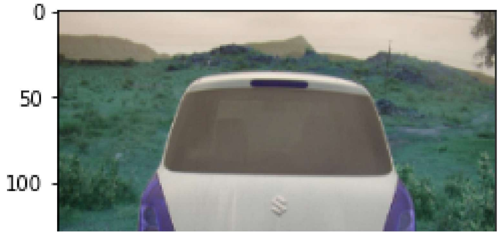
In [1]:

```
import numpy as np
import pandas as pd
import cv2

from bs4 import BeautifulSoup
boxes = {}
coches = {}
import os
# 필요한 라이브러리 임포트
for dirname, _, filenames in os.walk('C:₩₩Users₩₩rlarb₩₩Downloads₩₩archive'):
    for filename in filenames:
        if filename[-3:] == "xml":
            xml = open(os.path.join(dirname, filename), "r")
            contents = xml.read()
            soup = BeautifulSoup(contents,'xml')
            xmin = soup.find('xmin').text
            ymin = soup.find('ymin').text
            xmax = soup.find('xmax').text
            ymax = soup.find('ymax').text
            boxes[filename.split('/')[-1][0:-4]]  = (xmin,ymin,xmax,ymax)
        else:
            img_org = cv2.imread(os.path.join(dirname, filename))
            #print(img_org.shape)
            coches[filename.split('/')[-1][0:-4]] = img_org
#xml 정해져있는 모범답안 - 뷰티풀스프로 미리 태그되어있는 좌표값을 가져옴

# 데이터셋 임포트하는 과정
```

In [2]:

```
X = []
y = []
import matplotlib.pyplot as plt
for car in coches:
    data_coche = coches[car]
    data_coche_resized = cv2.resize(data_coche, (256,256))
    y_ = data_coche.shape[0]
    x_ = data_coche.shape[1]
    x_scale = (256/x_)
    y_scale = (256/y_)
    box_car = boxes[car]
    xmin = (int(box_car[0]))*x_scale
    ymin = (int(box_car[1]))*y_scale
    xmax= (int(box_car[2]))*x_scale
    ymax= (int(box_car[3]))*y_scale
    img = cv2.resize(data_coche, (256,256))

    plt.imshow(img)
    plt.plot([xmin,xmax], [ymax,ymax], c = "red",linewidth=7.0)
    plt.plot([xmin,xmax], [ymin,ymin], c = "red",linewidth=7.0)
    plt.plot([xmin,xmin],[ymax,ymin], c = "red",linewidth=7.0)
    plt.plot([xmax,xmax],[ymax,ymin], c = "red",linewidth=7.0)
    plt.show()

    X.append(img)
    y.append((xmin,ymin,xmax,ymax))

    ## 각각의 사진의 크기들이 다르기 때문에 학습에 용이하게 맞춰주는 과정
    ## 미리 입력해놓은 정확한 번호판 위치
```



In [3]:

```
X = np.array(X).reshape(-1, 256, 256, 3) / 255
```

In [4]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size = 0.1)

import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Flatten
from keras.applications.vgg16 import VGG16

## 학습시키는데 필요한 라이브러리를 임포트하는 과정
```

In [5]:

```
model = Sequential()
model.add(VGG16(weights="imagenet", include_top=False, input_shape=(256,256, 3)))
model.add(Flatten())
model.add(Dense(128, activation="relu"))
model.add(Dense(128, activation="relu"))
model.add(Dense(64, activation="relu"))
model.add(Dense(4, activation="linear"))

model.layers[-6].trainable = False

model.summary()

## relu 경사함수, linear 선형회귀를 통해 예측모델을 만듦.
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 8, 8, 512)         14714688

 flatten (Flatten)           (None, 32768)             0

 dense (Dense)               (None, 128)               4194432

 dense_1 (Dense)             (None, 128)               16512

 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 4)                 260

=================================================================
Total params: 18,934,148
Trainable params: 4,219,460
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [6]:

```
y_train_new = []
for x,y,z,t in y_train:
    y_train_new.append((float(x),float(y),float(z),float(t)))
y_train_new = np.array(y_train_new)

y_val_new = []
for x,y,z,t in y_val:
    y_val_new.append((float(x),float(y),float(z),float(t)))
y_val_new = np.array(y_val_new)
y_train_new
```

