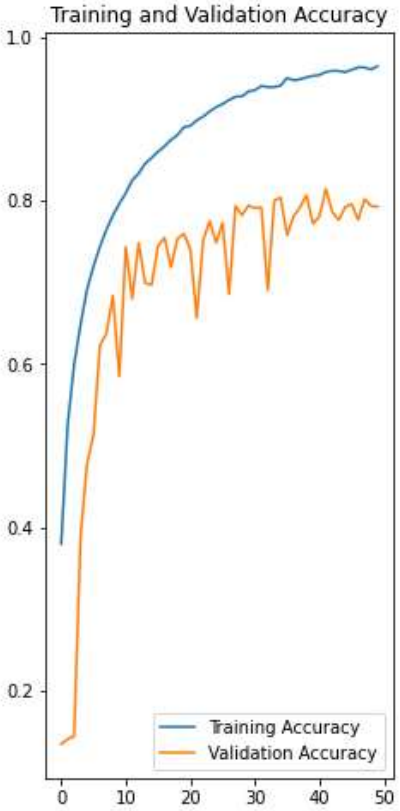
›› Fit model

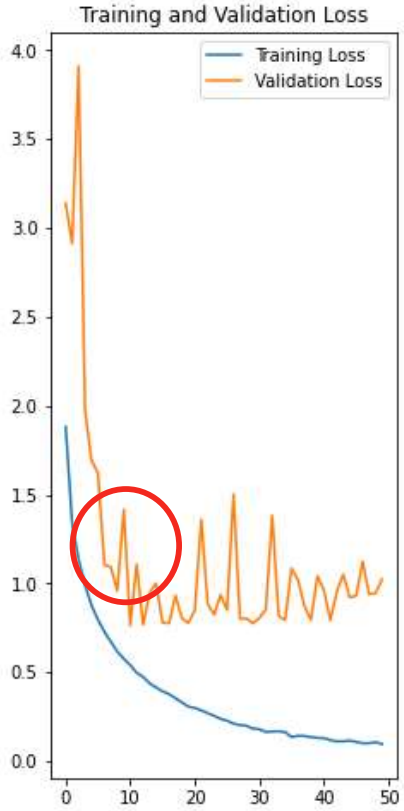
```
# 에포크 50까지 다 도는 경우
history = model.fit(X_train, Y_train, batch_size=batchSize, epochs=num_epochs, shuffle=True, validation_data=(X_test, Y_test))
```

```
Epoch 1/50
98/98 [==============================] - 21s 212ms/step - loss: 1.8809 - accuracy: 0.3796 - val_loss: 3.1391 - val_accuracy: 0.1344
Epoch 2/50
98/98 [==============================] - 20s 208ms/step - loss: 1.3366 - accuracy: 0.5261 - val_loss: 2.9113 - val_accuracy: 0.1403
Epoch 3/50
98/98 [==============================] - 20s 208ms/step - loss: 1.1332 - accuracy: 0.5993 - val_loss: 3.9097 - val_accuracy: 0.1441
Epoch 4/50
98/98 [==============================] - 20s 207ms/step - loss: 0.9905 - accuracy: 0.6488 - val_loss: 1.9713 - val_accuracy: 0.3894
Epoch 5/50
98/98 [==============================] - 20s 207ms/step - loss: 0.8760 - accuracy: 0.6911 - val_loss: 1.6904 - val_accuracy: 0.4775
Epoch 6/50
98/98 [==============================] - 20s 208ms/step - loss: 0.7957 - accuracy: 0.7198 - val_loss: 1.6216 - val_accuracy: 0.5134
Epoch 7/50
98/98 [==============================] - 20s 208ms/step - loss: 0.7280 - accuracy: 0.7440 - val_loss: 1.1056 - val_accuracy: 0.6228
Epoch 8/50
98/98 [==============================] - 20s 207ms/step - loss: 0.6719 - accuracy: 0.7645 - val_loss: 1.0910 - val_accuracy: 0.6372
Epoch 9/50
98/98 [==============================] - 20s 207ms/step - loss: 0.6158 - accuracy: 0.7818 - val_loss: 0.9601 - val_accuracy: 0.6837
Epoch 10/50
98/98 [==============================] - 20s 207ms/step - loss: 0.5758 - accuracy: 0.7966 - val_loss: 1.4157 - val_accuracy: 0.5847

Epoch 45/50
98/98 [==============================] - 20s 207ms/step - loss: 0.1156 - accuracy: 0.9575 - val_loss: 0.9217 - val_accuracy: 0.7925
Epoch 46/50
98/98 [==============================] - 20s 207ms/step - loss: 0.1092 - accuracy: 0.9605 - val_loss: 0.9300 - val_accuracy: 0.7959
Epoch 47/50
98/98 [==============================] - 20s 208ms/step - loss: 0.1012 - accuracy: 0.9633 - val_loss: 1.1235 - val_accuracy: 0.7767
Epoch 48/50
98/98 [==============================] - 20s 208ms/step - loss: 0.1009 - accuracy: 0.9634 - val_loss: 0.9395 - val_accuracy: 0.8022
Epoch 49/50
98/98 [==============================] - 20s 208ms/step - loss: 0.1072 - accuracy: 0.9608 - val_loss: 0.9447 - val_accuracy: 0.7936
Epoch 50/50
98/98 [==============================] - 20s 208ms/step - loss: 0.0965 - accuracy: 0.9649 - val_loss: 1.0224 - val_accuracy: 0.7927
```

Image Classification using CIFAR-10

›› Visualization & Evaluation






```
loss_and_acc = model.evaluate(X_test, Y_test, batch_size=batchSize)
print('Test set Loss and Accuracy')
print(loss_and_acc)
```

```
20/20 [==============================] - 1s 55ms/step - loss: 1.0224 - accuracy: 0.7927
Test set Loss and Accuracy
[1.0223722457885742, 0.7926999926567078]
```

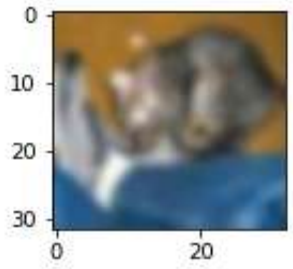
- Validation loss decreases and then **increase again.**
- Through this, can be determined to result in overfitting

Image Classification using CIFAR-10

›› Validation

```
# 모델 사용
labels = ['비행기', '자동차', '새','고양이','사슴','개','개구리','말','배','트럭']

# 출력 그림의 크기를 결정합니다.
plt.rcParams["figure.figsize"] = (2,2)
# Test Set의 10개를 맞추어 봅시다.
for i in range(10):
  # 모델 사용
  output = model.predict(X_test[i].reshape(1,32, 32, 3))

  # 이미지 출력
  plt.imshow(X_test[i].reshape(32, 32, 3), interpolation="bicubic")
  # np.argmax()가 labels의 인덱스가 되어 labels 배열에 있는 문자열을 출력합니다.
  print('예측: ' + labels[np.argmax(output)] + ' / 정답: ' + labels[np.argmax(Y_test[i])])
  plt.show()
```

```
예측: 개 / 정답: 고양이
```



Predictive accuracy is not high!

Image Classification using CIFAR-10

›› Changed model fit

Try using EarlyStopping() to changed model performance

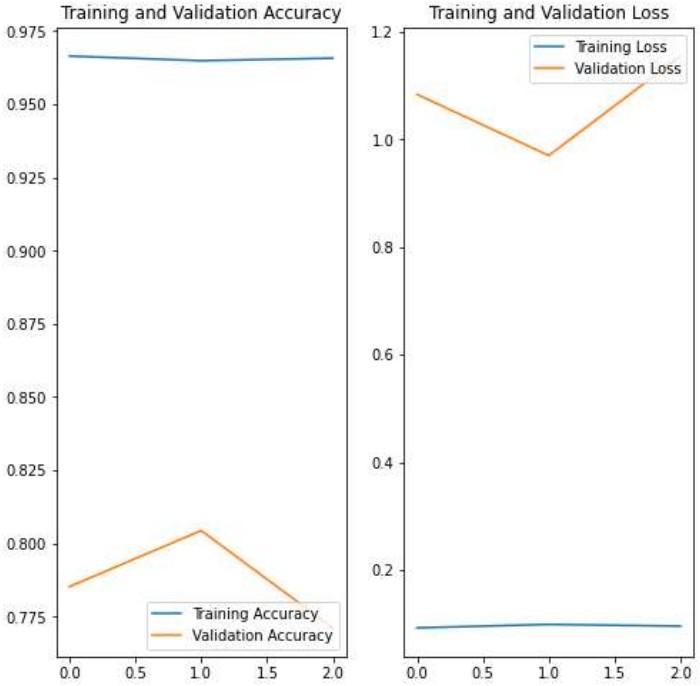
```
from keras.callbacks import EarlyStopping
early_stopping = EarlyStopping() # 조기 종료

# overfitting 이 생기는 경우 종료
history = model.fit(X_train, Y_train, batch_size=batchSize, epochs=num_epochs, shuffle=True, validation_data=(X_test, Y_test), callbacks=[early_stop
ping])
```

```
Epoch 1/50
98/98 [==============================] - 20s 208ms/step - loss: 0.0920 - accuracy: 0.9664 - val_loss: 1.0840 - val_accuracy: 0.7853
Epoch 2/50
98/98 [==============================] - 20s 206ms/step - loss: 0.0983 - accuracy: 0.9649 - val_loss: 0.9703 - val_accuracy: 0.8044
Epoch 3/50
98/98 [==============================] - 20s 207ms/step - loss: 0.0952 - accuracy: 0.9657 - val_loss: 1.1553 - val_accuracy: 0.7712
```

- What is EarlyStopping()?
  - **Stop learning if performance no longer increases** when learning
  - **The validation accuracy of the model is no longer lowered.**
  - When training is stopped, the validation error condition is generally higher than the previous model

Image Classification using CIFAR-10

›› Visualization & Evaluation using Improved model



Previous model result

```
loss_and_acc = model.evaluate(X_test, Y_test, batch_size=batchSize)
print('Test set Loss and Accuracy')
print(loss_and_acc)
```

```
20/20 [==============================] - 1s 55ms/step - loss: 1.0224 - accuracy: 0.7927
Test set Loss and Accuracy
[1.0223722457885742, 0.7926999926567078]
```

Current model result

```
loss_and_acc = model.evaluate(X_test, Y_test, batch_size=batchSize)
print('Test set Loss and Accuracy')
print(loss_and_acc)
```

```
20/20 [==============================] - 1s 55ms/step - loss: 1.1553 - accuracy: 0.7712
Test set Loss and Accuracy
[1.1553255319595337, 0.7712000012397766]
```

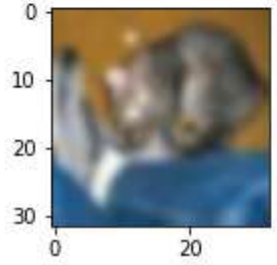
- The difference between training accuracy and validation accuracy is large
- Also, this accuracy lacks compared to previous

Image Classification using CIFAR-10

›› Validation

```
# 모델 사용
labels = ['비행기', '자동차', '새','고양이','사슴','개','개구리','말','배','트럭']

# 출력 그림의 크기를 결정합니다.
plt.rcParams["figure.figsize"] = (2,2)
# Test Set의 10개를 맞추어 봅시다.
for i in range(10):
  # 모델 사용
  output = model.predict(X_test[i].reshape(1,32, 32, 3))

  # 이미지 출력
  plt.imshow(X_test[i].reshape(32, 32, 3), interpolation="bicubic")
  # np.argmax()가 labels의 인덱스가 되어 labels 배열에 있는 문자열을 출력합니다.
  print('예측: ' + labels[np.argmax(output)] + ' / 정답: ' + labels[np.argmax(Y_test[i])])
  plt.show()
```

```
예측: 개 / 정답: 고양이
```



Changed model's predictive accuracy also is not high!

