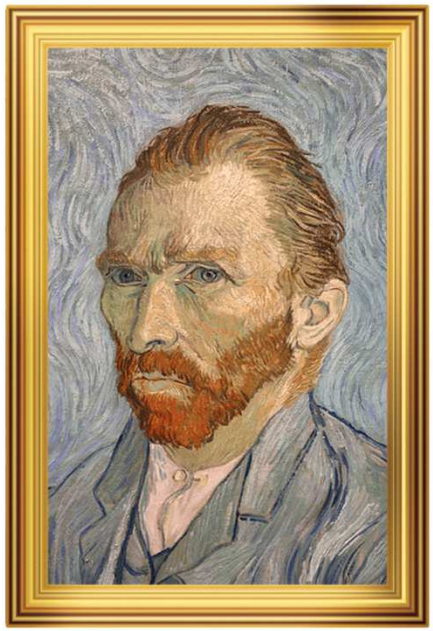


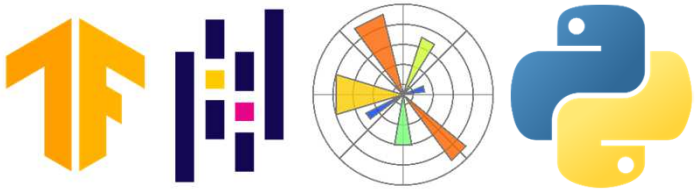
semi-project

崔文朴三金

Art Gallery Design

Our Project

딥러닝 모델 구현 사용 스택

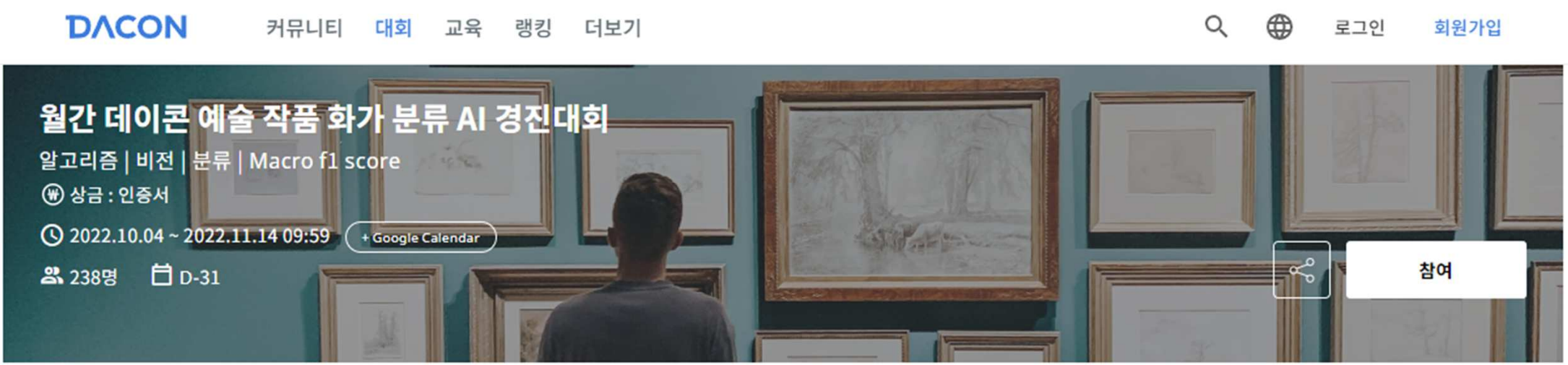


프론트&백엔드 사용 스택

Deep Learning Model

Web development

崔文朴三金



Dataset & Goal

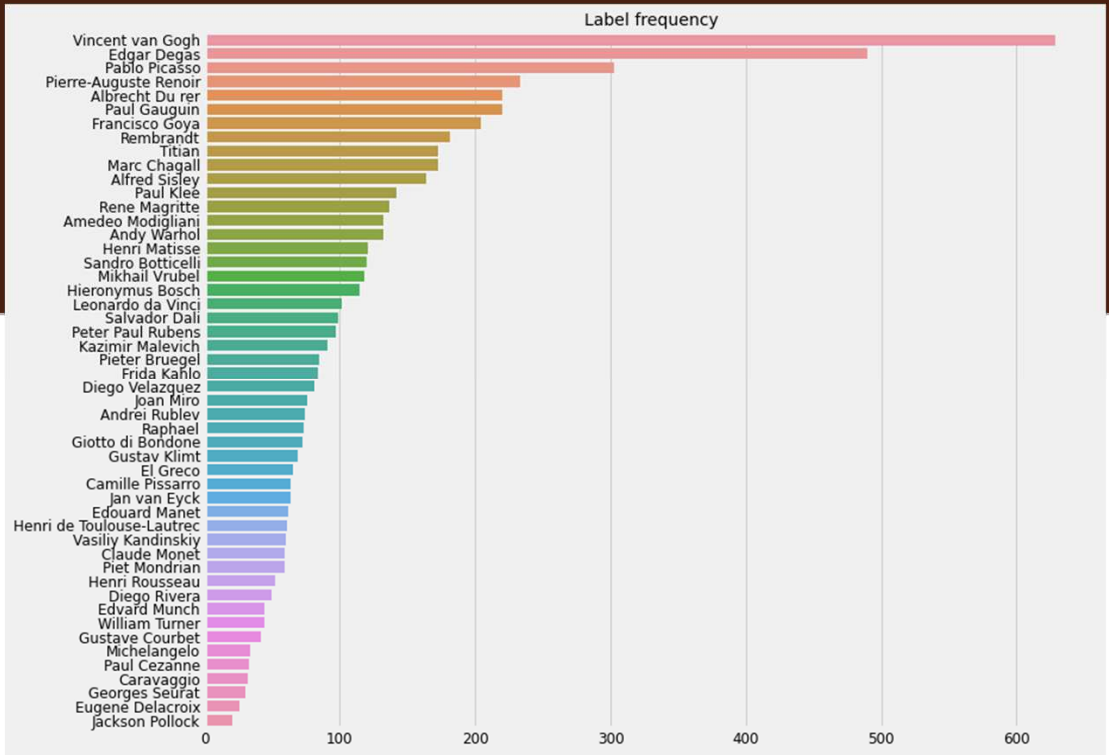
Deep Learnng Model

崔文朴三金

1.
Module import

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import json
import os
from tqdm import tqdm, tqdm_notebook
import random