Out[6]:

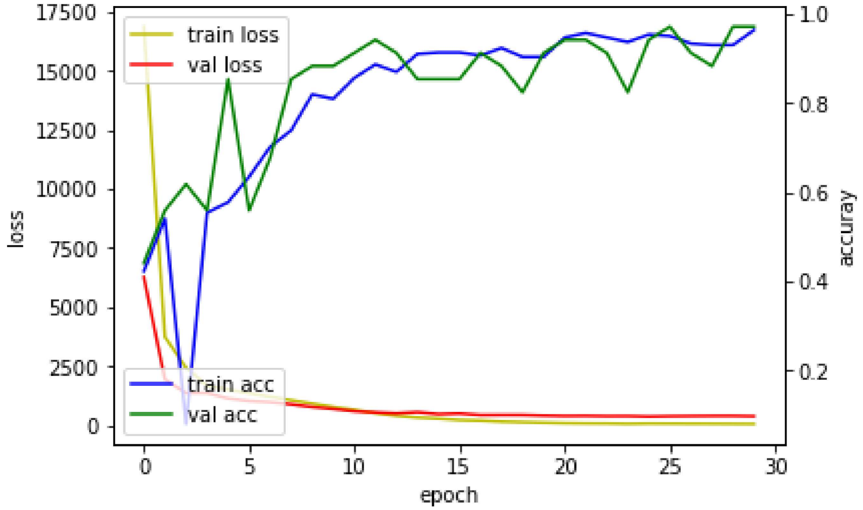
```
array([[ 85.76      , 132.12903226, 167.68      , 165.16129032],
       [106.24      , 129.37634409, 131.84      , 141.3046595 ],
       [ 91.52      , 146.40802676, 134.4       , 178.08695652],
       ...,
       [ 74.88      , 150.53183521, 172.8       , 183.13108614],
       [ 35.2       , 109.33333333, 219.52      , 196.        ],
       [215.04      , 163.84      , 228.48      , 174.08      ]])
```

```
In [7]: model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                  loss='mse',
                  #MSE는 회귀(regression) 용도의 딥러닝 모델을 훈련시킬때 많이 사용되는 손실 함수입니다.
                  metrics=["accuracy"])

        initial_epochs = 30

        #early_stop = keras.callbacks.EarlyStopping(patience=2, restore_best_weights=True)
        #과적합 되지않게 early_stop을 정해놨다.

        history = model.fit(X_train, y_train_new,
                  validation_data = (X_val, y_val_new),
                  epochs=initial_epochs,
                  batch_size= 32,)
                 #callbacks=[early_stop])
```

```
10/10 [==============================] - 58s 6s/step - loss: 75.0719 - accuracy: 0.9362 - val_loss: 401.7256 - val_a
ccuracy: 0.8235
Epoch 25/30
10/10 [==============================] - 49s 5s/step - loss: 86.0243 - accuracy: 0.9530 - val_loss: 382.3334 - val_a
ccuracy: 0.9412
Epoch 26/30
10/10 [==============================] - 46s 5s/step - loss: 82.8152 - accuracy: 0.9497 - val_loss: 396.4296 - val_a
ccuracy: 0.9706
Epoch 27/30
10/10 [==============================] - 44s 4s/step - loss: 78.9335 - accuracy: 0.9329 - val_loss: 403.1828 - val_a
ccuracy: 0.9118
Epoch 28/30
10/10 [==============================] - 45s 5s/step - loss: 76.6586 - accuracy: 0.9295 - val_loss: 408.3762 - val_a
ccuracy: 0.8824
Epoch 29/30
10/10 [==============================] - 43s 4s/step - loss: 72.6668 - accuracy: 0.9295 - val_loss: 408.9618 - val_a
ccuracy: 0.9706
Epoch 30/30
10/10 [==============================] - 43s 4s/step - loss: 68.2704 - accuracy: 0.9631 - val_loss: 395.9052 - val_a
ccuracy: 0.9706
```

