PAINTING

TEAM PROJECT

명화 분석기

(산대특) 스마트 팩토리 혁신을 위한 AI 솔루션 개발 양성과정

2025.02.17 ~ 2025.03.07

임세현

김형진

오시윤

CONTENTS

목차

프로젝트 배경

명화 이미지를 분석해
그림의 화가, 장르 및 스타일 예측 시스템 개발

예술 작품에 대한 인공지능 기반의
자동 분류 모델 구축

이미지 인식 기술을 활용하여
미술 작품에 대한 이해도를 높임

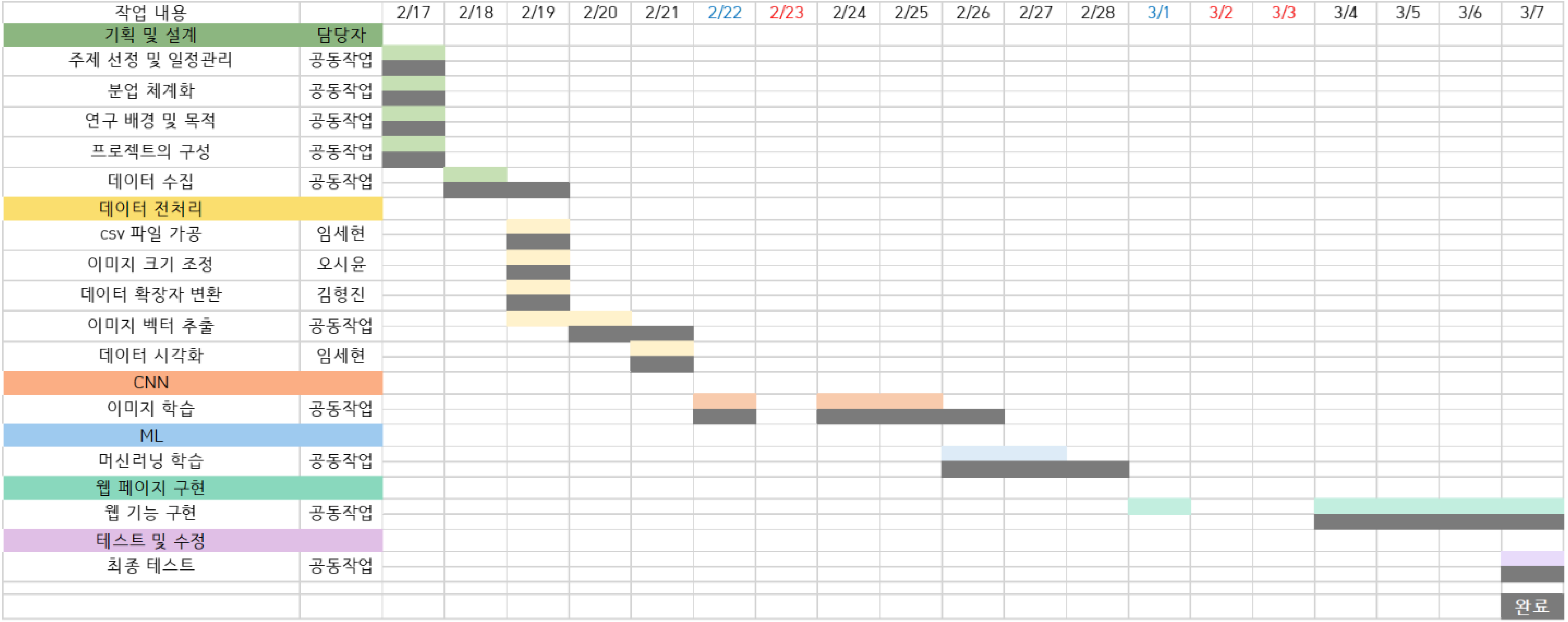
팀 구성 및 역할

임세현 - 팀장	김형진	오시윤
원본 csv 파일 가공 데이터 시각화 PPT 작업	데이터 확장자 변환	이미지 크기 조정
style part	artist part	genre part