import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import *
from tensorflow.keras.optimizers import *
from tensorflow.keras.applications import *
from tensorflow.keras.callbacks import *
from tensorflow.keras.initializers import *
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from numpy.random import seed
seed(1)
tf.random.set_seed(1)
```

2. Raw Data Labels



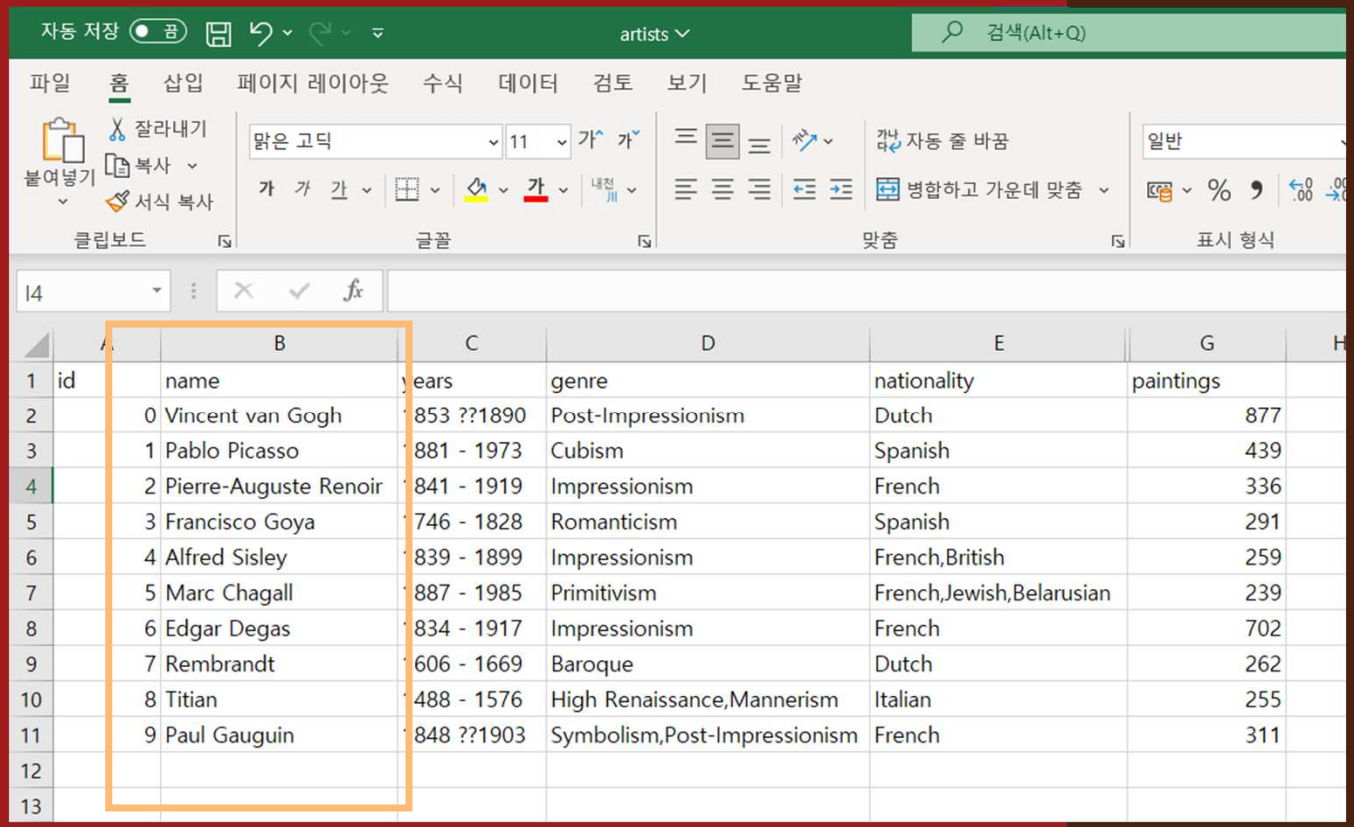
3. Calculate Class Weights

```
artists = artists.sort_values(by=['paintings'], ascending=False)

artists_top = artists[artists['paintings'] >= 200].reset_index()
artists_top = artists_top[['name', 'paintings']]

artists_top['class_weight'] = artists_top.paintings.sum() / (artists_top.shape[0] * artists_top.paintings)
artists_top
```

	name	paintings	class_weight
0	Vincent van Gogh	877	0.452794
1	Edgar Degas	702	0.565670
2	Pablo Picasso	439	0.904556
3	Pierre-Auguste Renoir	336	1.181845
4	Paul Gauguin	311	1.276849
5	Francisco Goya	291	1.364605
6	Rembrandt	262	1.515649
7	Alfred Sisley	259	1.533205
8	Titian	255	1.557255
9	Marc Chagall	239	1.661506

```
[ ]  class_weights = artists_top['class_weight'].to_dict()
     class_weights
```

```
{0: 0.45279361459521095,
 1: 0.5656695156695156,
 2: 0.9045558086560365,
 3: 1.181845238095238,
 4: 1.2768488745980708,
 5: 1.3646048109965636,
 6: 1.515648854961832,
 7: 1.5332046332046332,
 8: 1.5572549019607844,
 9: 1.6615062761506276}
```

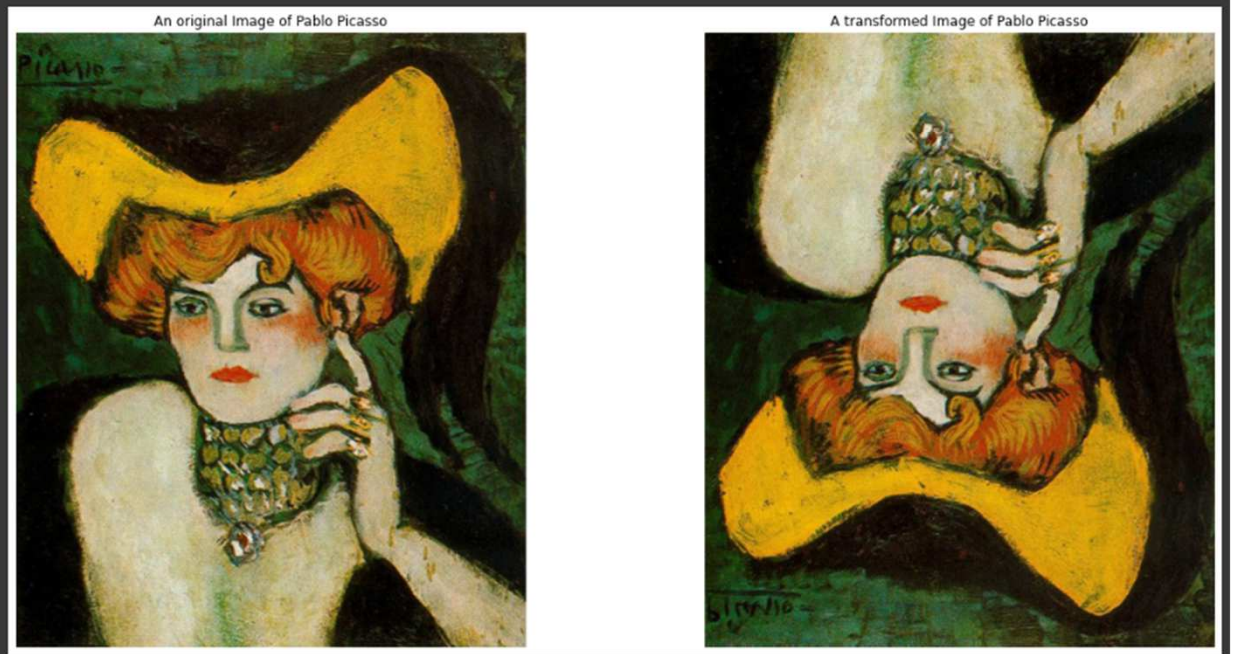


	A	B	C	D	E	G	H
1	id	name	years	genre	nationality	paintings	
2	0	Vincent van Gogh	1853 ??1890	Post-Impressionism	Dutch	877	
3	1	Pablo Picasso	1881 - 1973	Cubism	Spanish	439	
4	2	Pierre-Auguste Renoir	1841 - 1919	Impressionism	French	336	
5	3	Francisco Goya	1746 - 1828	Romanticism	Spanish	291	
6	4	Alfred Sisley	1839 - 1899	Impressionism	French,British	259	
7	5	Marc Chagall	1887 - 1985	Primitivism	French,Jewish,Belarusian	239	
8	6	Edgar Degas	1834 - 1917	Impressionism	French	702	
9	7	Rembrandt	1606 - 1669	Baroque	Dutch	262	
10	8	Titian	1488 - 1576	High Renaissance,Mannerism	Italian	255	
11	9	Paul Gauguin	1848 ??1903	Symbolism,Post-Impressionism	French	311	
12							
13							

4.
X - lable

5. Data Transform

```
fig, axes = plt.subplots(1, 2, figsize=(20,10))

random_artist = random.choice(artists_top_name)
random_image = random.choice(os.listdir(os.path.join(images_dir, random_artist)))
random_image_file = os.path.join(images_dir, random_artist, random_image)

image = plt.imread(random_image_file)
axes[0].imshow(image)
axes[0].set_title("An original Image of " + random_artist.replace('_', ' '))
axes[0].axis('off')

aug_image = train_datagen.random_transform(image)
axes[1].imshow(aug_image)
axes[1].set_title("A transformed Image of " + random_artist.replace('_', ' '))
axes[1].axis('off')

plt.show()
```



6. ResNet50

```
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=train_input_shape)

for layer in base_model.layers:
    layer.trainable = True
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet5
94773248/94765736 [==============================] - 0s 0us/step
94781440/94765736 [==============================] - 0s 0us/step
```

```
[ ]
    X = base_model.output
    X = Flatten()(X)

    X = Dense(512, kernel_initializer='he_uniform')(X)
    #X = Dropout(0.5)(X)
    X = BatchNormalization()(X)
    X = Activation('relu')(X)

    X = Dense(16, kernel_initializer='he_uniform')(X)
    #X = Dropout(0.5)(X)
    X = BatchNormalization()(X)
    X = Activation('relu')(X)

    output = Dense(n_classes, activation='softmax')(X)

    model = Model(inputs=base_model.input, outputs=output)
```

첫번째 학습

epoch= 10

```
[ ]  optimizer = Adam(lr=0.0001)
     model.compile(loss='categorical_crossentropy',
                   optimizer=optimizer,
                   metrics=['accuracy'])
```

```
     /usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarning: The `lr` argumer
       super(Adam, self).__init__(name, **kwargs)
```

```
[ ]  n_epoch = 10

     early_stop = EarlyStopping(monitor='val_loss', patience=20, verbose=1,
                                mode='auto', restore_best_weights=True)

     reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5,
                                   verbose=1, mode='auto')
```

```
     history1 = model.fit_generator(generator=train_generator, steps_per_epoch=STEP_SIZE_TRAIN,
                                    validation_data=valid_generator, validation_steps=STEP_SIZE_VALID,
                                    epochs=n_epoch,
                                    shuffle=True,
                                    verbose=1,
                                    callbacks=[reduce_lr],
                                    use_multiprocessing=True,
                                    workers=16,
                                    class_weight=class_weights
                                   )
```

崔文朴三金

학습결과 부진

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version.
  if __name__ == '__main__':
Epoch 1/10
198/198 [==============================] - 130s 546ms/step - loss: 1.5420 - accuracy: 0.5039 - val_loss: 5.0807 - val_accuracy: 0.0804 - lr: 1.0000e-04
Epoch 2/10
198/198 [==============================] - 23s 107ms/step - loss: 1.1671 - accuracy: 0.7036 - val_loss: 3.1224 - val_accuracy: 0.0599 - lr: 1.0000e-04
Epoch 3/10
198/198 [==============================] - 23s 106ms/step - loss: 1.0153 - accuracy: 0.7640 - val_loss: 2.6333 - val_accuracy: 0.0842 - lr: 1.0000e-04
Epoch 4/10
198/198 [==============================] - 22s 104ms/step - loss: 0.8970 - accuracy: 0.8123 - val_loss: 2.3421 - val_accuracy: 0.2117 - lr: 1.0000e-04
Epoch 5/10
198/198 [==============================] - 22s 104ms/step - loss: 0.8009 - accuracy: 0.8502 - val_loss: 1.7182 - val_accuracy: 0.4490 - lr: 1.0000e-04
Epoch 6/10
198/198 [==============================] - 22s 104ms/step - loss: 0.7145 - accuracy: 0.8736 - val_loss: 1.1766 - val_accuracy: 0.7283 - lr: 1.0000e-04
Epoch 7/10
198/198 [==============================] - 22s 105ms/step - loss: 0.6197 - accuracy: 0.9011 - val_loss: 0.9248 - val_accuracy: 0.7959 - lr: 1.0000e-04
Epoch 10/10
198/198 [==============================] - 23s 108ms/step - loss: 0.4625 - accuracy: 0.9292 - val_loss: 0.8755 - val_accuracy: 0.7870 - lr: 1.0000e-04
198/198 [==============================] - 22s 104ms/step - loss: 0.5286 - accuracy: 0.9109 - val_loss: 0.8575 - val_accuracy: 0.7895 - lr: 1.0000e-04
Epoch 10/10
198/198 [==============================] - 23s 108ms/step - loss: 0.4625 - accuracy: 0.9292 - val_loss: 0.8755 - val_accuracy: 0.7870 - lr: 1.0000e-04
```

두번째 학습

epoch= 50

```
[ ]
    for layer in model.layers:
        layer.trainable = False

    for layer in model.layers[:50]:
        layer.trainable = True

    optimizer = Adam(lr=0.0001)

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

    n_epoch = 50
    history2 = model.fit_generator(generator=train_generator, steps_per_epoch=STEP_SIZE_TRAIN,
                                   validation_data=valid_generator, validation_steps=STEP_SIZE_VALID,
                                   epochs=n_epoch,
                                   shuffle=True,
                                   verbose=1,
                                   callbacks=[reduce_lr, early_stop],
                                   use_multiprocessing=True,
                                   workers=16,
                                   class_weight=class_weights
                                  )
```

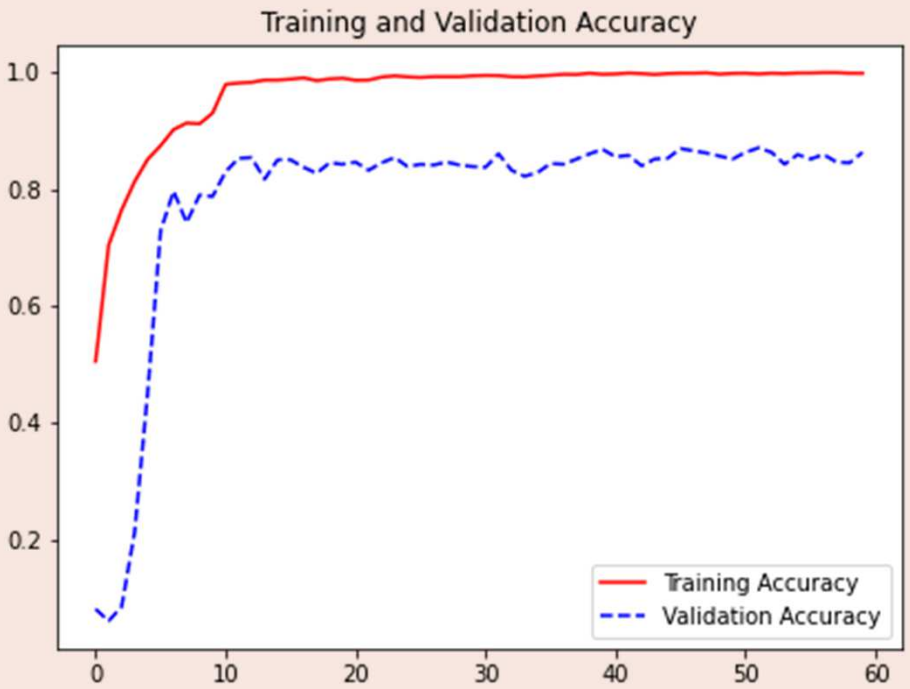
崔文朴三金

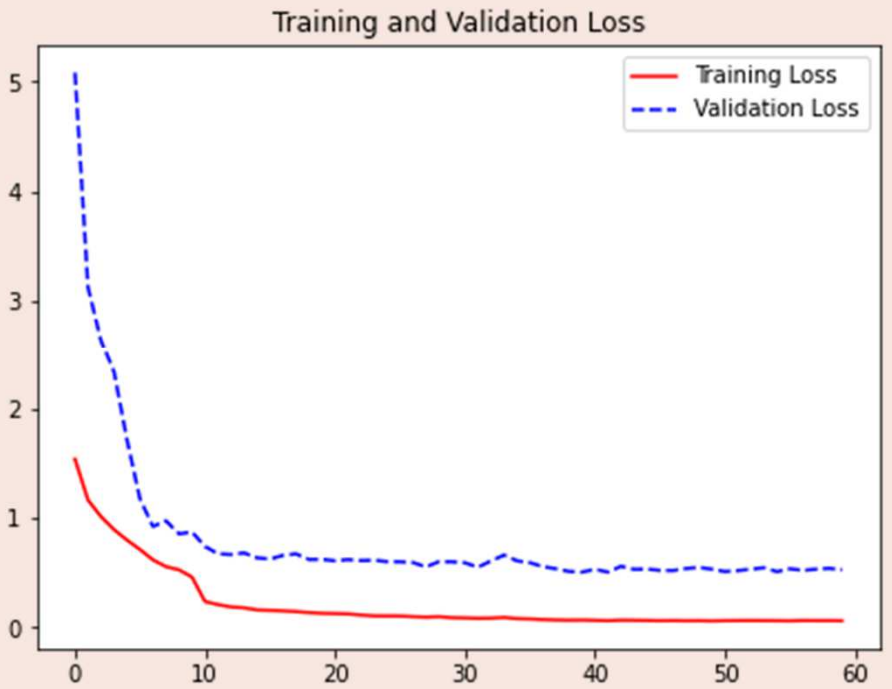
학습결과 **타협**

```
198/198 [==============================] - 23s 108ms/step - loss: 0.0674 - accuracy: 0.9965 - val_loss: 0.5608 - val_accuracy: 0.8393 - lr: 1.0000e-05
Epoch 34/50
198/198 [==============================] - 24s 108ms/step - loss: 0.0661 - accuracy: 0.9949 - val_loss: 0.5335 - val_accuracy: 0.8508 - lr: 1.0000e-05
Epoch 35/50
198/198 [==============================] - 24s 109ms/step - loss: 0.0645 - accuracy: 0.9965 - val_loss: 0.5349 - val_accuracy: 0.8520 - lr: 1.0000e-05
Epoch 36/50
198/198 [==============================] - 23s 107ms/step - loss: 0.0625 - accuracy: 0.9972 - val_loss: 0.5239 - val_accuracy: 0.8686 - lr: 1.0000e-05
Epoch 37/50
198/198 [==============================] - ETA: 0s - loss: 0.0638 - accuracy: 0.9972
Epoch 37: ReduceLROnPlateau reducing learning rate to 9.999999747378752e-07.
198/198 [==============================] - 24s 112ms/step - loss: 0.0638 - accuracy: 0.9972 - val_loss: 0.5219 - val_accuracy: 0.8648 - lr: 1.0000e-05
Epoch 38/50
198/198 [==============================] - 23s 109ms/step - loss: 0.0620 - accuracy: 0.9978 - val_loss: 0.5402 - val_accuracy: 0.8610 - lr: 1.0000e-06
Epoch 39/50
198/198 [==============================] - 23s 107ms/step - loss: 0.0630 - accuracy: 0.9956 - val_loss: 0.5510 - val_accuracy: 0.8559 - lr: 1.0000e-06
Epoch 40/50
198/198 [==============================] - 23s 110ms/step - loss: 0.0605 - accuracy: 0.9968 - val_loss: 0.5344 - val_accuracy: 0.8508 - lr: 1.0000e-06
Epoch 41/50
198/198 [==============================] - 24s 109ms/step - loss: 0.0628 - accuracy: 0.9972 - val_loss: 0.5128 - val_accuracy: 0.8622 - lr: 1.0000e-06
Epoch 42/50
198/198 [==============================] - ETA: 0s - loss: 0.0637 - accuracy: 0.9959
Epoch 42: ReduceLROnPlateau reducing learning rate to 9.999999974752428e-08.
198/198 [==============================] - 24s 109ms/step - loss: 0.0637 - accuracy: 0.9959 - val_loss: 0.5217 - val_accuracy: 0.8699 - lr: 1.0000e-06
Epoch 43/50
198/198 [==============================] - 23s 109ms/step - loss: 0.0642 - accuracy: 0.9972 - val_loss: 0.5339 - val_accuracy: 0.8622 - lr: 1.0000e-07
Epoch 44/50
198/198 [==============================] - 23s 107ms/step - loss: 0.0635 - accuracy: 0.9965 - val_loss: 0.5482 - val_accuracy: 0.8418 - lr: 1.0000e-07
Epoch 45/50
198/198 [==============================] - 23s 107ms/step - loss: 0.0629 - accuracy: 0.9975 - val_loss: 0.5118 - val_accuracy: 0.8584 - lr: 1.0000e-07
Epoch 46/50
198/198 [==============================] - 24s 110ms/step - loss: 0.0613 - accuracy: 0.9975 - val_loss: 0.5382 - val_accuracy: 0.8508 - lr: 1.0000e-07
Epoch 50/50
198/198 [==============================] - 23s 109ms/step - loss: 0.0621 - accuracy: 0.9972 - val_loss: 0.5285 - val_accuracy: 0.8622 - lr: 1.0000e-08
Epoch 48/50
198/198 [==============================] - 24s 112ms/step - loss: 0.0634 - accuracy: 0.9981 - val_loss: 0.5341 - val_accuracy: 0.8457 - lr: 1.0000e-08
Epoch 49/50
198/198 [==============================] - 23s 107ms/step - loss: 0.0635 - accuracy: 0.9972 - val_loss: 0.5424 - val_accuracy: 0.8444 - lr: 1.0000e-08
Epoch 50/50
198/198 [==============================] - 23s 109ms/step - loss: 0.0621 - accuracy: 0.9972 - val_loss: 0.5285 - val_accuracy: 0.8622 - lr: 1.0000e-08
```

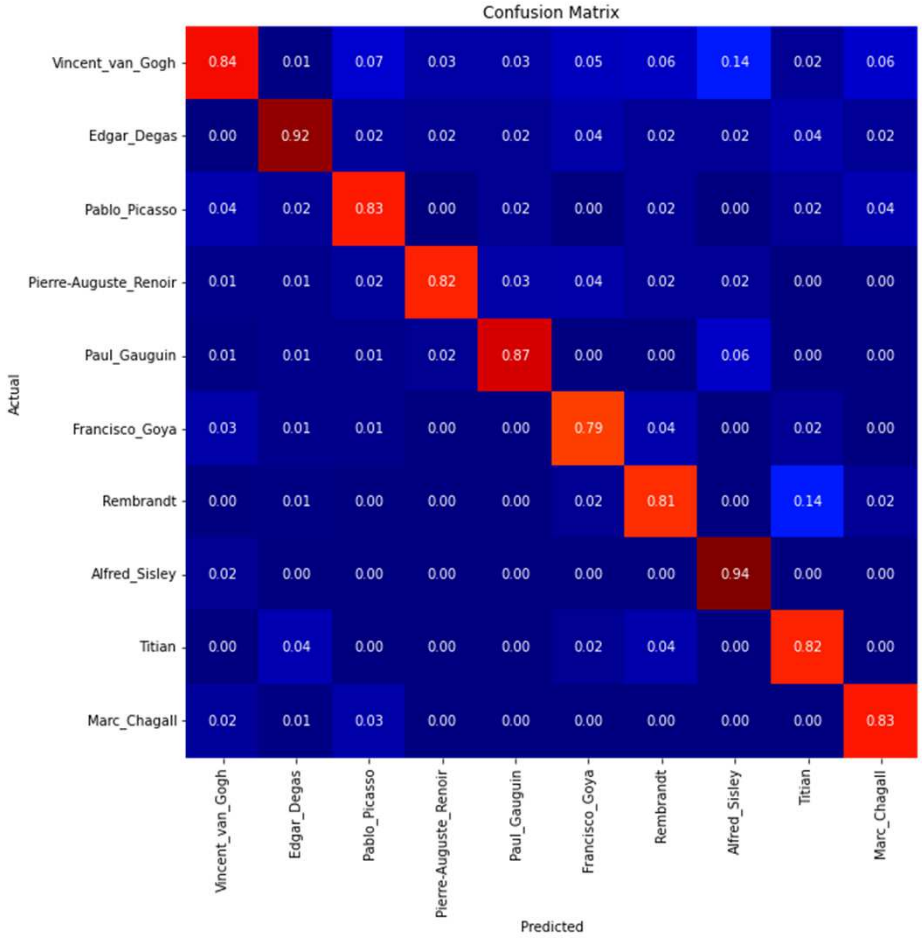
崔文朴三金

ROC


Training and Validation Accuracy
1.0
0.8
0.6
0.4
0.2
0
10
20
30
40
50
60
Training Accuracy
Validation Accuracy


Training and Validation Loss
5
4
3
2
1
0
0
10
20
30
40
50
60
Training Loss
Validation Loss

Comfusion Matrix



崔文朴三金

5 Random Prediction

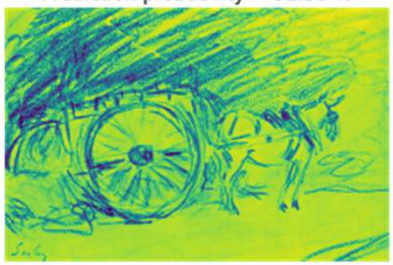
Actual artist = Marc Chagall
Predicted artist = Marc Chagall
Prediction probability = 99.30 %

Actual artist = Pablo Picasso
Predicted artist = Pablo Picasso
Prediction probability = 94.99 %

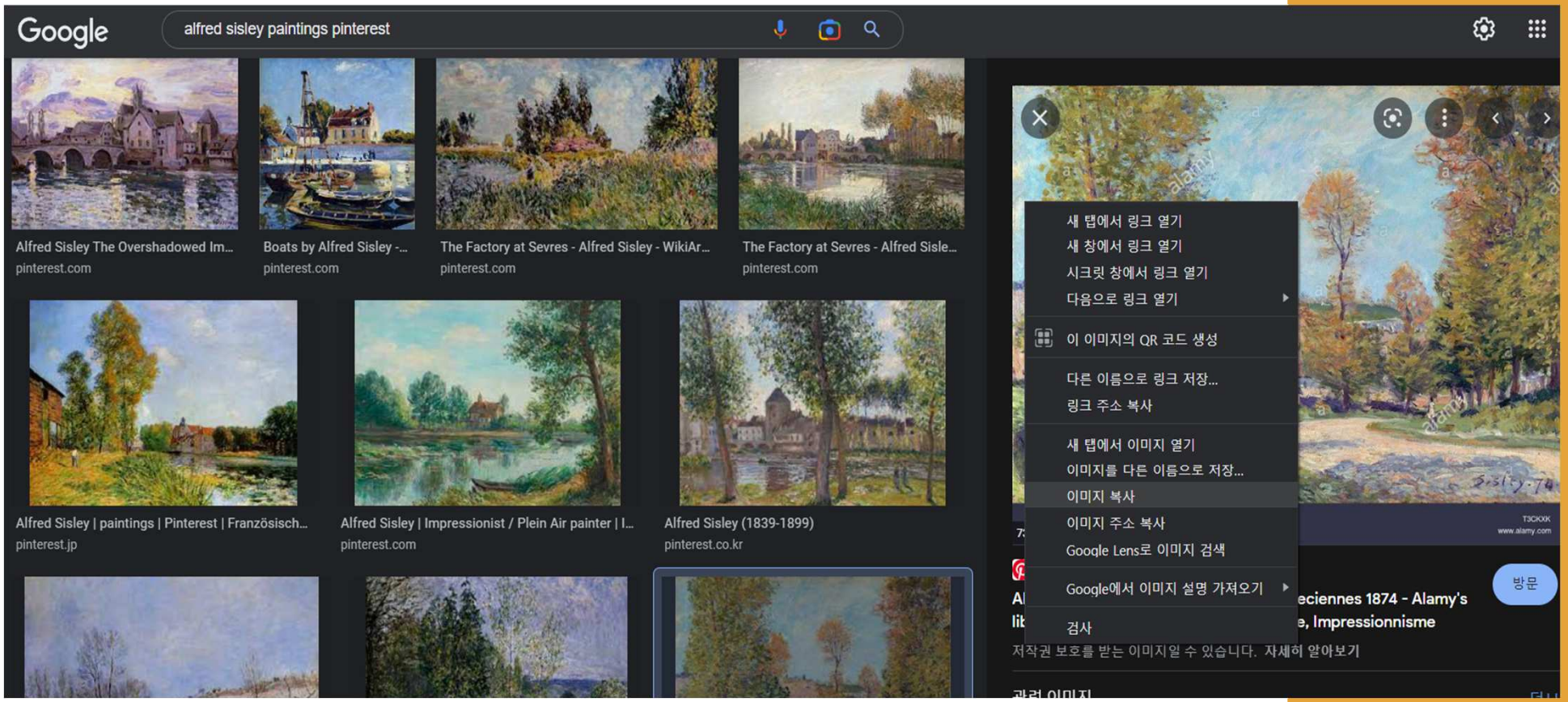
Actual artist = Pablo Picasso
Predicted artist = Pablo Picasso
Prediction probability = 97.76 %

Actual artist = Francisco Goya
Predicted artist = Francisco Goya
Prediction probability = 99.30 %

Actual artist = Alfred Sisley
Predicted artist = Alfred Sisley
Prediction probability = 92.39 %



Select a Random Image

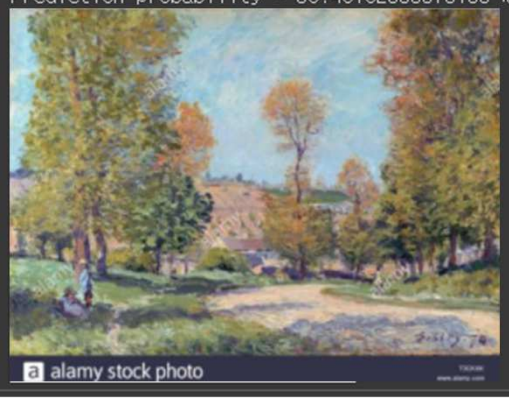


崔文朴三金

Genius of Prediction

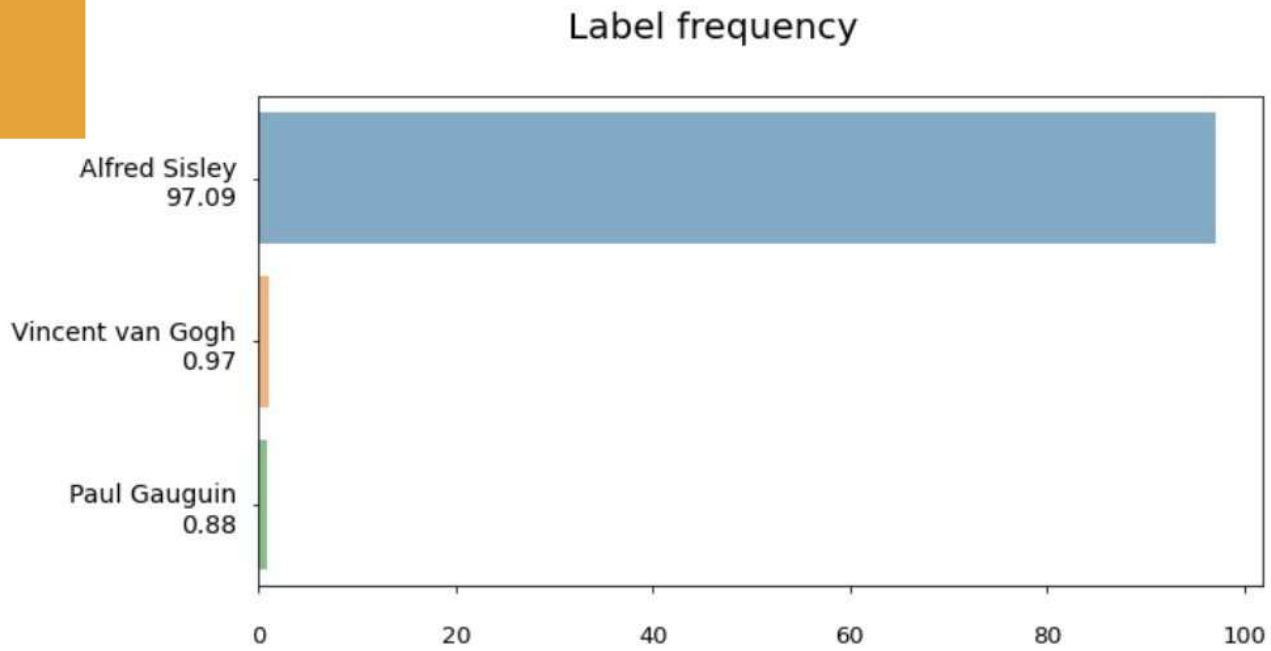
```
[ ]  url = 'https://i.pinimg.com/736x/25/a1/7f/25a17f1edf8c95fd0517c62b9c813dd6--dance-class-the-dance.jpg'

     import imageio
     import cv2

     web_image = imageio.imread(url)
     web_image = cv2.resize(web_image, dsize=train_input_shape[0:2], )
     web_image = image.img_to_array(web_image)
     web_image /= 255.
     web_image = np.expand_dims(web_image, axis=0)


     prediction = model.predict(web_image)
     prediction_probability = np.amax(prediction)
     prediction_idx = np.argmax(prediction)

     print("Predicted artist =", labels[prediction_idx].replace('_', ' '))
     print("Prediction probability =", prediction_probability*100, "%")

     plt.imshow(imageio.imread(url))
     plt.axis('off')
     plt.show()
```

```
Predicted artist = Alfred Sisley
Prediction probability = 90.40102958679199 %
```



崔文朴三金

TOP 3


Label frequency
Alfred Sisley
97.09
Vincent van Gogh
0.97
Paul Gauguin
0.88
0
20
40
60
80
100

Web Development

崔文朴三金

1. STUDY

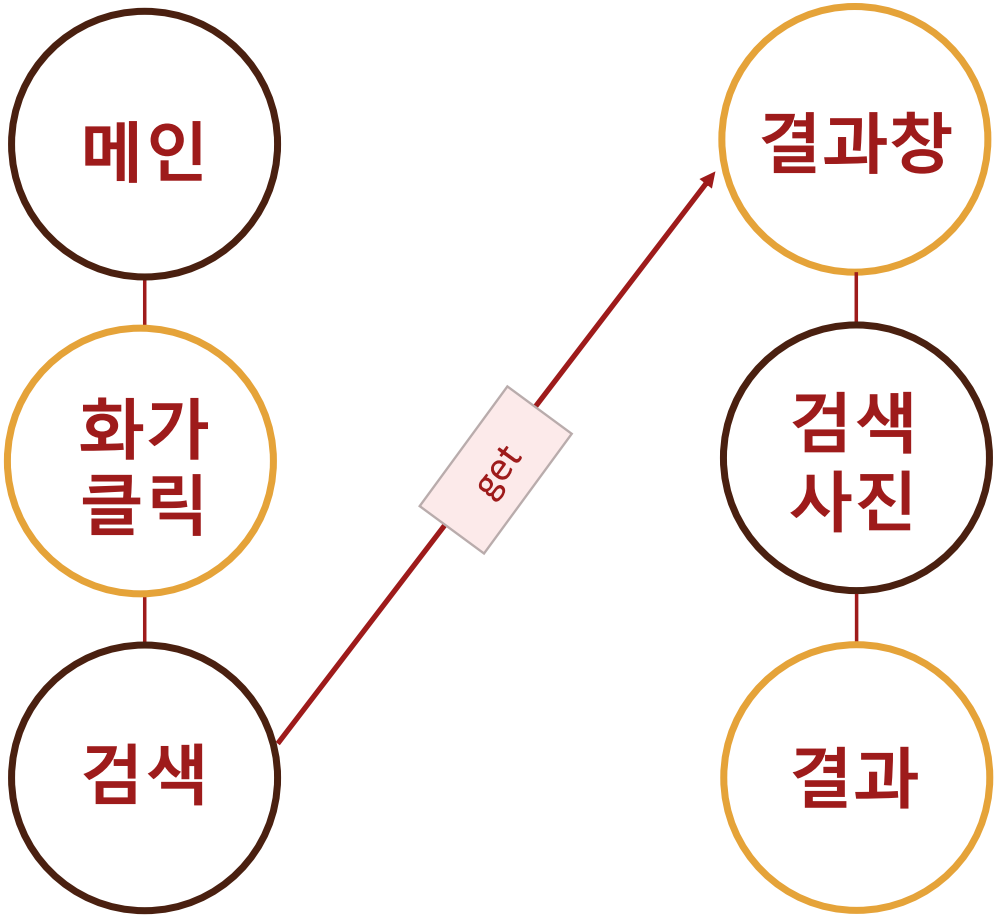
⏰ 생활코딩 진도체크!