In [8]:

```
import matplotlib.pyplot as plt

fig, loss_ax = plt.subplots()

acc_ax = loss_ax.twinx()

loss_ax.plot(history.history['loss'], 'y', label='train loss')
loss_ax.plot(history.history['val_loss'], 'r', label='val loss')

acc_ax.plot(history.history['accuracy'], 'b', label='train acc')
acc_ax.plot(history.history['val_accuracy'], 'g', label='val acc')

loss_ax.set_xlabel('epoch')
loss_ax.set_ylabel('loss')
acc_ax.set_ylabel('accuray')

loss_ax.legend(loc='upper left')
acc_ax.legend(loc='lower left')

plt.show()

# train loss 훈련 손실값
## 손실도와 정확도를 나타내는 것
```



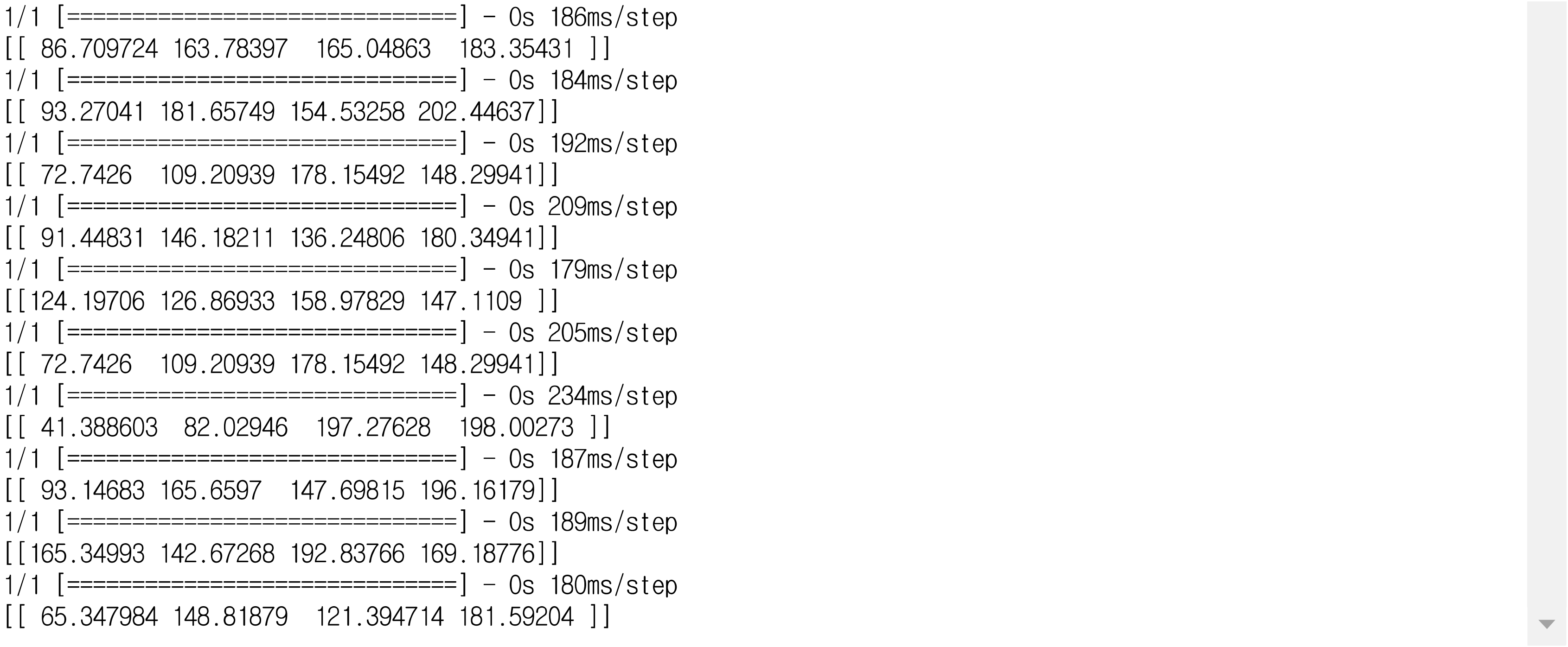
In [9]:

```
y_test_new = []
for x,y,z,t in y_test:
    y_test_new.append((float(x),float(y),float(z),float(t)))
y_test_new = np.array(y_test_new)
```