CNN, Machine Learning, Web 페이지 구현

프로젝트 일정

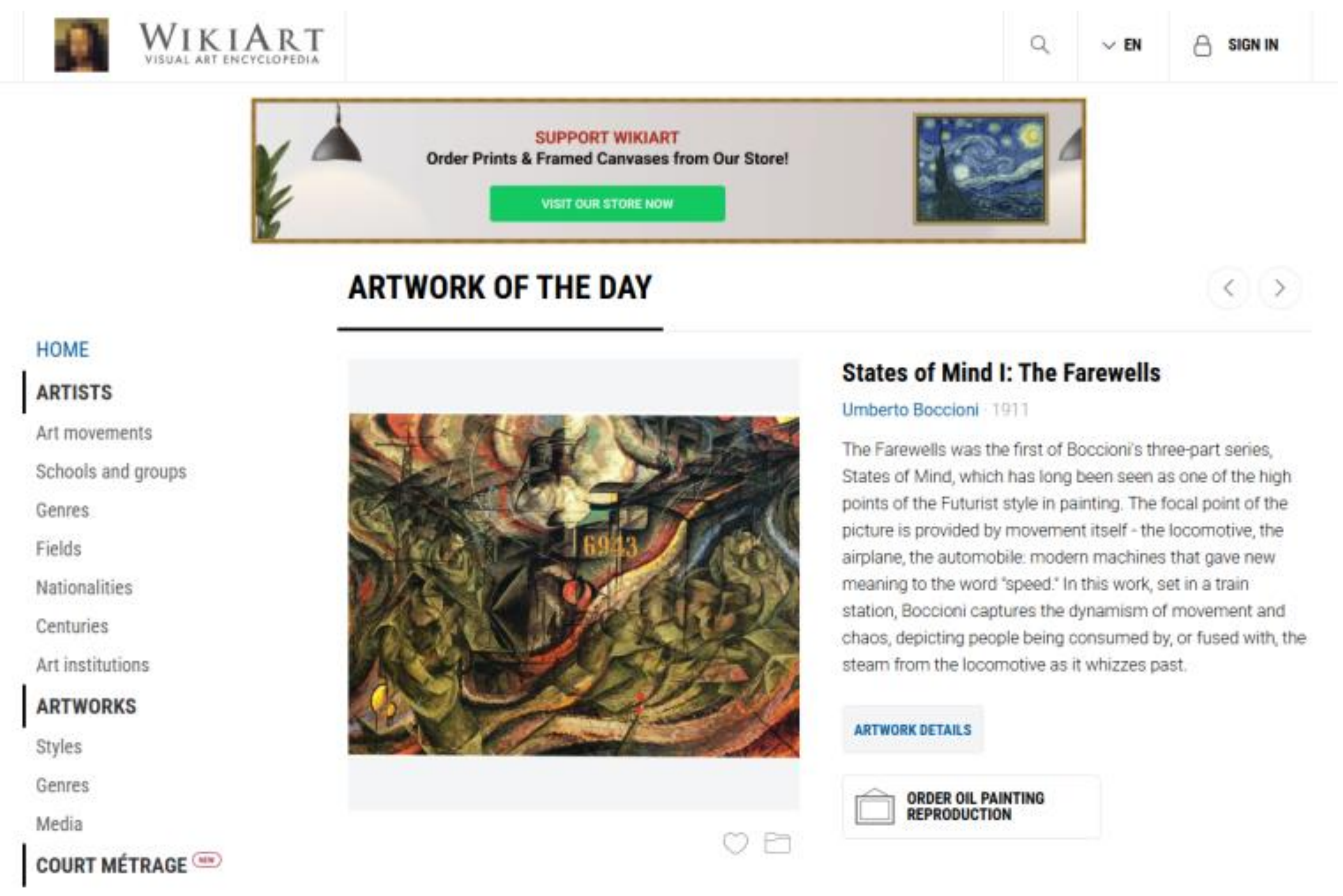
Gantt Chart



개발환경

OS	Window 10
Language	Python 3.10
IDE	anaconda jupyter notebook, pycharm 3.1.1
Open Source	tensorflow 2.10.0 numpy 1.23.5 pandas 1.5.3 scikit-learn 1.2.1 beautifulsoup 4.13.3
Web Framework	django 5.1.6

자료 수집



WikiArt https://www.wikiart.org/

이미지 크기 조정 ▶ numpy 배열로 변환

```
os.makedirs(output_folder, exist_ok=True) # 저장 폴더 없으면 생성

data_size=(128,128) # 사이즈 조절

# 폴더별로 처리
for folder in os.listdir(root_folder):
    folder_path = os.path.join(root_folder, folder)

    if os.path.isdir(folder_path): # 폴더 인지 확인
        save_folder = os.path.join(output_folder, folder)
        os.makedirs(save_folder, exist_ok=True)

        # 파일 변환
        for file in os.listdir(folder_path):
            if file.lower().endswith(('.png','jpg','jpeg')):
                file_path = os.path.join(folder_path, file)
                save_path = os.path.join(save_folder, file)

                img = Image.open(file_path)
                img = img.resize(data_size)
                img.save(save_path)
        print(f'{folder} 폴더 변환 완료')

print('변환 완료')
```

```
data_list = []
for i in range(len(data_csv)):  # 또는 data_csv.shape[0]
    file_path = os.path.join(filepath, data_csv.loc[i, 'file'])

    try:
        img = Image.open(file_path)
        img_array = np.array(img)  # NumPy 배열로 변환
        data_list.append(img_array)  # 변환된 데이터 저장
    except Exception as e:
        print(f"파일 {file_path}을(를) 여는 중 오류 발생: {e}")

    if i % 50 == 0:
        print(f'{i} 번째 완료')

print("모든 데이터 로딩 완료!")
np.save('data.npy',data_list)
```

data.npy 파일 생성

CNN을 위한 이미지 확장자 변환

이미지 벡터 추출

VGG16 모델 사용

```
# VGG16 모델 로드
base_model = VGG16(weights="imagenet", include_top=False, input_shape=(128, 128, 3))
model = Model(inputs=base_model.input, outputs=tf.keras.layers.GlobalAveragePooling2D()(base_model.output))

def extract_features_batch(image_batch):
    """이미지 배치를 받아 특징 벡터 추출"""
    image_batch = preprocess_input(image_batch)  # EfficientNet 전처리 적용
    features = model.predict(image_batch, verbose=0)  # 특징 벡터 추출
    return features  # shape: (batch_size, 1280)

# 데이터셋 로드 (메모리 절약을 위해 float32 변환)
painting_lst = np.array(painting_lst, dtype=np.float32)  # float64 → float32 변환
num_samples = painting_lst.shape[0]

# 배치 단위로 특징 벡터 추출
batch_size = 128  # 메모리 부담을 줄이기 위해 조정
feature_list = []

for i in tqdm(range(0, num_samples, batch_size), desc="Extracting Features"):
    batch = painting_lst[i:i+batch_size]  # 배치 단위로 데이터 가져오기
    features = extract_features_batch(batch)  # 특징 벡터 추출
    feature_list.append(features)

# 모든 특징 벡터를 하나의 배열로 결합
feature_vectors = np.vstack(feature_list)  # 최종 결과 (80158, 1280)

print("Final Feature Vector Shape:", feature_vectors.shape)  # (80158, 1280)
```