생활코딩 MySQL 페이지 링크: https://opentutorials.org/course/3161
생활코딩 FLASK 페이지 링크: https://opentutorials.org/course/4904
유튜브 FastAPI 링크: https://www.youtube.com/channel/UCdTRjVhVcEKBiuUm4r-PHxw/videos
FastAPI GitHub 링크 : https://github.com/riseryan89/notification-api

표 | 표 | 표 (1) | +

Aa 강의	김경은	김보라	김예담	문정태	최혁규	박수정
생활코딩- FLASK	☑	☑	☑	☑	☑	☑
FastAPI 1	☑	☑	☑	☑	☑	☑
FastAPI 2	☑	☑	☑	☑	☑	☑
FastAPI 3	☑	☑	☑	☑	☑	☑
FastAPI 4	☑	☑	☑	☑	☑	☑
FastAPI 5	☑	☑	☑	☑	☑	☑
FastAPI 6	☑	☑	☑	☑	☑	☑
FastAPI 7	☑	☑	☑	☑	☑	☑
FastAPI 8	☑	☑	☑	☑	☑	☑
FastAPI 9	☑	☑	☑	☑	☑	☑
FastAPI 10	☑	☑	☑	☑	☑	☑
FastAPI 11	☑	☑	☑	☑	☑	☑
FastAPI 12	☑	☑	☑	☑	☑	☑
FastAPI 13	☑	☑	☑	☑	☑	☑
FastAPI 14	☑	☑	☑	☑	☑	☑
FastAPI 15	☑	☑	☑	☑	☑	☑
FastAPI 16	☑	☑	☑	☑	☑	☑
FastAPI 17	☑	☑	☑	☑	☑	☑

2.
Web Working Process



Frontend
(HTML5 CSS Bootstrap)

Backend
(Ubuntu Flask)

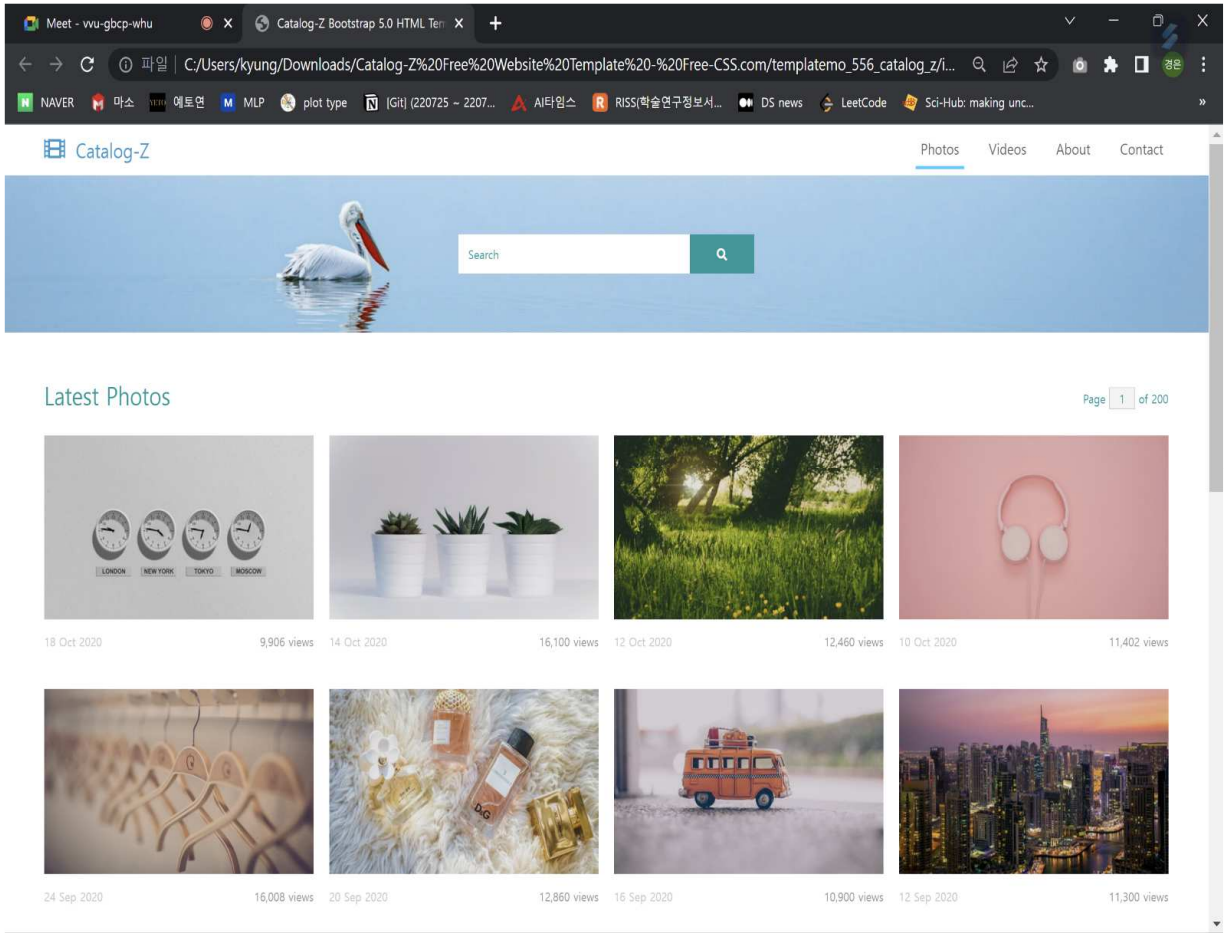
Deploy FLASK API

```
from flask import Flask
from flask import render_template
from flask import request
app = Flask(__name__)
```

```
@app.route("/model", methods=["GET"])
def model():
    url = request.args.get('url')
    answer_labels, prediction_prob_list, ans_prob = ArtistPrediction(url)

    # Figure 1. Show original url figure
    s = io.BytesIO()

    plt.imshow(imageio.v2.imread(url))
    plt.axis('off')
    plt.savefig(s, format='png', bbox_inches="tight")
    plt.close()

    s = base64.b64encode(s.getvalue()).decode("utf-8").replace("\n", "")
    Figure1 = Markup(f'<img align="left" src="%s">' % s)

    # Figure 2. Show bar-plot (probability)
    prob = pd.Series(prediction_prob_list)
    label_freq = ans_prob

    s = io.BytesIO()
    fig = plt.figure(figsize= (10,5))
    ax = plt.subplot(1,1,1)
    ax = sns.barplot(y=label_freq, x=prob, order= label_freq, alpha=0.6)

    ax = plt.xlabel("")
    ax = plt.xticks(fontsize= 13, x=-0.03, y=-0.05)
    ax = plt.yticks(fontsize= 14, x=-0.01)
    ax = plt.title("Label frequency", fontsize= 20, x=0.453, y=1.1)
    plt.savefig(s, format='png', bbox_inches="tight")
    plt.close()

    s = base64.b64encode(s.getvalue()).decode("utf-8").replace("\n", "")
    Figure2 = Markup(f'<img align="left" src="%s">' % s)
    return render_template('about.html', Figure1=Figure1, Figure2=Figure2)
```

Backend

Frontend



崔文朴三金

‘HOW TO’ GUIDE


Return to the Webpage
select an artist
Check the Prediction Result
Prezi

Deployment

http://cdss.waterclean.shop/index.html