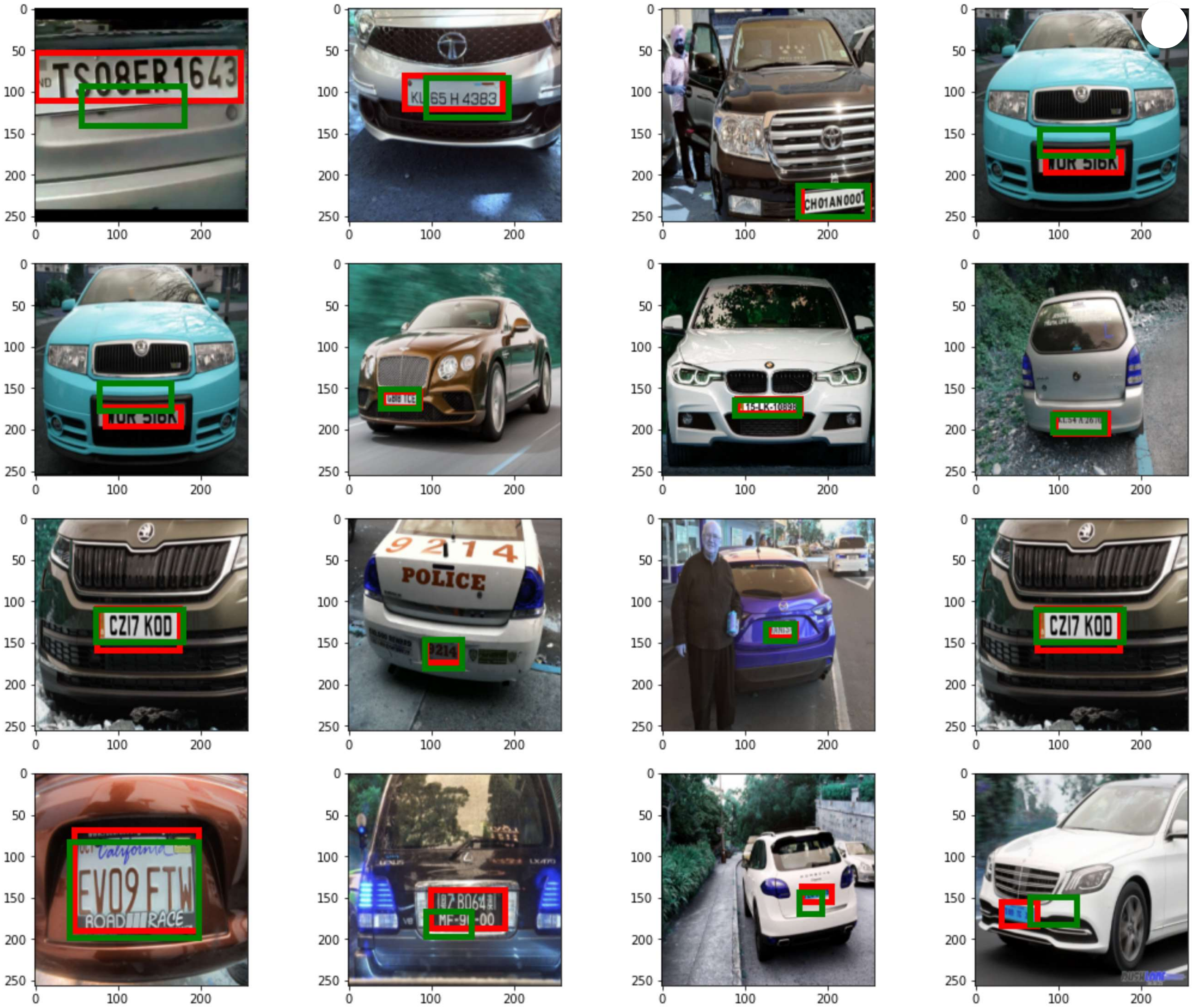
In [10]:

```
import random
def ejemplo_preds(X,y):
    plt.figure(figsize=(18,15))
    cont = 1
    indices = [random.randint(0,len(X)-1) for i in range(16)]
    for ind in indices:
        pred = model.predict(X[ind].reshape(-1,256,256,3))
        print(pred)
        plt.subplot(4,4,cont)
        plt.imshow(X[ind])
        plt.plot([y[ind][0],y[ind][2]], [y[ind][3],y[ind][3]], c = "red",linewidth=5.0)
        plt.plot([y[ind][0],y[ind][2]], [y[ind][1],y[ind][1]], c = "red",linewidth=5.0)
        plt.plot([y[ind][0],y[ind][0]],[y[ind][3],y[ind][1]], c = "red",linewidth=5.0)
        plt.plot([y[ind][2],y[ind][2]],[y[ind][3],y[ind][1]], c = "red",linewidth=5.0)


        plt.plot([pred[0][0],pred[0][2]], [pred[0][3],pred[0][3]], c = "green",linewidth=5.0)
        plt.plot([pred[0][0],pred[0][2]], [pred[0][1],pred[0][1]], c = "green",linewidth=5.0)
        plt.plot([pred[0][0],pred[0][0]],[pred[0][3],pred[0][1]], c = "green",linewidth=5.0)
        plt.plot([pred[0][2],pred[0][2]],[pred[0][3],pred[0][1]], c = "green",linewidth=5.0)
        cont += 1
    plt.show()



ejemplo_preds(X_test,y_test_new)

# 상대적으로 정확도가 떨어지는 10번의 학습
```

```
1/1 [==============================] - 1s 669ms/step
[[ 55.230965  92.96156  178.96414  140.25958 ]]
1/1 [==============================] - 0s 201ms/step
[[ 92.31488  83.0574  192.81042 130.71863]]
1/1 [==============================] - 0s 195ms/step
[[164.07127 212.51071 247.93842 250.70279]]