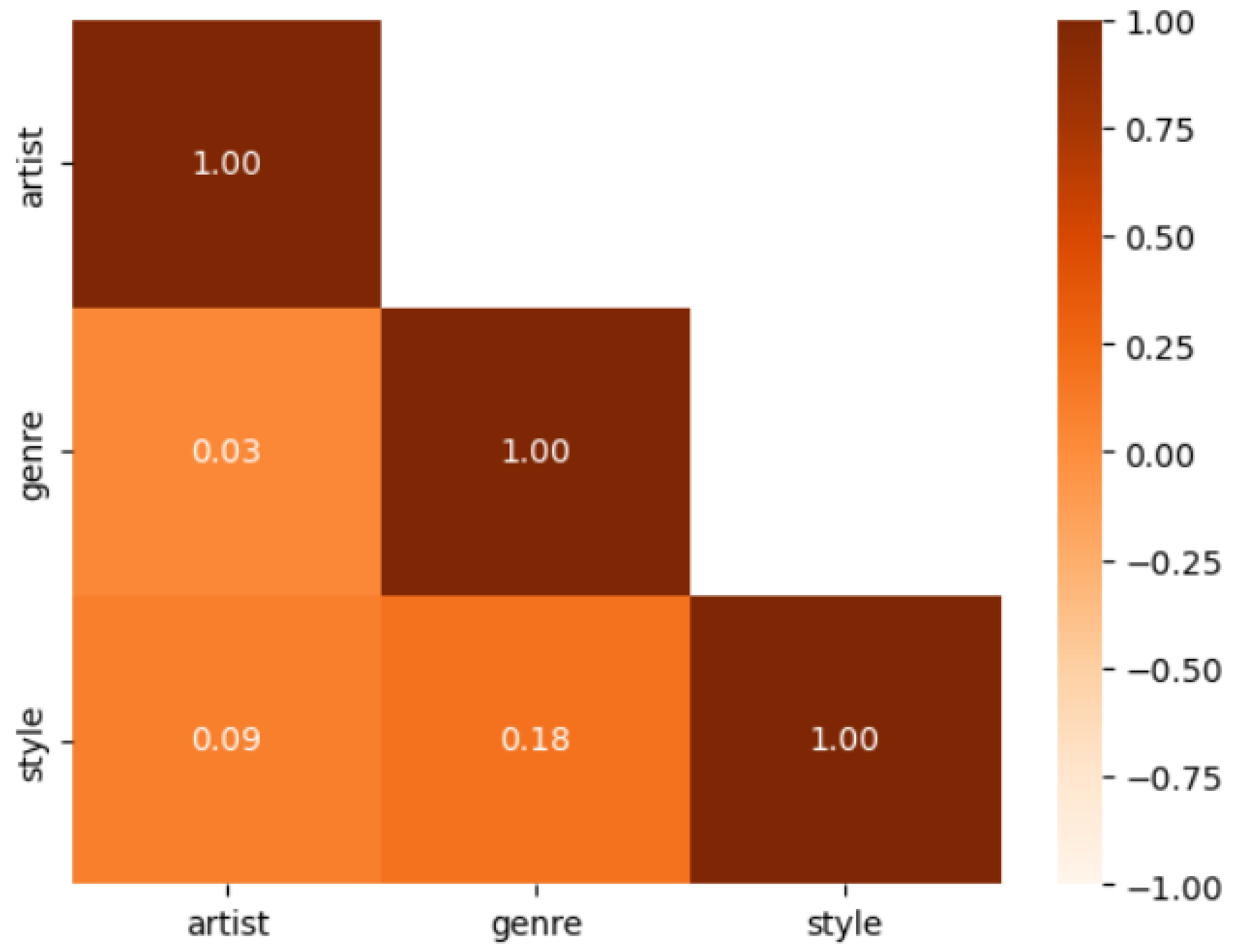
VGG16 모델을 이용해 머신러닝에 적용

```
Extracting Features: 100%|██████████████████████████████████████████████████| 627/627 [04:58<00:00,  2.10it/s]
Final Feature Vector Shape: (80158, 512)
```

```
np.save(os.path.join(filepath,'VGG_vectors.npy'),feature_vectors)
print('모델저장완료')
```

VGG_vectors.npy 생성

데이터 시각화



종속변수 별 상관관계

artist, genre, style 라벨인코딩

데이터 준비

독립변수 data.npy
종속변수 artist, genre, style
라벨인코딩 및 원핫인코딩

»

데이터 40,000개 유지 및
샘플링할 종속변수의 비율 유지

ImageDataGenerator를
이용한 학습 데이터 증강

«

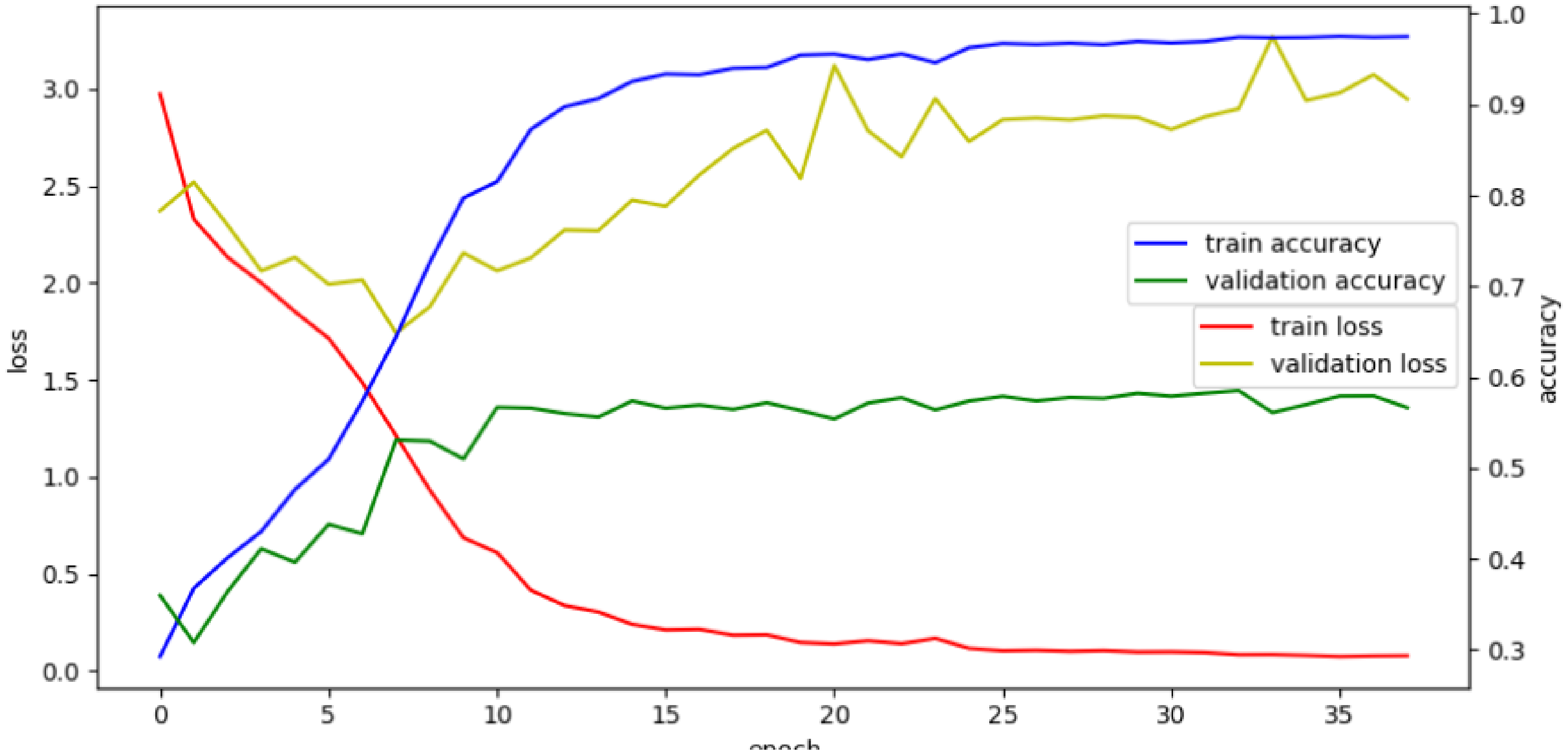
train 데이터 및 test 데이터 분리

학습시키기 (LeNet)

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 128, 128, 32)      2432

 batch_normalization (BatchN  (None, 128, 128, 32)     128
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 64, 64, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 64, 64, 64)        51264

 batch_normalization_1 (Batc  (None, 64, 64, 64)       256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 32, 32, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 32, 32, 128)       73856

 batch_normalization_2 (Batc  (None, 32, 32, 128)      512
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 16, 16, 128)      0
 2D)

 flatten (Flatten)           (None, 32768)             0

 dense (Dense)               (None, 256)               8388864

 batch_normalization_3 (Batc  (None, 256)              1024
 hNormalization)

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 128)               32896

 batch_normalization_4 (Batc  (None, 128)              512
 hNormalization)

 dropout_1 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 44)                5676

=================================================================
Total params: 8,557,420
Trainable params: 8,556,204
Non-trainable params: 1,216
_________________________________________________________________
```

```python
# 모델평가
loss, accuracy = model.evaluate(X_test, y_test)
print('accuracy : {:.2f}%'.format(accuracy*100))
```

```
376/376 [==============================] - 4s 11ms/step - loss: 2.9472 - accuracy: 0.5665
accuracy : 56.65%
```

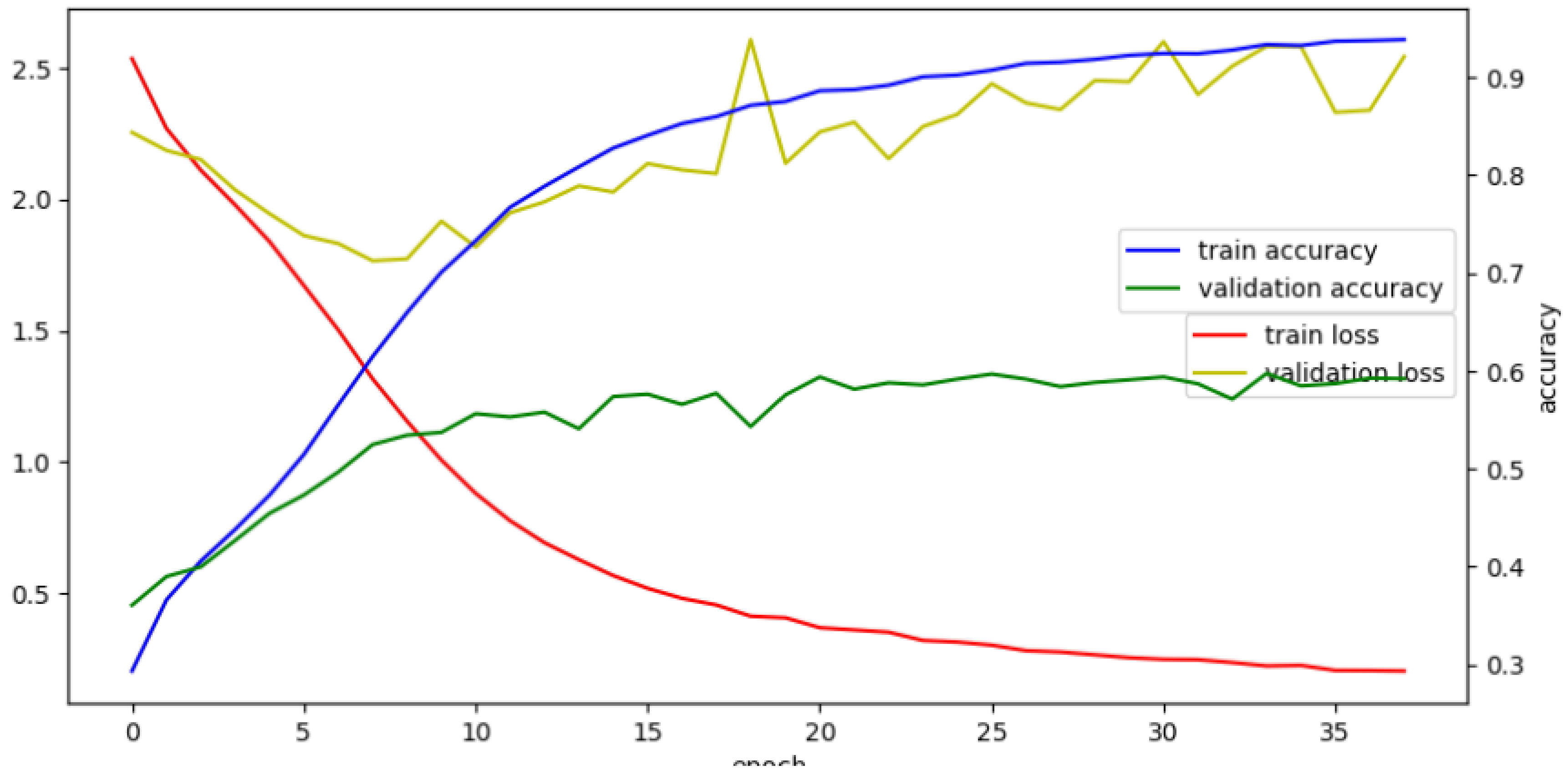


학습시키기 (AlexNet)

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 64, 64, 32)        2432

 max_pooling2d (MaxPooling2D  (None, 32, 32, 32)       0
 )

 batch_normalization (BatchN  (None, 32, 32, 32)       128
 ormalization)

 conv2d_1 (Conv2D)           (None, 32, 32, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 batch_normalization_1 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 conv2d_2 (Conv2D)           (None, 16, 16, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 8, 8, 128)        0
 2D)

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 512)               4194816

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 512)               262656

 dropout_1 (Dropout)         (None, 512)               0

 dense_2 (Dense)             (None, 44)                22572

=================================================================
Total params: 4,575,212
Trainable params: 4,575,020
Non-trainable params: 192
_________________________________________________________________
```

```python
# 모델평가
loss, accuracy = model.evaluate(X_test, y_test)
print('accuracy : {:.2f}%'.format(accuracy*100))
```

```
376/376 [==============================] - 2s 4ms/step - loss: 2.5418 - accuracy: 0.5919
accuracy : 59.19%
```

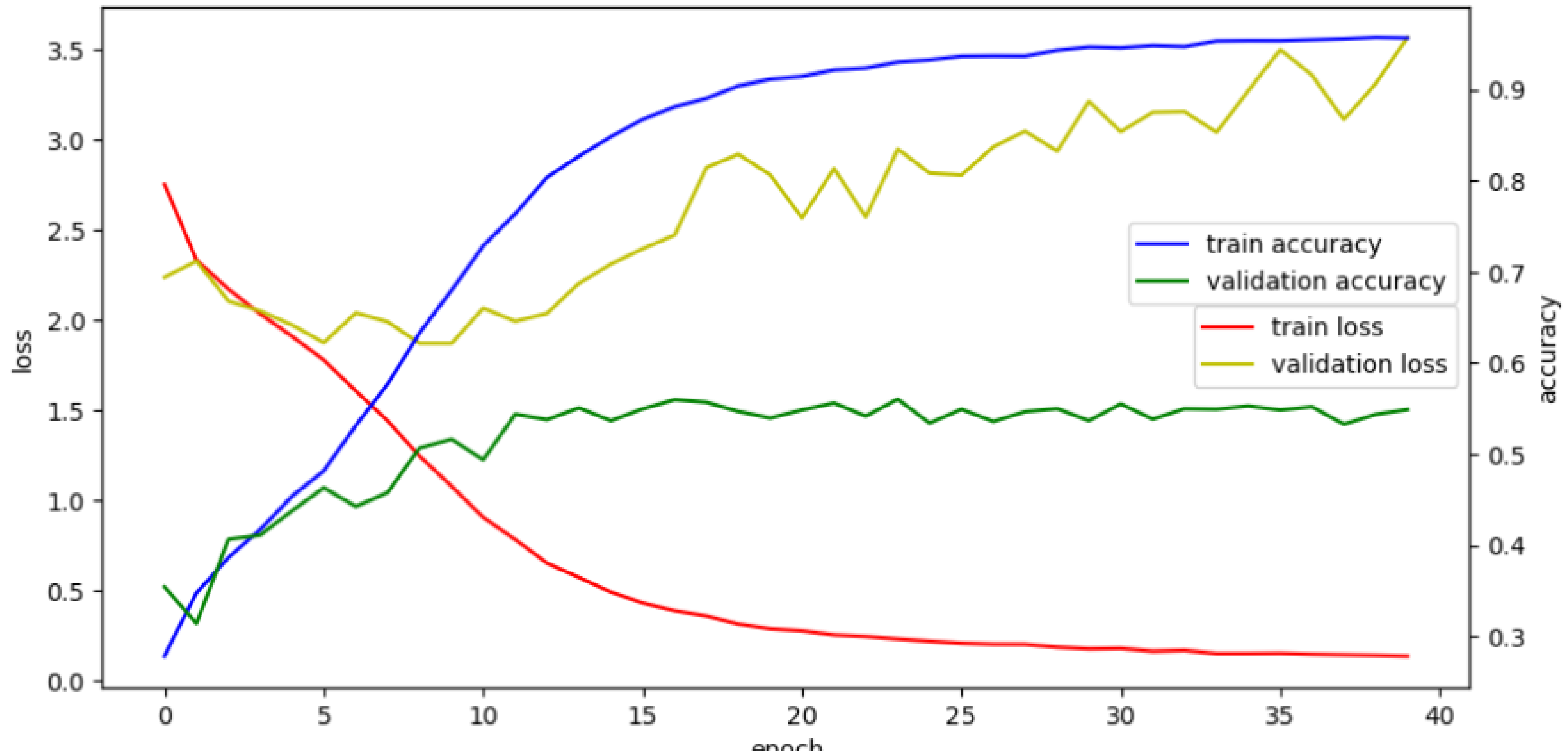


학습시키기 (VGGNet)

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 128, 128, 32)      896

 conv2d_7 (Conv2D)           (None, 128, 128, 32)      9248

 batch_normalization_4 (Batc  (None, 128, 128, 32)     128
 hNormalization)

 max_pooling2d_3 (MaxPooling  (None, 64, 64, 32)       0
 2D)

 conv2d_8 (Conv2D)           (None, 64, 64, 64)        18496

 conv2d_9 (Conv2D)           (None, 64, 64, 64)        36928

 batch_normalization_5 (Batc  (None, 64, 64, 64)       256
 hNormalization)

 max_pooling2d_4 (MaxPooling  (None, 32, 32, 64)       0
 2D)

 conv2d_10 (Conv2D)          (None, 32, 32, 128)       73856

 conv2d_11 (Conv2D)          (None, 32, 32, 128)       147584

 batch_normalization_6 (Batc  (None, 32, 32, 128)      512
 hNormalization)

 max_pooling2d_5 (MaxPooling  (None, 16, 16, 128)      0
 2D)

 flatten_1 (Flatten)         (None, 32768)             0

 dense_1 (Dense)             (None, 512)               16777728

 batch_normalization_7 (Batc  (None, 512)              2048
 hNormalization)

 leaky_re_lu (LeakyReLU)     (None, 512)               0

 dropout (Dropout)           (None, 512)               0

 dense_2 (Dense)             (None, 256)               131328

 batch_normalization_8 (Batc  (None, 256)              1024
 hNormalization)

 leaky_re_lu_1 (LeakyReLU)   (None, 256)               0

 dropout_1 (Dropout)         (None, 256)               0

 dense_3 (Dense)             (None, 44)                11308

=================================================================
Total params: 17,211,340
Trainable params: 17,209,356
Non-trainable params: 1,984
_________________________________________________________________
```

```python
# 모델평가
loss, accuracy = model.evaluate(X_test, y_test)
print('accuracy : {:.2f}%'.format(accuracy*100))
```

```
376/376 [==============================] - 6s 16ms/step - loss: 3.5693 - accuracy: 0.5488
accuracy : 54.88%
```

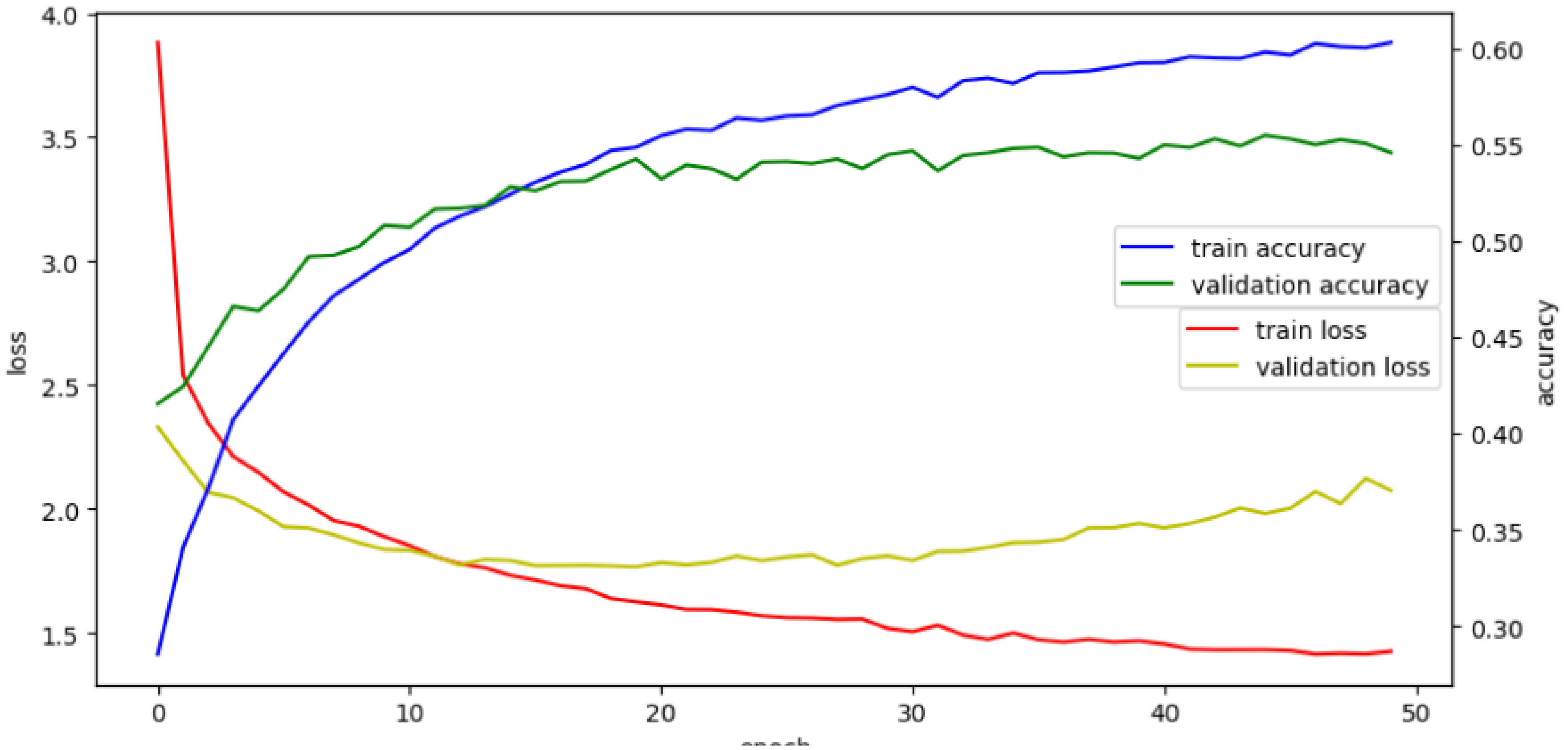


학습시키기 (VGG16)

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 128, 128, 3)]     0

 block1_conv1 (Conv2D)       (None, 128, 128, 64)      1792

 block1_conv2 (Conv2D)       (None, 128, 128, 64)      36928

 block1_pool (MaxPooling2D)  (None, 64, 64, 64)        0

 block2_conv1 (Conv2D)       (None, 64, 64, 128)       73856

 block2_conv2 (Conv2D)       (None, 64, 64, 128)       147584

 block2_pool (MaxPooling2D)  (None, 32, 32, 128)       0

 block3_conv1 (Conv2D)       (None, 32, 32, 256)       295168

 block3_conv2 (Conv2D)       (None, 32, 32, 256)       590080

 block3_conv3 (Conv2D)       (None, 32, 32, 256)       590080

 block3_pool (MaxPooling2D)  (None, 16, 16, 256)       0

 block4_conv1 (Conv2D)       (None, 16, 16, 512)       1180160

 block4_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block4_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 8, 8, 512)         0

 block5_conv1 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_conv2 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_conv3 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 512)               4194816

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 44)                11308

=================================================================
Total params: 19,052,140
Trainable params: 4,337,452
Non-trainable params: 14,714,688
_________________________________________________________________
```

```
# 모델평가
loss, accuracy = model.evaluate(X_test, y_test)
print('accuracy : {:.2f}%'.format(accuracy*100))
```

```
376/376 [==============================] - 19s 51ms/step - loss: 1.7681 - accuracy: 0.5425
accuracy : 54.25%
```



ML

데이터 준비

STEP 01
독립변수

STEP 02
종속변수

STEP 03
스케일조정

STEP 04
데이터 분리

이미지 벡터 추출
VGG_vectors.npy

genre

SMOTE와
NearMiss 사용

train 데이터
test 데이터

분류분석

DecisionTreeClassifier

Classification Report:

	precision	recall	f1-score	support
Canadian	0.9956	0.9998	0.9977	4288
abstract	0.6912	0.6246	0.6562	4289
advertisement	0.9821	0.9977	0.9898	4289
allegorical painting	0.8284	0.8829	0.8548	4288
animal painting	0.7996	0.8293	0.8142	4288
battle painting	0.9173	0.9681	0.9420	4288
bird-and-flower painting	0.9926	0.9984	0.9955	4288
capriccio	0.9889	0.9958	0.9923	4289
caricature	0.9497	0.9783	0.9638	4288
cityscape	0.6477	0.6134	0.6301	4289
cloudscape	0.9506	0.9862	0.9681	4288
design	0.7711	0.8004	0.7854	4288
figurative	0.7851	0.8120	0.7983	4288
flower painting	0.8632	0.8839	0.8734	4289
genre painting	0.2765	0.2281	0.2500	4288
graffiti	0.9984	0.9998	0.9991	4288
history painting	0.8342	0.8955	0.8638	4288
illustration	0.7291	0.7369	0.7330	4288
installation	0.9575	0.9781	0.9677	4288
interior	0.8856	0.9242	0.9045	4288
landscape	0.4506	0.3640	0.4027	4288
literary painting	0.8694	0.9343	0.9007	4289
marina	0.7788	0.8284	0.8028	4288
miniature	0.9981	0.9998	0.9990	4288
mythological painting	0.7259	0.7558	0.7405	4288
nude painting	0.7178	0.7194	0.7186	4288
nude painting (nu)	0.9728	0.9937	0.9832	4288
panorama	0.9972	1.0000	0.9986	4288
pastorale	0.9786	0.9935	0.9860	4288
photo	0.9701	0.9909	0.9804	4288
portrait	0.4397	0.3357	0.3808	4289
poster	0.9557	0.9806	0.9680	4288
quadratura	0.9972	0.9993	0.9983	4289
religious painting	0.4467	0.3909	0.4169	4288
sculpture	0.9012	0.9167	0.9089	4288
self-portrait	0.7985	0.8347	0.8162	4288
sketch and study	0.6397	0.6294	0.6345	4288
still life	0.6777	0.6364	0.6564	4288
symbolic painting	0.8081	0.8550	0.8309	4289
tessellation	0.9998	0.9998	0.9998	4288
vanitas	0.9889	0.9984	0.9936	4288
veduta	0.9741	0.9923	0.9831	4288
wildlife painting	0.9699	0.9932	0.9814	4288
yakusha-e	0.9954	0.9993	0.9973	4288
accuracy			0.8472	188681
macro avg	0.8386	0.8472	0.8422	188681
weighted avg	0.8386	0.8472	0.8422	188681

Precision: 0.8386
Recall: 0.8472
F1-score: 0.8422

MLPClassifier

Classification Report:

	precision	recall	f1-score	support
Canadian	0.9974	1.0000	0.9987	4288
abstract	0.7319	0.6239	0.6736	4289
advertisement	0.9853	1.0000	0.9926	4289
allegorical painting	0.8334	0.8624	0.8477	4288
animal painting	0.7989	0.8468	0.8221	4288
battle painting	0.9431	0.9774	0.9599	4288
bird-and-flower painting	0.9974	0.9981	0.9978	4288
capriccio	0.9958	0.9984	0.9971	4289
caricature	0.9734	0.9741	0.9738	4288
cityscape	0.7126	0.5994	0.6511	4289
cloudscape	0.9779	0.9916	0.9847	4288
design	0.6456	0.7976	0.7136	4288
figurative	0.7037	0.7910	0.7448	4288
flower painting	0.8543	0.8547	0.8545	4289
genre painting	0.2899	0.2031	0.2389	4288
graffiti	1.0000	1.0000	1.0000	4288
history painting	0.8545	0.8710	0.8627	4288
illustration	0.5114	0.6880	0.5867	4288
installation	0.9642	0.9916	0.9777	4288
interior	0.9344	0.9128	0.9234	4288
landscape	0.5617	0.4841	0.5200	4288
literary painting	0.8802	0.9555	0.9163	4289
marina	0.7746	0.8309	0.8018	4288
miniature	0.9986	1.0000	0.9993	4288
mythological painting	0.5939	0.5385	0.5648	4288
nude painting	0.7036	0.5763	0.6336	4288
nude painting (nu)	0.9480	0.9993	0.9730	4288
panorama	0.9965	1.0000	0.9983	4288
pastorale	0.9880	0.9967	0.9923	4288
photo	0.9838	0.9928	0.9883	4288
portrait	0.5076	0.3598	0.4211	4289
poster	0.9618	0.9932	0.9773	4288
quadratura	0.9961	1.0000	0.9980	4289
religious painting	0.4668	0.2757	0.3466	4288
sculpture	0.9286	0.9741	0.9508	4288
self-portrait	0.6625	0.8461	0.7431	4288
sketch and study	0.6144	0.5499	0.5804	4288
still life	0.6832	0.6544	0.6685	4288
symbolic painting	0.7103	0.9433	0.8104	4289
tessellation	0.9993	1.0000	0.9997	4288
vanitas	0.9914	1.0000	0.9957	4288
veduta	0.9862	0.9979	0.9920	4288
wildlife painting	0.9781	0.9984	0.9881	4288
yakusha-e	0.9986	1.0000	0.9993	4288
accuracy			0.8397	188681
macro avg	0.8322	0.8397	0.8332	188681
weighted avg	0.8322	0.8397	0.8332	188681

Precision: 0.8322
Recall: 0.8397
F1-score: 0.8332

앙상블 모형

RandomForestClassifier, XGBClassifier, LGBMClassifier

```
def model_measure(model, X_train=X_train, y_train=y_train, X_test=X_test, y_test=y_test):
    model.fit(X_train, y_train)
    pred = model.predict(X_test)
    accuracy  = model.score(X_test, y_test)
    precision = precision_score(y_test, pred, average="macro")
    recall    = recall_score(y_test, pred, average="macro")
    f1score  = f1_score(y_test, pred, average="macro")
    return '정확도:{:.3f}, 정밀도:{:.3f}, 재현율:{:.3f}, f1_score:{:.3f}'.format(accuracy, precision, recall, f1score)
```

```
rf_model = model_measure(RandomForestClassifier())
rf_model
```

'정확도:0.958, 정밀도:0.956, 재현율:0.958, f1_score:0.956'

```
xgb = model_measure(XGBClassifier())
xgb
```

'정확도:0.942, 정밀도:0.941, 재현율:0.942, f1_score:0.941'

```
lgb = model_measure(LGBMClassifier(force_col_wise=True))
lgb
```

'정확도:0.921, 정밀도:0.920, 재현율:0.921, f1_score:0.920'

앙상블 모형

RandomForestClassifier, XGBClassifier, LGBMClassifier

```python
le = LabelEncoder()
train_y = le.fit_transform(train_y)
test_y = le.fit_transform(test_y)

def model_measure(model, train_X=train_X, train_y=train_y, test_X=test_X, test_y=test_y):
    model.fit(train_X, train_y)
    pred = model.predict(test_X)
    accuracy  = model.score(test_X, test_y)
    precision = precision_score(test_y, pred, average="macro")
    recall    = recall_score(test_y, pred, average="macro")
    f1score   = f1_score(test_y, pred, average="macro")
    return '정확도:{:.3f}, 정밀도:{:.3f}, 재현율:{:.3f}, f1_score:{:.3f}'.format(accuracy, precision, recall, f1score)
```

```
정확도:0.954, 정밀도:0.951, 재현율:0.954, f1_score:0.951
정확도:0.900, 정밀도:0.896, 재현율:0.900, f1_score:0.897
정확도:0.921, 정밀도:0.920, 재현율:0.921, f1_score:0.920
```

```python
# ✅ 경량화된 랜덤포레스트 모델
rf_model = RandomForestClassifier(
    n_estimators=50,       # 트리 개수 줄이기
    max_depth=5,           # 트리 깊이 제한
    max_features='sqrt',   # 최적의 특징 개수 자동 선택
    random_state=42
)

# ✅ 경량화된 XGBoost 모델
xgb_model = XGBClassifier(
    max_depth=4,           # 트리 깊이 제한
    n_estimators=50,       # 트리 개수 줄이기
    learning_rate=0.1,     # 학습 속도 증가
    subsample=0.8,         # 데이터 일부 샘플링
    colsample_bytree=0.8,  # 일부 특성만 사용
    tree_method='hist',    # 히스토그램 기반 트리 (메모리 절약)
    eval_metric='logloss',
#    use_label_encoder=True,
    random_state=42
)

# ✅ 경량화된 LightGBM 모델
lgb_model = LGBMClassifier(
    n_estimators=50,       # 트리 개수 줄이기
    max_depth=4,           # 트리 깊이 제한
    num_leaves=16,         # 리프 개수 줄이기
    subsample=0.8,         # 데이터 일부 샘플링
    colsample_bytree=0.8,  # 일부 특성만 사용
    verbose=-1,            # 불필요한 출력 제거
    random_state=42
)
```

투표를 이용한 앙상블

VotingClassifier(voting = hard/soft) 비교

예측

```
voting_model_hard.predict(X_test[0].reshape(1, -1))
```

array(['landscape'], dtype=object)

```
voting_model_soft.predict(X_test[0].reshape(1, -1))
```

array(['landscape'], dtype=object)

```
y_test[0]
```

20

실제값

```
y_inverse_test = le.inverse_transform(y_test)
y_inverse_test[0]
```

'landscape'

```
model_measure(voting_model_hard)
```

'정확도:0.770, 정밀도:0.762, 재현율:0.770, f1_score:0.758'

```
model_measure(voting_model_soft)
```

'정확도:0.801, 정밀도:0.794, 재현율:0.801, f1_score:0.795'

voting_model_hard < voting_model_soft

웹 페이지 구현

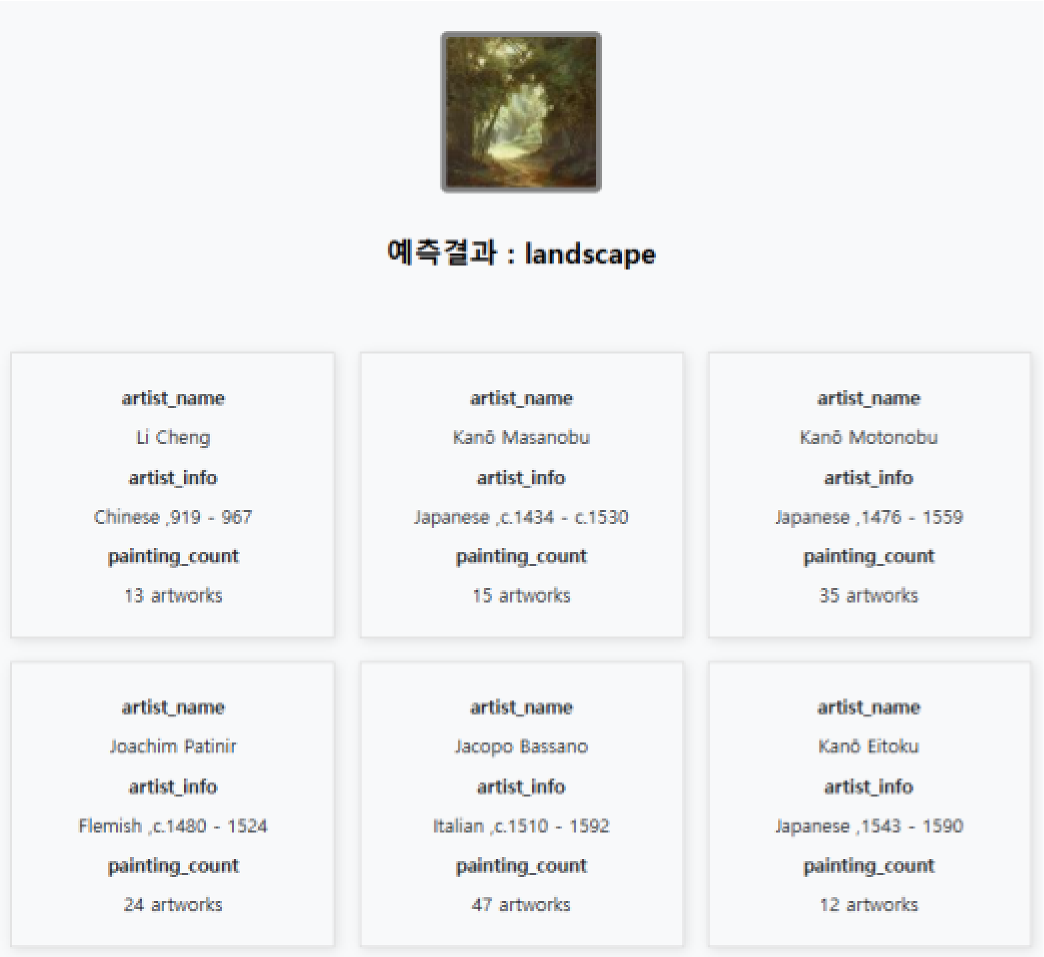
① navigator bar
원하는 예측 선택

② file 업로드
이미지 선택

③ Upload
결과 페이지

웹 페이지 구현

Style 예측 결과



Genre 예측 결과

Artist 예측 결과

결론

기능적 측면

이미지에 대한 style, genre, artist 별 예측 가능

프로젝트 의의

예술과 AI 기술을 결합하여
미술 감상과 연구를 새로운 방식으로 접근

개선 방안

데이터의 다양성	특정 출처의 데이터 뿐만 아니라 공공 데이터셋 활용
모델 성능 향상	학습 결과에 대한 정확도 높이기
웹 페이지 기능 구체화	예측 결과에 대한 신뢰도(accuracy) 제공