Image Classification using CIFAR-10

›› Improved model fit

```
from keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(patience=5) # 과적합이 발생해도 에포크 5만큼 기다린 후 종료

history = model.fit(X_train, Y_train, batch_size=batchSize, epochs=num_epochs, shuffle=True, validatio
n_data=(X_test, Y_test), callbacks=[early_stopping])
```

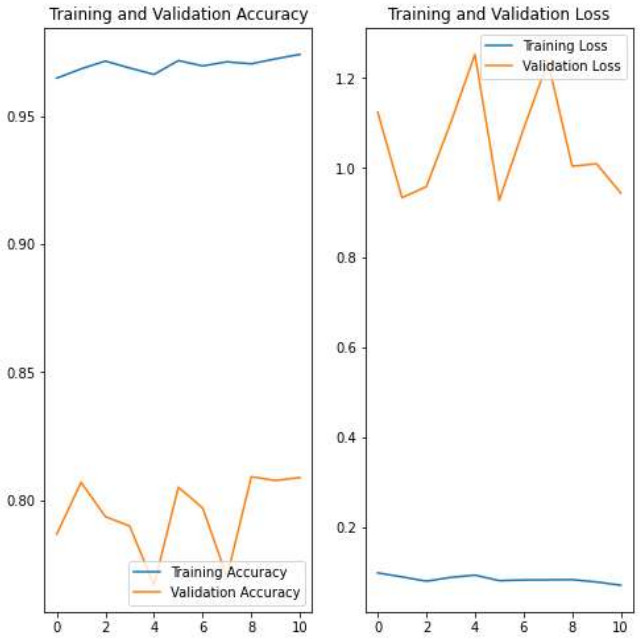
```
Epoch 1/50
98/98 [==============================] - 20s 207ms/step - loss: 0.0975 - accuracy: 0.9649 - val_loss:
1.1246 - val_accuracy: 0.7867
Epoch 2/50
98/98 [==============================] - 20s 206ms/step - loss: 0.0885 - accuracy: 0.9685 - val_loss:
0.9336 - val_accuracy: 0.8069
Epoch 3/50
98/98 [==============================] - 20s 206ms/step - loss: 0.0790 - accuracy: 0.9715 - val_loss:
0.9583 - val_accuracy: 0.7935
Epoch 4/50
98/98 [==============================] - 20s 206ms/step - loss: 0.0874 - accuracy: 0.9688 - val_loss:
1.1003 - val_accuracy: 0.7898
Epoch 5/50
98/98 [==============================] - 20s 206ms/step - loss: 0.0925 - accuracy: 0.9663 - val_loss:
1.2532 - val_accuracy: 0.7667
Epoch 6/50
98/98 [==============================] - 20s 207ms/step - loss: 0.0802 - accuracy: 0.9717 - val_loss:
0.9279 - val_accuracy: 0.8050
```

When using EarlyStopping, **set patience to 5**

to wait for 5 epoch even if over fitting occurs.

Image Classification using CIFAR-10

›› Visualization & Evaluation



Previous model result

```
20/20 [==============================] - 1s 55ms/step - loss: 1.0224 - accuracy: 0.7927
Test set Loss and Accuracy
[1.0223722457885742, 0.7926999926567078]
```

Changed model result

```
20/20 [==============================] - 1s 55ms/step - loss: 1.1553 - accuracy: 0.7712
Test set Loss and Accuracy
[1.1553255319595337, 0.7712000012397766]
```

Improved model result

```
20/20 [==============================] - 1s 55ms/step - loss: 0.9439 - accuracy: 0.8088
Test set Loss and Accuracy
[0.943939566122437, 0.8087999820709229]
```

Over fitting occurred while waiting 5 patient,

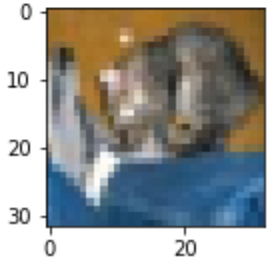
Improved test accuracy compared to previous models.

Image Classification using CIFAR-10

›› Validation

```
# 모델 사용
labels = ['비행기', '자동차', '새','고양이','사슴','개','개구리','말','배','트럭']


# 출력 그림의 크기를 결정합니다.
plt.rcParams["figure.figsize"] = (2,2)
# Test Set의 10개를 맞추어 봅시다.
for i in range(10):
  # 모델 사용
  output = model.predict(X_test[i].reshape(1,32, 32, 3))

  # 이미지 출력
  plt.imshow(X_test[i].reshape(32, 32, 3))
  # np.argmax()가 labels의 인덱스가 되어 labels 배열에 있는 문자열을 출력합니다.
  print('예측: ' + labels[np.argmax(output)] + ' / 정답: ' + labels[np.argmax(Y_test[i])])
  plt.show()
```

```
예측: 고양이 / 정답: 고양이
```



Accurately predict images that previous models did not predict

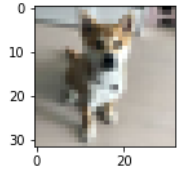
Image Classification using CIFAR-10

Testing using new data

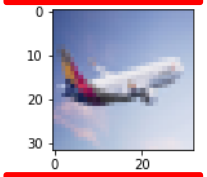
```
# 결과값 예측
output = []
for i in range(len(imgs)):
    pred =  model.predict(X[i].reshape(1, 32, 32, 3))
    output.append(pred)
```

```
for i in range(len(imgs)):
    plt.imshow(X[i].reshape(32, 32, 3))
    print("Prediction: ", labels[np.argmax(output[i])])
    plt.show()
```

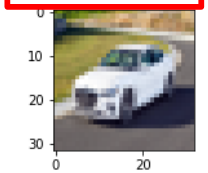
Prediction: 개



Prediction: 비행기



Prediction: 자동차



- Testing performed on the most accurate model using new image
- Verify that prediction is going well

CONCLUSION

Conclusion

Use a significant amount of data to process image classification.

Initial learning of image classification leads to poor layer design and accuracy.

However, it was good to learn about functions such as EarlyStopping
after experiencing over-compatibility.

I would like to improve the model with the same data and deeper knowledge later.

Thank you

Transfer learning

In Internship program

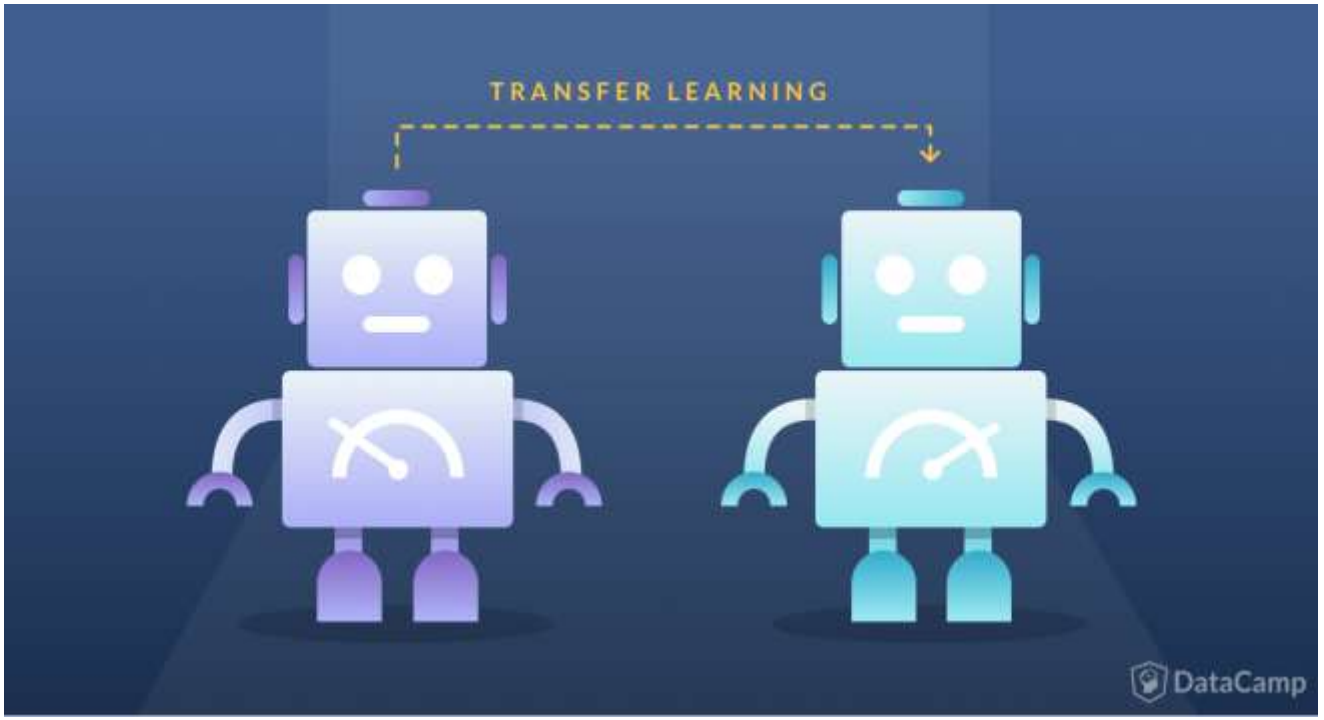
INDEX

- Introduction
- Dataset
- Architecture
- Experiment
- Conclusion

INTRODUCTION

What did I do a project?

- Implementing Image Classification Model with very little data
- Use **Transfer learning** to increase model performance



Why did I do a project?

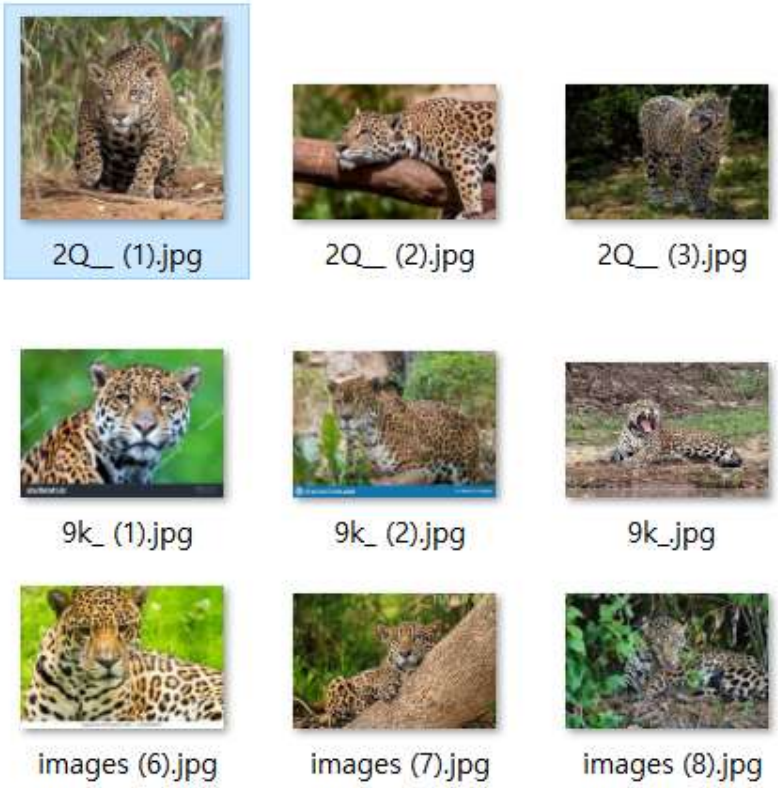
- Learning models based on rich data without missing values are likely to show good performance.
- However, learning data is not abundant in all areas.

So I went ahead with the project to practice overcoming these limitations.

DATASET

Dataset

Used leopard – jaguar dataset



- Class: **2 classes**
- Image size

: Each class has **50 color image data**

Image size and pixels are different

- Label

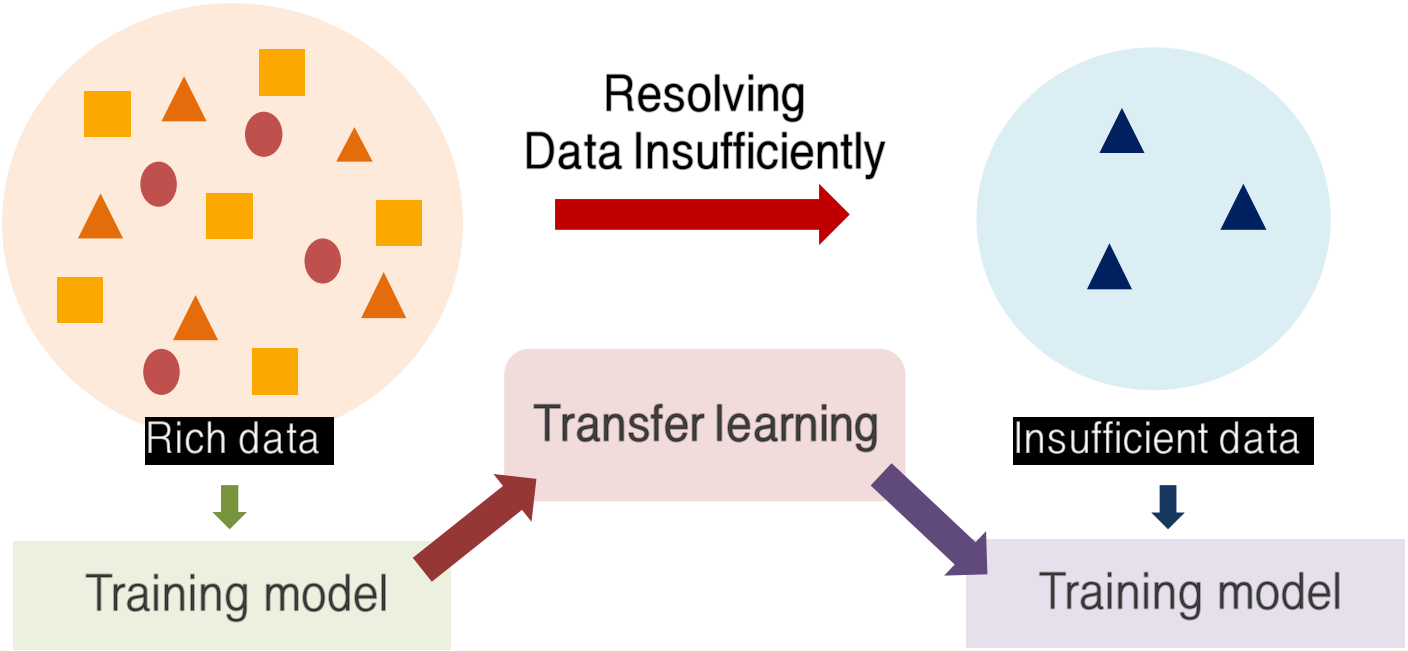
: [leopard, jaguar]

ARCHITECTURE

Transfer learning

›› Transfer learning

- Learning technique for **reusing trained models in data-rich areas** to build models in areas **where learning data is scarce**



EXPERIMENT

Image Classification using transfer learning

Loading data and Generating data

```
from keras.preprocessing.image import ImageDataGenerator
```

```
train_image_gen = ImageDataGenerator(rescale=1/255.,
                                     horizontal_flip = True,
                                     vertical_flip = True,
                                     fill_mode = 'nearest',
                                     rotation_range=15)
```

Used to transform data to create new learning data

Ex) horizontal flip, vertical flip, rotation

```
import matplotlib.pyplot as plt

new_training_images = []

i = 0
for x in sample_training_images[i]:
    #plt.imshow(x)
    x = x.reshape((1,) + x.shape)
    j=0
    for batch in train_image_gen.flow(x):
        new_training_images.append(batch)
        j+=1
        #print(i)
        if j==20:
            break

print(len(new_training_images))
```

```
400
```

Increase 50 images to 400 images using ImageDataGenerator

Image Classification using transfer learning

» Loading saved data divided by train, validation, and test

```
In [ ]: path = '/content/gdrive/My Drive/Colab Notebooks/original_jag_leo/'
```

```
In [ ]: # jag_dir = path + 'jaguar/'
        # leo_dir = path + 'leopard/'

        train_dir = path + 'train/'
        validation_dir = path + 'validation/'
        test_dir = path + 'test/'
```

```
In [ ]: train_jag_dir = train_dir + 'train_jag/'
        train_leo_dir = train_dir + 'train_leo/'

        test_jag_dir = test_dir + 'test_jag/'
        test_leo_dir = test_dir + 'test_leo/'

        validation_jag_dir = validation_dir + 'validation_jag/'
        validation_leo_dir = validation_dir + 'validation_leo/'
```

```
train_jag_size = len(os.listdir(train_jag_dir))
test_jag_size = len(os.listdir(test_jag_dir))
validation_jag_size = len(os.listdir(validation_jag_dir))

train_leo_size = len(os.listdir(train_leo_dir))
test_leo_size = len(os.listdir(test_leo_dir))
validation_leo_size = len(os.listdir(validation_leo_dir))
```

```
160
70
70
160
70
70
```

Each class has 160 images for training,

70 images for validation, and 70 images for testing

Image Classification using transfer learning

›› Using “Resnet50V2” as a pre-trained model

```
conv_base = ResNet50V2(weights = 'imagenet', include_top= False, input_shape=(img_width, img_height, 3)) # include_top: 상단의 레이어를 포함할 것인지 아닌지 결정하는 파라미터
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50v2_weights_tf_dim_ordering_tf_kernels_notop.h5
94674944/94668760 [==============================] - 1s 0us/step
```

Weights: **Whether to use pre-prepared weights** in Keras

Include_top: **Whether to include the uppermost fully connected layers**

- **Using weight of imagenet** consisting of more than 20,000 classes and a total of 14,197,122 images

- **Does not include the fully connected layer** of resnet

Image Classification using transfer learning

›› Extract features

In [ ]:
```
import os, shutil
from keras.preprocessing.image import ImageDataGenerator
```

In [ ]:
```
datagen = ImageDataGenerator(rescale=1/255.)
batch_size = 32
```

In [ ]:
```
def extract_features(directory, sample_count):
    features = np.zeros(shape = (sample_count,7,7,512))
    labels = np.zeros(shape=(sample_count))

    generator = datagen.flow_from_directory(directory, target_size=(img_width, img_height), batch_size=batch_size, class_mode='binary')

    i=0
    for inputs_batch, labels_batch in generator:
        features_batch = conv_base.predict(inputs_batch)
        features[i * batch_size: (i+1) * batch_size] = features_batch
        labels[i * batch_size : (i+1) * batch_size] = labels_batch
        i+=1
        if i*batch_size >= sample_count:
            break
    return features, labels
```

- Convolutional base is used to extract characteristics
- These vectors enter classifier as input and classify leopard and jaguar

Image Classification using transfer learning

›› Modeling & Classifier – Global average pooling

```
model = models.Sequential()
model.add(layers.GlobalAveragePooling2D(input_shape=(7,7,512)))
model.add(layers.Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d (Gl (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 1)                 513
=================================================================
Total params: 513
Trainable params: 513
Non-trainable params: 0
_________________________________________________________________
```

- The features extracted from convolutional base were **put into Global average pooling**
- And then modelled in the **sigmoid activation function**

Activation function for classifying **two categories** in keras

Image Classification using transfer learning

›› Model compile and fit

```
model.compile(optimizer = optimizers.Adam(), loss='binary_crossentropy', metrics=['acc'])
```

```
history = model.fit(train_features, train_labels,
                    epochs = epochs,
                    batch_size = batch_size,
                    validation_data = (validation_features, validation_labels))
```

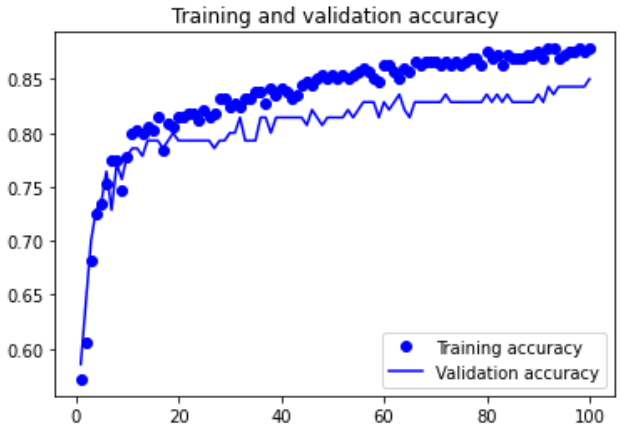
```
Epoch 1/100
10/10 [==============================] - 0s 22ms/step - loss: 0.6842 - acc: 0.5719 - val_loss: 0.6738 - val_acc: 0.5857
Epoch 2/100
10/10 [==============================] - 0s 8ms/step - loss: 0.6703 - acc: 0.6062 - val_loss: 0.6656 - val_acc: 0.6429
Epoch 3/100
10/10 [==============================] - 0s 7ms/step - loss: 0.6596 - acc: 0.6812 - val_loss: 0.6560 - val_acc: 0.7000
Epoch 4/100
10/10 [==============================] - 0s 6ms/step - loss: 0.6499 - acc: 0.7250 - val_loss: 0.6471 - val_acc: 0.7286
Epoch 5/100
10/10 [==============================] - 0s 7ms/step - loss: 0.6404 - acc: 0.7344 - val_loss: 0.6385 - val_acc: 0.7357
Epoch 6/100
10/10 [==============================] - 0s 6ms/step - loss: 0.6315 - acc: 0.7531 - val_loss: 0.6301 - val_acc: 0.7643
Epoch 7/100
10/10 [==============================] - 0s 6ms/step - loss: 0.6242 - acc: 0.7750 - val_loss: 0.6230 - val_acc: 0.7286
Epoch 8/100
10/10 [==============================] - 0s 7ms/step - loss: 0.6150 - acc: 0.7750 - val_loss: 0.6149 - val_acc: 0.7714
Epoch 9/100
10/10 [==============================] - 0s 6ms/step - loss: 0.6083 - acc: 0.7469 - val_loss: 0.6092 - val_acc: 0.7571
Epoch 10/100
10/10 [==============================] - 0s 7ms/step - loss: 0.5985 - acc: 0.7781 - val_loss: 0.6003 - val_acc: 0.7786
```

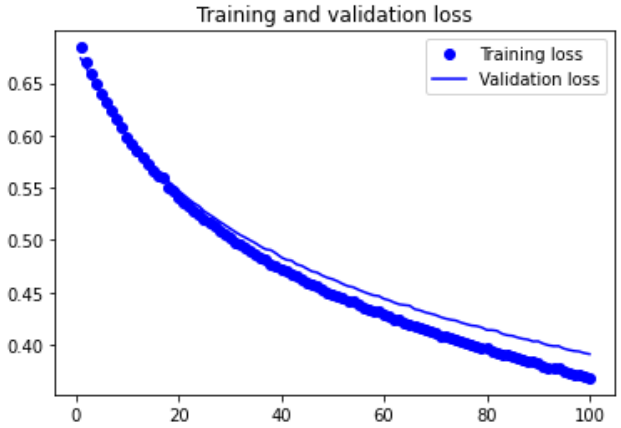
```
Epoch 100/100
10/10 [==============================] - 0s 6ms/step - loss: 0.3680 - acc: 0.8781 - val_loss: 0.3909 - val_acc: 0.8500
```

- Both training accuracy and validation accuracy are good
- Both training loss and validation loss are also low

Image Classification using transfer learning

Visualization



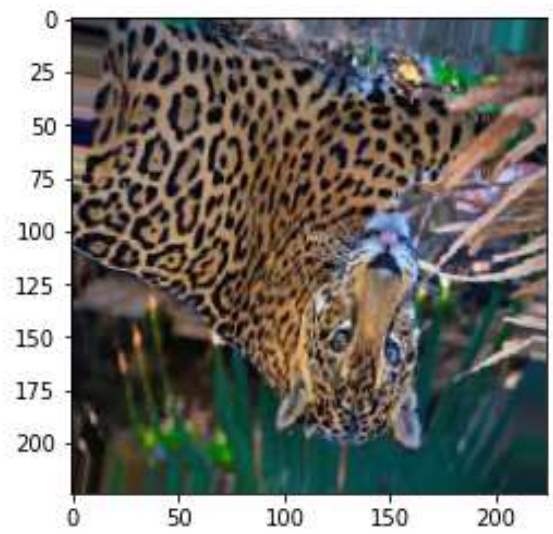


- a low overfit
- Loss function values continue to decrease to the end of the graph
- Perhaps increasing the number of epoch will allow the model to continue learning

Image Classification using transfer learning

Testing

```
visualize_predictoins(model, 5)
```



Jaguar

=

Original Image – Jaguar

Testing performed well with high accuracy

CONCLUSION

Conclusion

In the future, I will continue to study deep learning, and when I do my own projects,

There may be situations where data is difficult to obtain.

As I was working on this project, I could learn a lot about what to do in the above situation.

Object Detection Project

In Internship program

INDEX

- Introduction
- Method
- Preprocessing
- Experiment
- Conclusion

INTRODUCTION

What did I do a project?

- **Sensitive data detection** from document images using YOLOv3
- **ID card detection** using YOLOv3

Brief Introduction

Input Image

Deep learning model

output Image

Why did I do a project?

In internship program, **learned about image processing**

So, I Worked on a project using object detection that is based on image processing

METHOD

YOLOv3

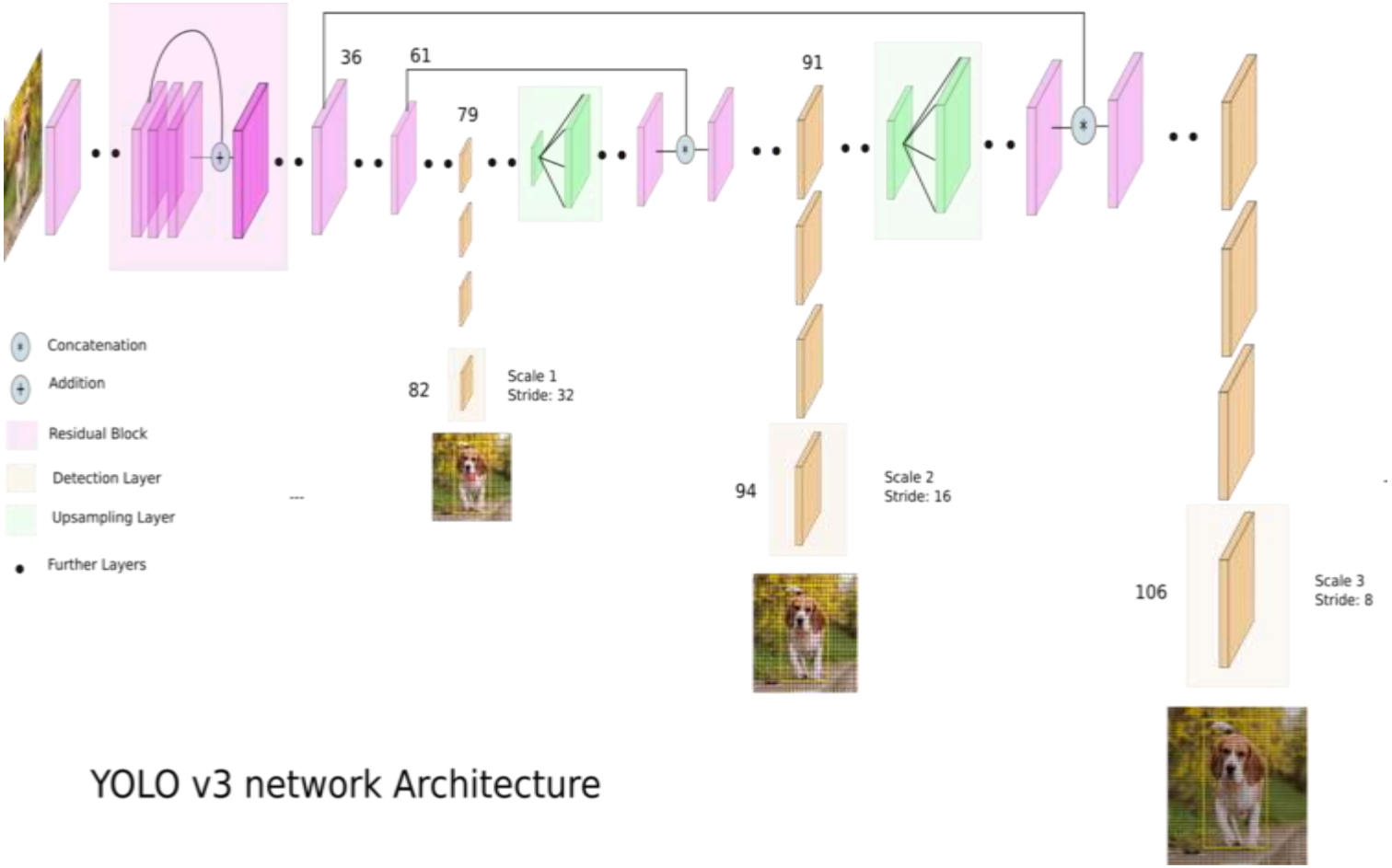
» YOLOv3 is object detection algorithm

» Operating principle

- Divide the image into areas
- Predict bounding boxes and probabilities for each area

» Trained in a data set with COCO that has 80 different object classes

YOLOv3



YOLO v3 network Architecture

YOLOv3

YOLOv3 is object detection algorithm

BUT, The data that I will train is not included in "COCO"

Operating principle

- Divide the image into areas
- Predict bounding boxes and probabilities for each area

So I done data collection and preprocessing

Trained in a data set with COCO that has 80 different object classes

Interfaces for ML Models

- Generate an easy-to-use UI for ML model
- Integrate directly into Python notebook
- Can share a link with anyone

PRE-PROCESSING

Data processing

1. Data Generation

Sample Image

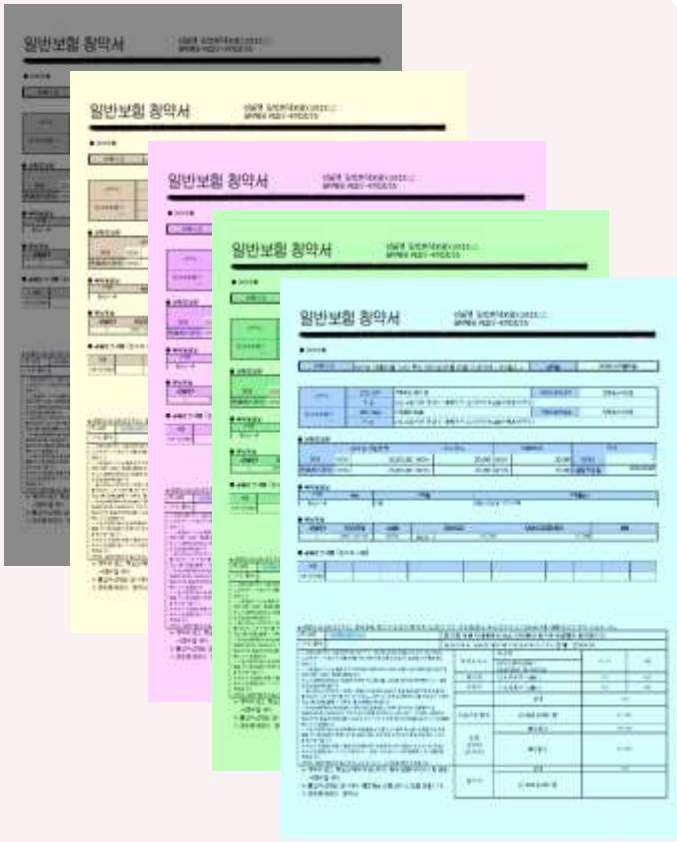


Image Generating

Data processing

1. Data Generating

Sample Image

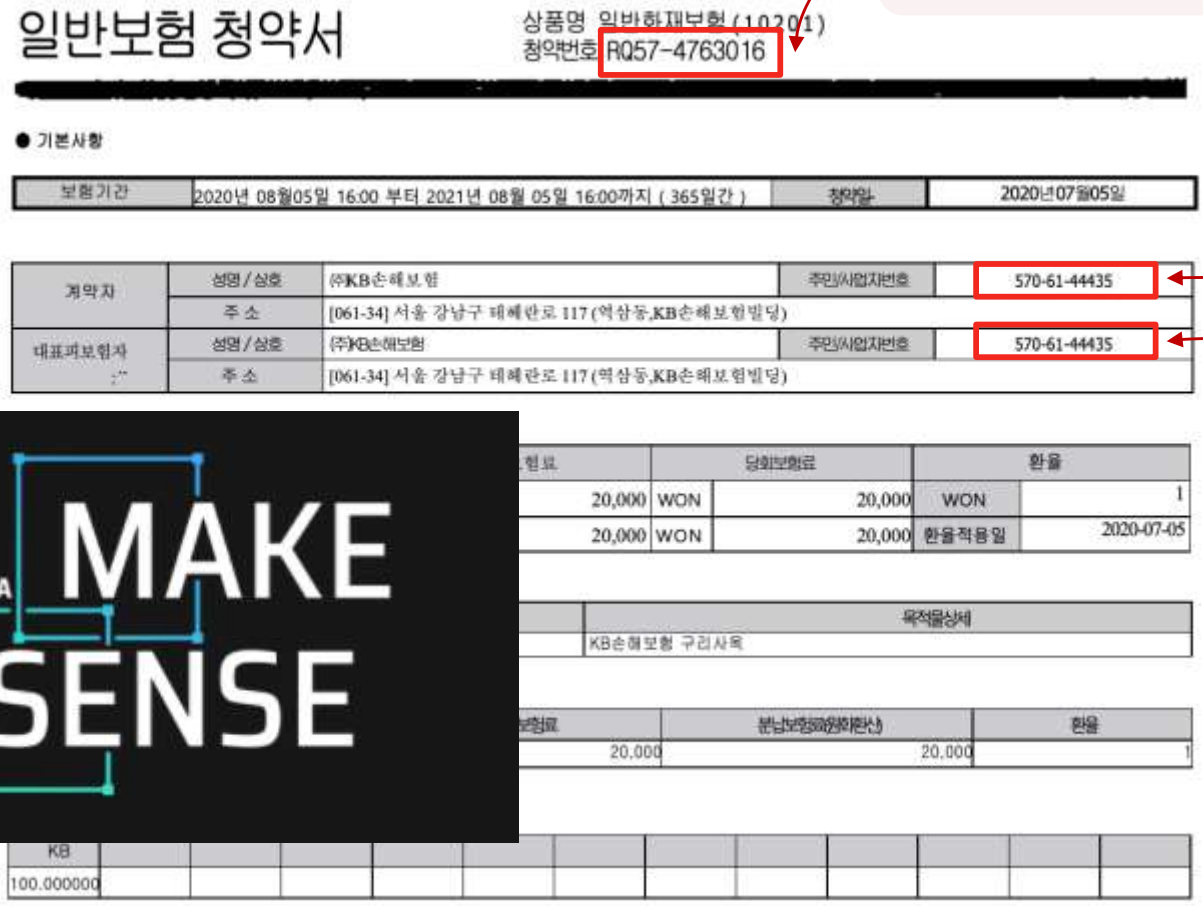
Extending to 460 Images

Image Generating

Data processing

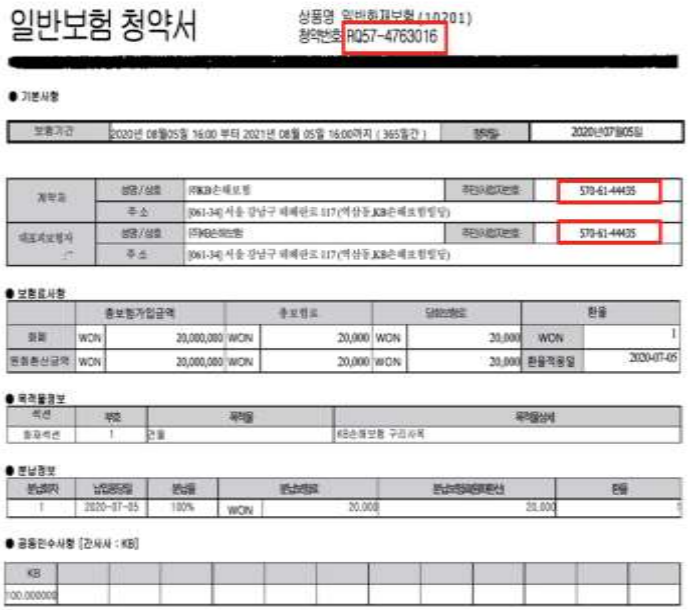
2. Data Labeling

Set the area to detect using Make Sense.ai



Data processing

2. Data Labeling


일반보험 청약서

상품명 일반화재보험 (10201)
청약번호 RQ57-47630016

● 기본사항

보험기간	2020년 08월05일 16:00 부터 2021년 08월 05일 16:00까지 (365일간)	청약일	2020년07월05일

계약자	성명/상호	(주)KB손해보험	주민(사업자)번호	570-61-44435
	주소	[061-34] 서울 강남구 테헤란로 117(역삼동,KB손해보험빌딩)		
피보험자	성명/상호	(주)KB손해보험	주민(사업자)번호	570-61-44435
	주소	[061-34] 서울 강남구 테헤란로 117(역삼동,KB손해보험빌딩)		

● 보험료사항

	총보험가입금액		총보험료		납입보험료		환율	
화재	WON	20,000,000	WON	20,000	WON	20,000	WON	1
원화환산금액	WON	20,000,000	WON	20,000	WON	20,000	환율적용일	2020-07-05

● 목적물정보

목적	부호	목적물	목적물상세
화재목적	1	건물	KB손해보험 구리사옥

● 분납정보

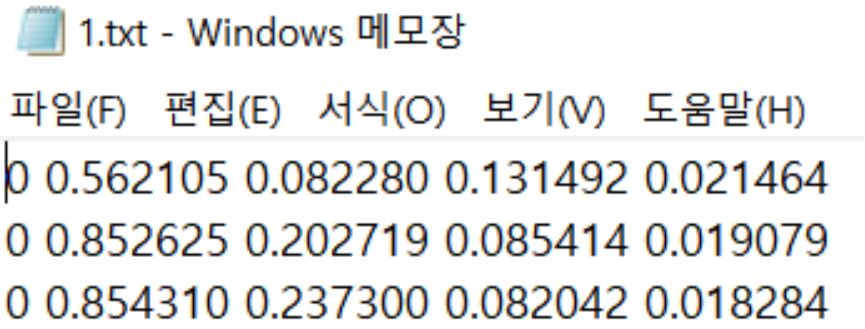
분납회차	납입예정일	분납률	분납보험료	분납보험료(원화환산)	환율
1	2020-07-05	100%	WON 20,000	20,000	1

● 공동인수사항 [간사사 : KB]

KB
100.000000

Labeled Image

A .txt file is created containing the label and x, y coordinate values for the selected area


1.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

0 0.562105 0.082280 0.131492 0.021464
0 0.852625 0.202719 0.085414 0.019079
0 0.854310 0.237300 0.082042 0.018284

Labeling Output

Data processing

2. Data Labeling

Set the area to detect using Make Sense.ai

Data processing

2. Data Labeling

Labeled Image

A .txt file is created containing the label and x, y coordinate values for the selected area

CA13_01.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

0 0.484712 0.512899 0.834289 0.391892

Labeling Output

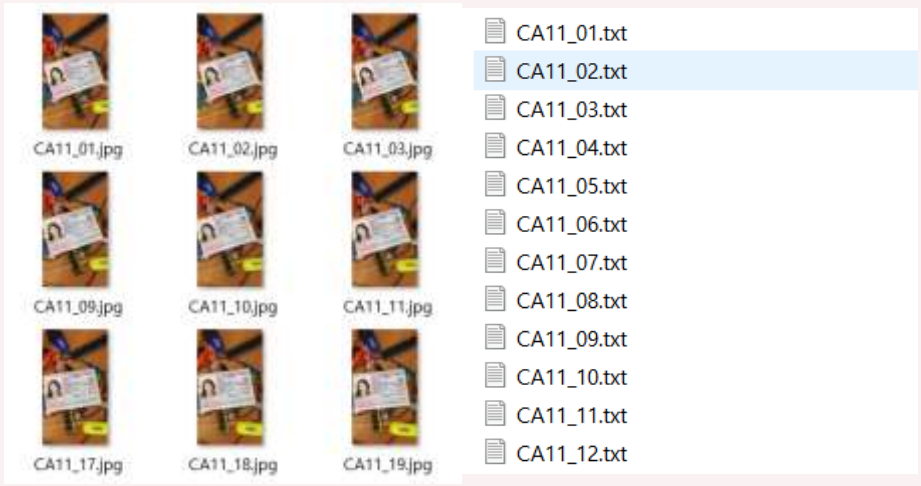
EXPERIMENT

– ID card detection

ID card detection

Dataset ftp://smartengines.com/midv-500/dataset/

- I used the dataset that can get online
- Download 3,000 image data and proceed labelling
- But, only used 350 data for quick results



350 Dataset

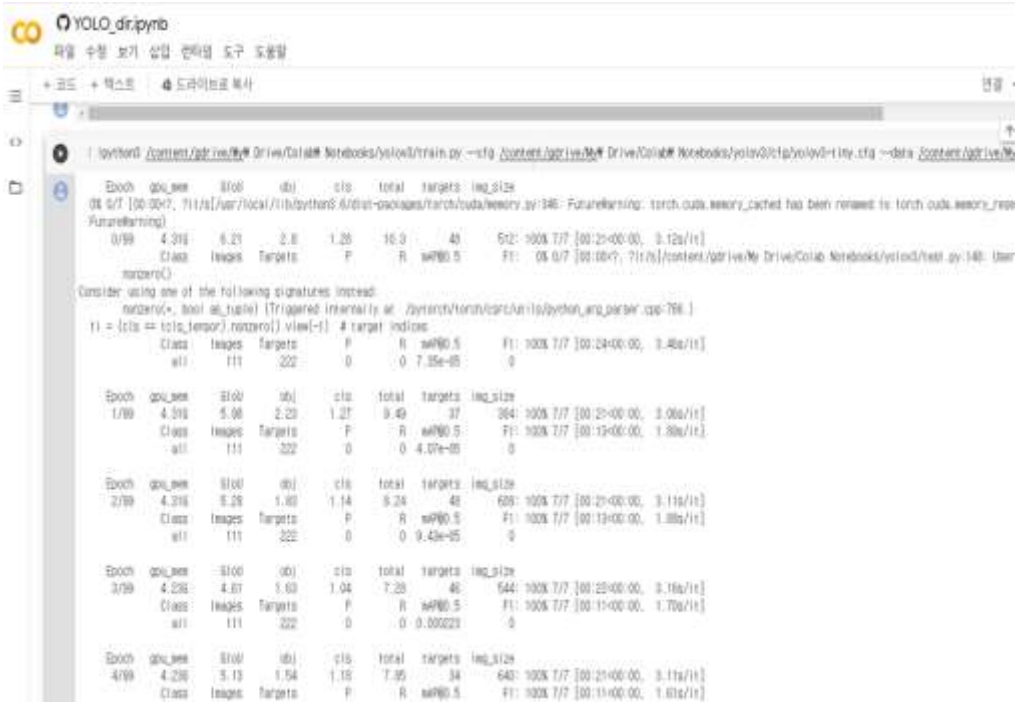
Training data: 300

Test data: 50

ID card detection

›› Training model

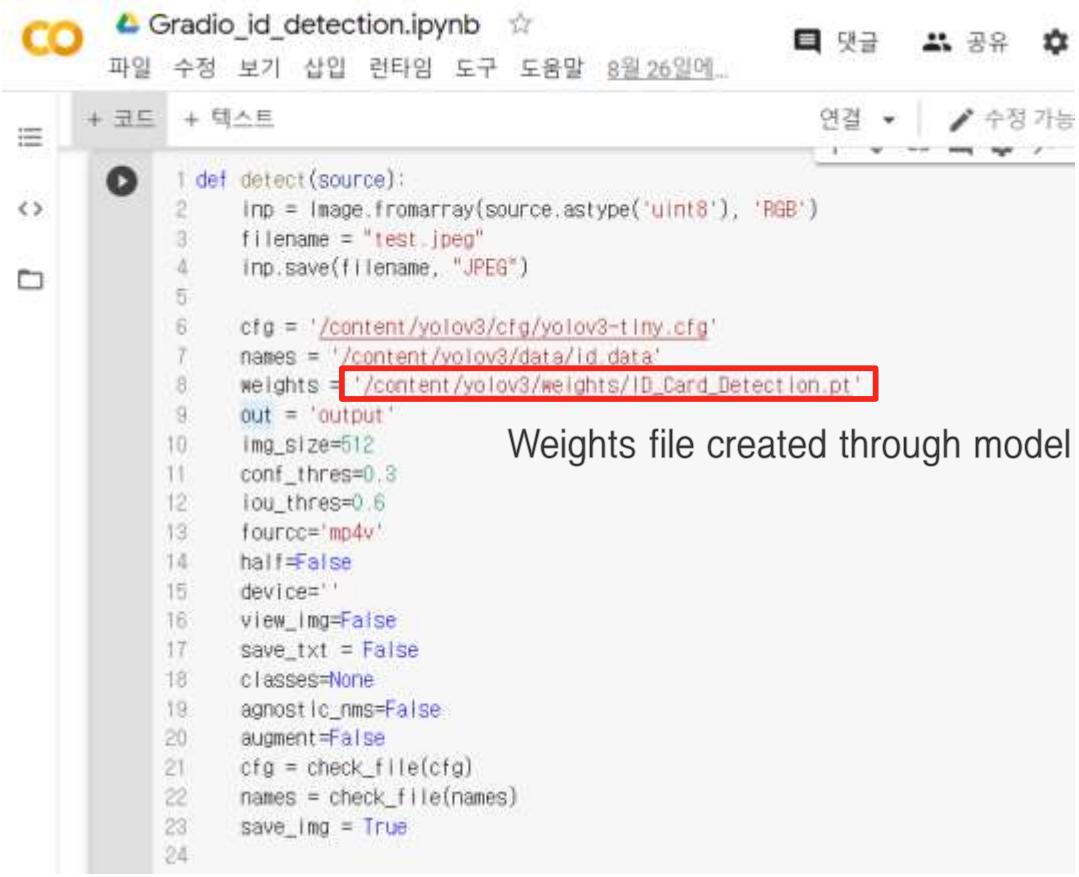
Train with **300 image files and txt files**



At the end of the training, a **weight file** is created, where **the trained results are stored**

ID card detection

›› Execution of training model



Weights file created through model training

Execute with weight file previously created through model training

ID card detection

Connect to Gradio.app

Gradio_id_detection.ipynb ☆

파일 수정 보기 삽입 런타임 도구 도움말 8월 26일에 마지막으로 수정됨

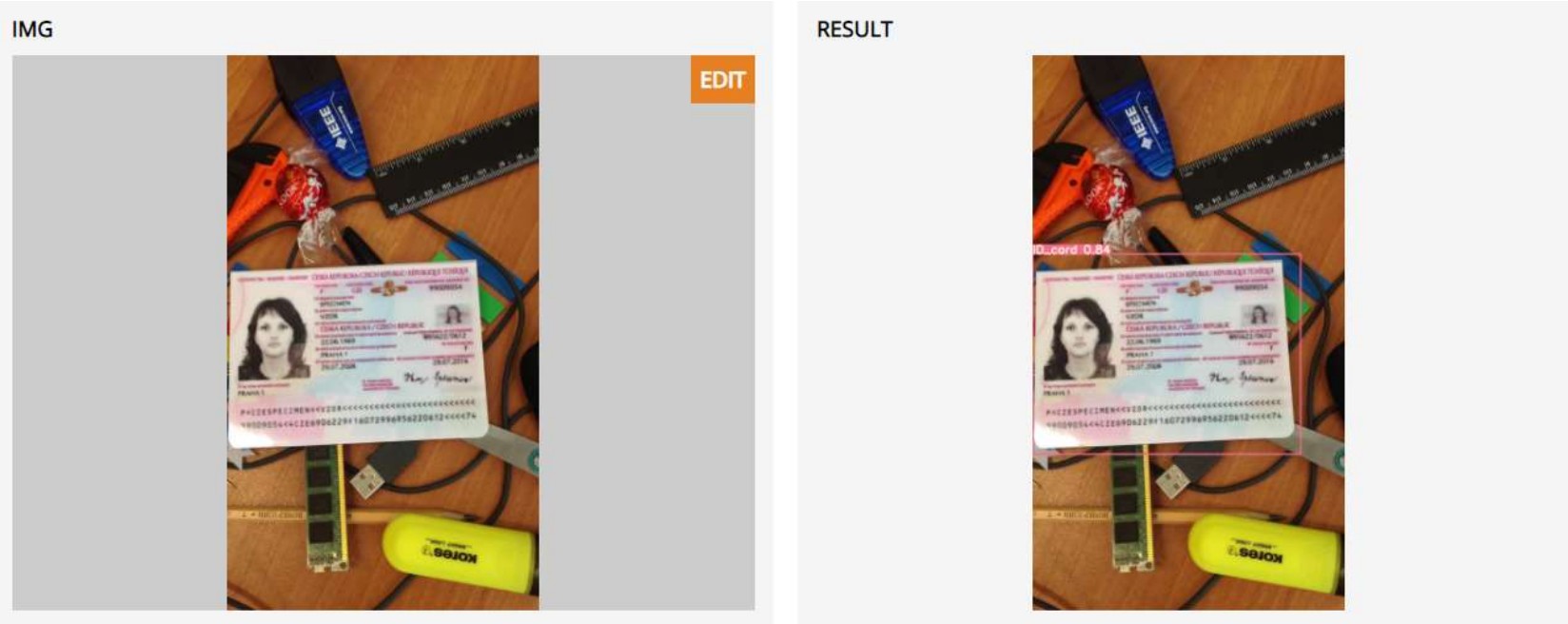
+ 코드 + 텍스트

```
170 image = gr.inputs.Image()
171 label = gr.outputs.Image(label = 'ID Card Detection', type='file')
172 gr.Interface(fn=detect, inputs=image, outputs=label, capture_session=True).launch(debug=True)
```

Connect so that results can be seen in the Gradio.app interface

ID card detection

›› Check the training result in Gradio.app



Uploading a photo to be tested will result in the detection of an Id card on the right

Object detected well BUT Accuracy is 85%, not so high

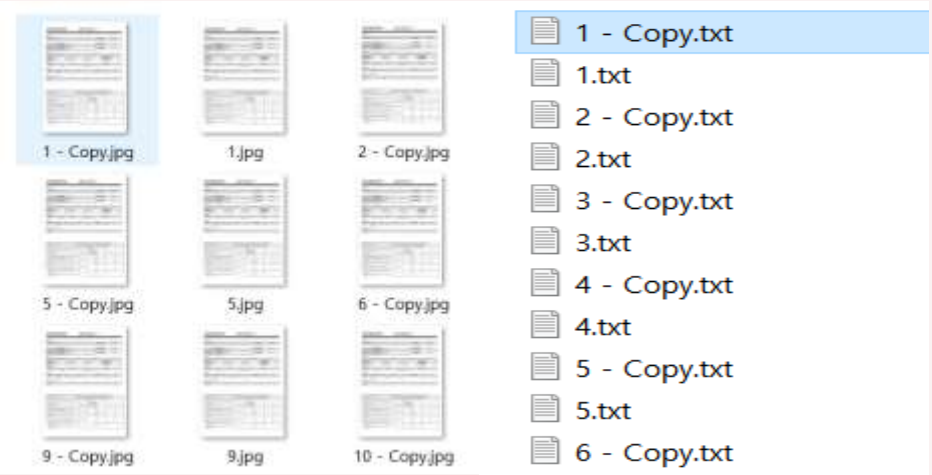
EXPERIMENT

– Sensitive label detection

Sensitive label detection

›› Dataset

- View printed documents and create similar sample document
- Generate data by different filters and numeric values
- Used 460 document image data



460 Dataset

Training data: 400

Test data: 60

Sensitive label detection

›› Training model

Train with **400 image files and txt files**

At the end of the training, a **weight file** is created, where **the trained results are stored**

ID card detection

›› Execution of training model

Gradio_Sensitive_label.ipynb ☆

파일 수정 보기 삽입 런타임 도구 도움말 모든 변경사항이 저장됨

+ 코드 + 텍스트

```
def detect(source):
    inp = Image.fromarray(source.astype('uint8'), 'RGB')
    filename = "test.jpeg"
    inp.save(filename, "JPEG")

    cfg = '/content/grive/My Drive/Colab Notebooks/sensitive_label_detection/yolov3/cfg/yolov3-tiny.cfg'
    names = '/content/grive/My Drive/Colab Notebooks/sensitive_label_detection/yolov3/data/coco_id.names'
    weights = '/content/grive/My Drive/Colab Notebooks/sensitive_label_detection/yolov3/weights/last.pt'
    out = 'output'
    img_size=512
    conf_thres=0.3
    iou_thres=0.6
    fourcc='mp4v'
    half=False
    device=''
    view_img=False
    save_txt = False
    classes=None
    agnostic_nms=False
    augment=False
    cfg = check_file(cfg)
    names = check_file(names)
    save_img = True
```

Weights file created through model training

Execute with weight file previously created through model training

ID card detection

Connect to Gradio.app

Gradio_Sensitive_label.ipynb ☆

파일 수정 보기 삽입 런타임 도구 도움말 모든 변경사항이 저장됨

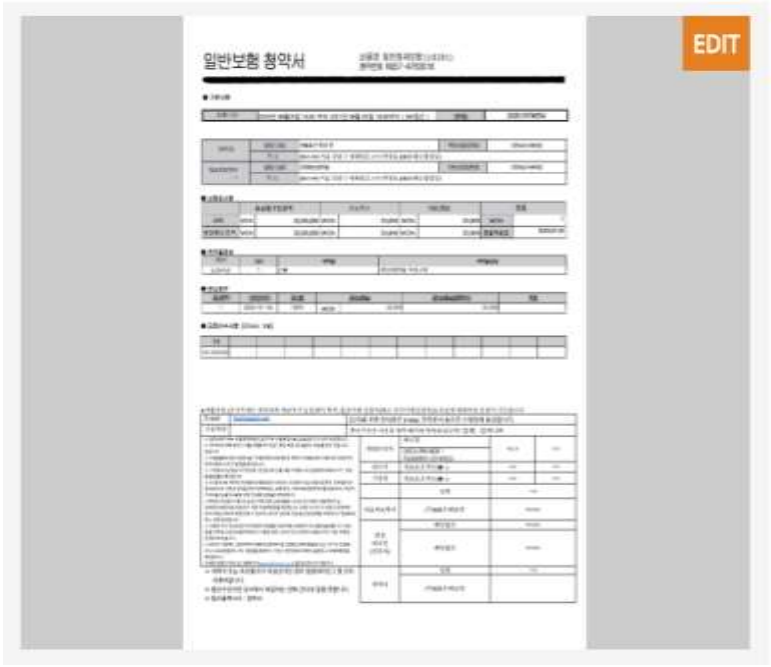
+ 코드 + 텍스트

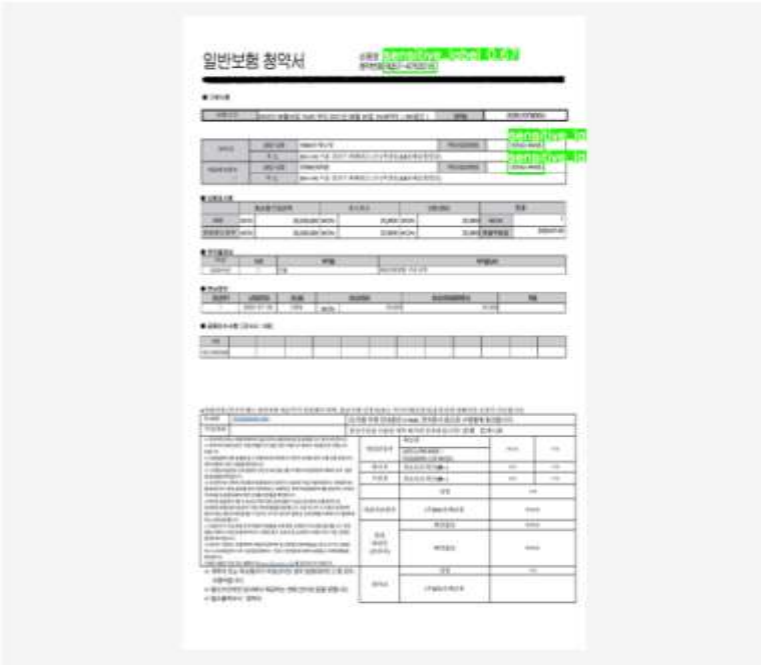
```
170 image = gr.inputs.Image()
171 label = gr.outputs.Image(label = 'Sensitive Label Detection', type='file')
172 gr.Interface(fn=detect, inputs=image, outputs=label, capture_session=True).launch(debug=True)
```

Connect so that results can be seen in the Gradio.app interface

ID card detection

›› Check the training result in Gradio.app





Uploading a photo to be tested will result in the detection of an Id card on the right

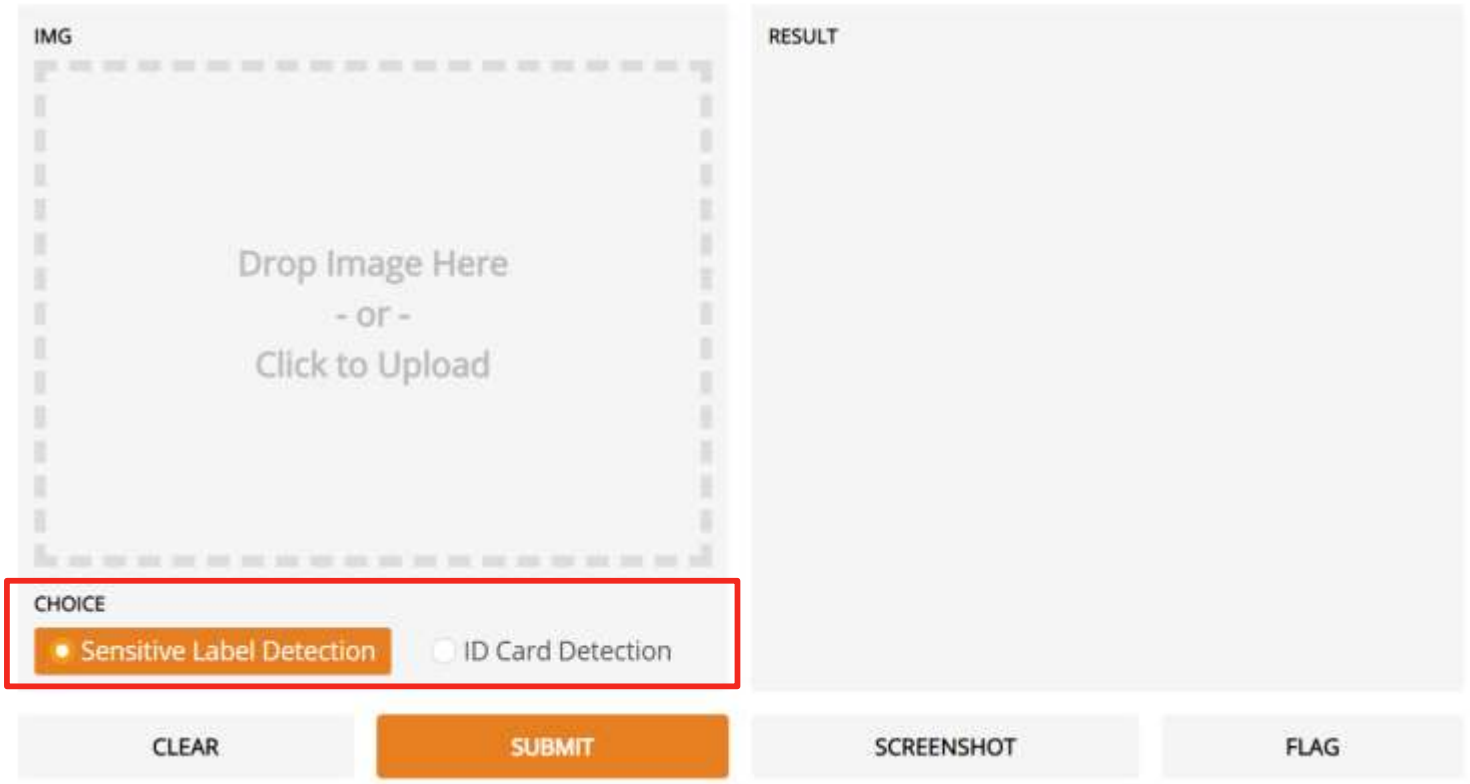
Object detected well BUT Accuracy is 69%, not good

EXPERIMENT

– Concatenate models

Use two models in Gradio

›› Desired output



Want to create an interface that allows users to **freely choose** which model to use using radio buttons

Use two models in Gradio

›› Functionalization of ID card detect model

In [61]:
```
def ID_card_detect(source):
    inp = Image.fromarray(source.astype('uint8'), 'RGB')
    filename = "test.jpeg"
    inp.save(filename, "JPEG")

    cfg = '/content/grive/My Drive/Colab Notebooks/ID_card_detection/yolov3/cfg/yolov3-tiny.cfg'
    names = '/content/grive/My Drive/Colab Notebooks/ID_card_detection/yolov3/data/coco_id.names'
    weights = '/content/grive/My Drive/Colab Notebooks/ID_card_detection/yolov3/weights/ID_Card_Detection.pt'
    out = 'output'
    img_size=512
    conf_thres=0.3
    iou_thres=0.6
    fourcc='mp4v'
    half=False
    device=''
    view_img=False
    save_txt = False
    classes=None
    agnostic_nms=False
    augment=False
    cfg = check_file(cfg)
    names = check_file(names)
    save_img = True
```

Use two models in Gradio

›› Functionalization of sensitive label detect model

In [62]:
```
def sensitive_label_detect(source):
    inp = Image.fromarray(source.astype('uint8'), 'RGB')
    filename = "test.jpeg"
    inp.save(filename, "JPEG")

    cfg = '/content/grive/My Drive/Colab Notebooks/sensitive_label_detection/yolov3/cfg/yolov3-tiny.cfg'
    names = '/content/grive/My Drive/Colab Notebooks/sensitive_label_detection/yolov3/data/coco_id.names'
    weights = '/content/grive/My Drive/Colab Notebooks/sensitive_label_detection/yolov3/weights/last.pt'
    out = 'output'
    img_size=512
    conf_thres=0.3
    iou_thres=0.6
    fourcc='mp4v'
    half=False
    device=''
    view_img=False
    save_txt = False
    classes=None
    agnostic_nms=False
    augment=False
    cfg = check_file(cfg)
    names = check_file(names)
    save_img = True


    imgsz = (320, 192) if ONNX_EXPORT else img_size

    device = torch_utils.select_device(device='cpu' if ONNX_EXPORT else device)
    if os.path.exists(out):
        shutil.rmtree(out)
    os.makedirs(out)
```

Use two models in Gradio

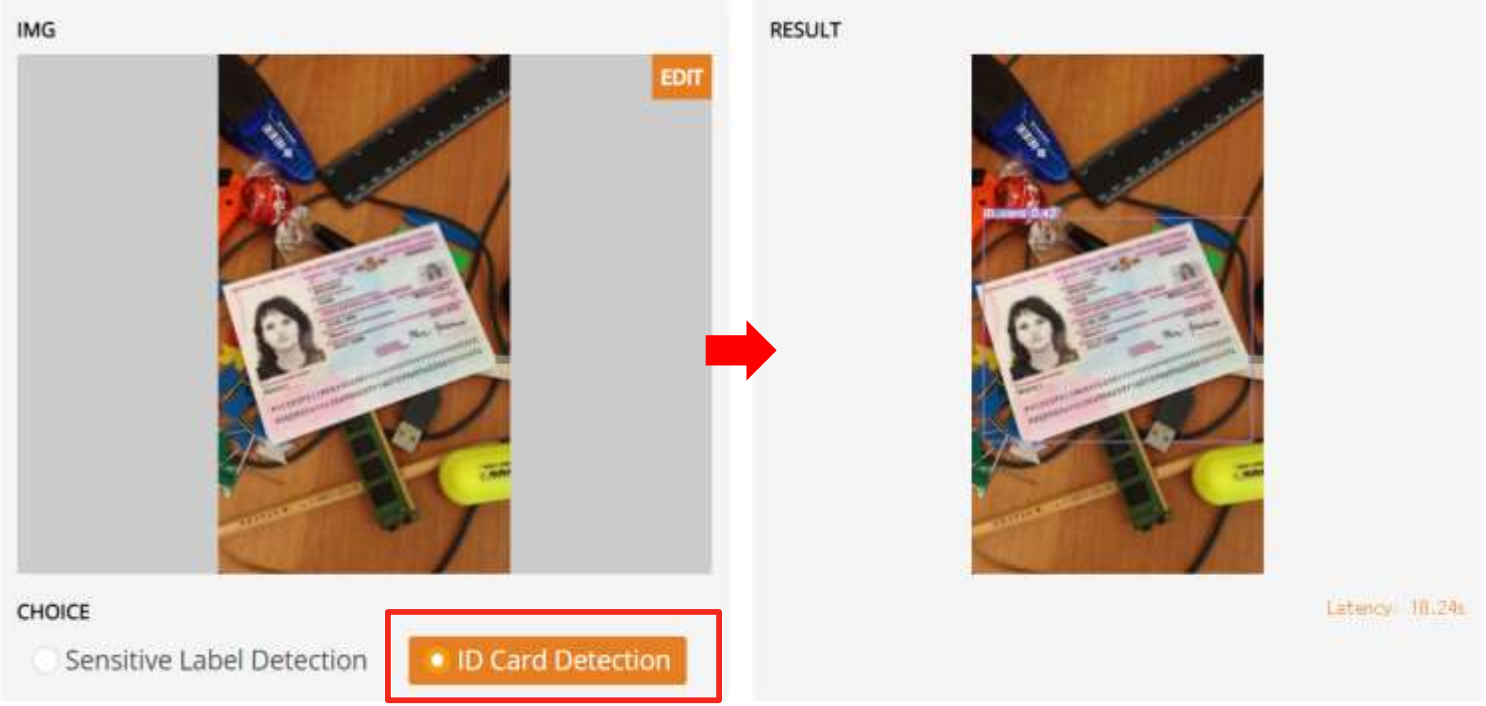
›› Concatenate two models

```
def FN(img,choice):
    if choice ==  "Sensitive Label Detection":
        #!cd /content/grive/My Drive/Colab Notebooks/sensitive_label_detection/yolov3
        return sensitive_label_detect(img)
    elif choice ==  "ID Card Detection":
        !cd /content/grive/My Drive/Colab Notebooks/ID_card_detection/yolov3
        import models
        import utils
        return ID_card_detect(img)
```

Allows the model used to change depending on the radio button selected by the user

Use two models in Gradio

›› Completed Interface

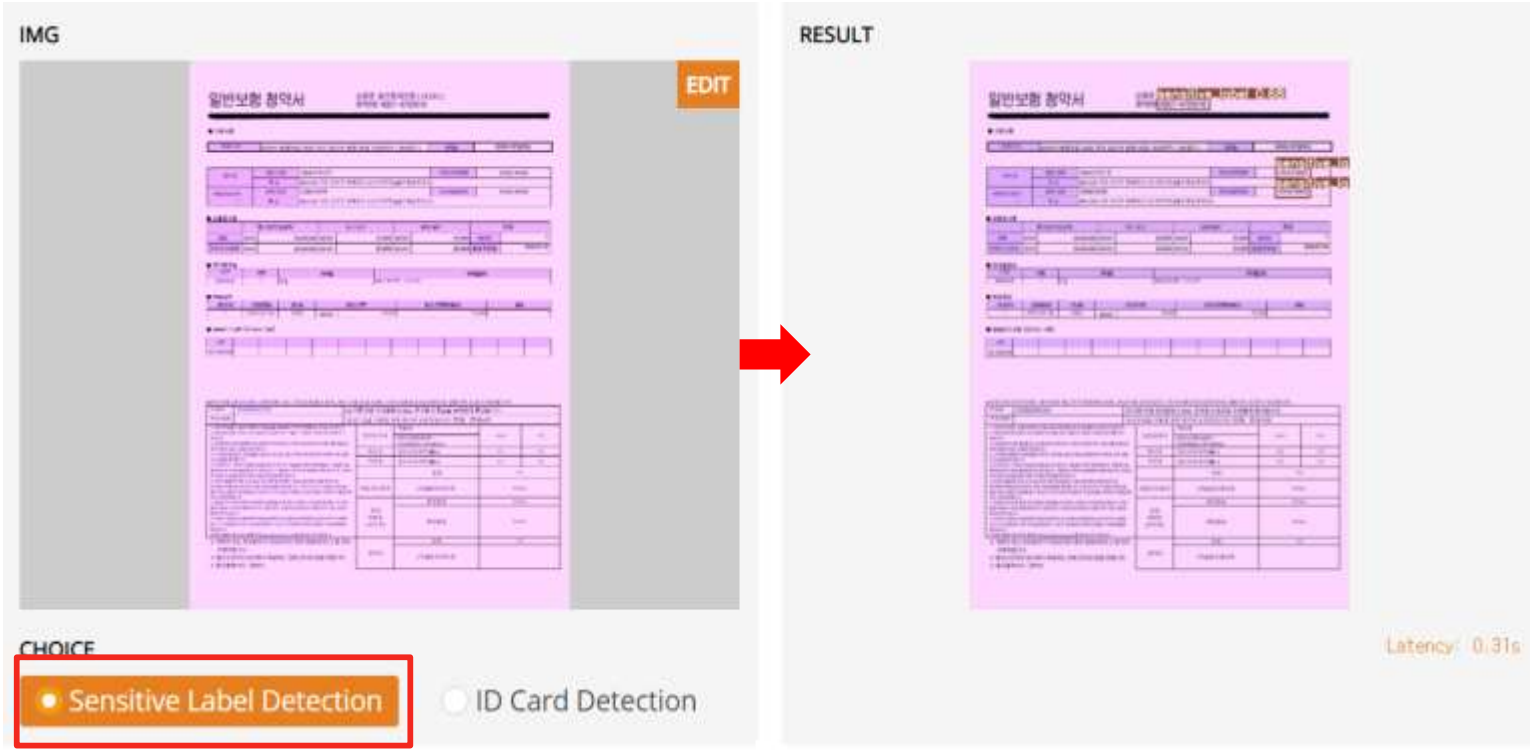


When the user **chose the ID Card Detection model,**

The ID card was detected successfully

Use two models in Gradio

›› Completed Interface



When the user **chose the Sensitive Label Detection model,**

The sensitive labels were detected successfully

CONCLUSION

Conclusion

During this project, I could learn a lot from making data to training models.

Also, I was proud to have made simple results with image processing that I had studied for six weeks.

The accuracy of detecting objects is not so good yet.

So, based on my experience this time, I am going to make my own project by proceeding with data processing more thoroughly.

CONCLUSION

Conclusion

During this project, I could learn a lot from making data to training models.

Also, I was proud to have made simple results with image processing that I had studied for six weeks.

The accuracy of detecting objects is not so good yet.

So, based on my experience this time, I am going to make my own project by proceeding with data processing more thoroughly.

Thank you