1/1 [==============================] - 0s 219ms/step
[[ 77.00528 144.3144  164.34203 176.67107]]
1/1 [==============================] - 0s 192ms/step
[[ 77.00528 144.3144  164.34203 176.67107]]
1/1 [==============================] - 0s 203ms/step
[[ 38.220978 150.55148   83.68123  173.5743  ]]
```



```
1/1 [==============================] - 0s 186ms/step
[[ 86.709724 163.78397  165.04863  183.35431 ]]
1/1 [==============================] - 0s 184ms/step
[[ 93.27041 181.65749 154.53258 202.44637]]
1/1 [==============================] - 0s 192ms/step
[[ 72.7426  109.20939 178.15492 148.29941]]
1/1 [==============================] - 0s 209ms/step
[[ 91.44831 146.18211 136.24806 180.34941]]
1/1 [==============================] - 0s 179ms/step
[[124.19706 126.86933 158.97829 147.1109 ]]
1/1 [==============================] - 0s 205ms/step
[[ 72.7426  109.20939 178.15492 148.29941]]
1/1 [==============================] - 0s 234ms/step
[[ 41.388603  82.02946  197.27628  198.00273 ]]
1/1 [==============================] - 0s 187ms/step
[[ 93.14683 165.6597  147.69815 196.16179]]
1/1 [==============================] - 0s 189ms/step
[[165.34993 142.67268 192.83766 169.18776]]
1/1 [==============================] - 0s 180ms/step
[[ 65.347984 148.81879  121.394714 181.59204 ]]
```


0
50
100
150
200
250
0
100
200
TS08ER1643
KL 65 H 4383
CH01AN0001
WUR 516K
CZ17 KOD
9214
POLICE
9214
California
EV09 FTW
ROAD RACE
7 B064
MF-91-00

In [ ]:

In [ ]: