

영화 분석기



(산대특) 스마트 팩토리 혁신을 위한 AI 솔루션 개발 양성과정

2025.02.17 ~ 2025.03.07



임세현



김형진



오시윤

목차

서론

CONTENTS 01

데이터
전처리

CONTENTS 02

CNN

CONTENTS 03

Machine
Learning

CONTENTS 04

웹 페이지

CONTENTS 05

결론

CONTENTS 06



프로젝트 배경

영화 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

원본 csv 파일 가공
데이터 시각화
PPT 작업



style part

김형진

데이터 형식 변환



artist part

오시윤

이미지 크기 조정

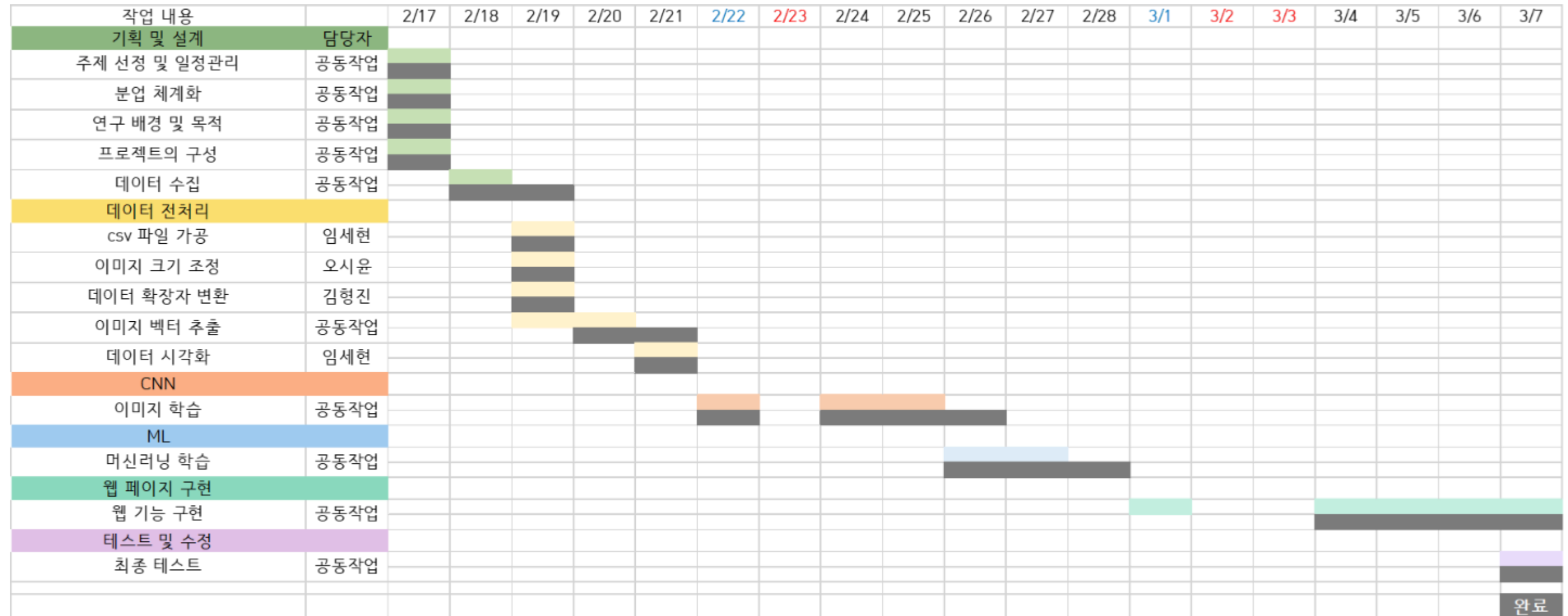


genre part

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

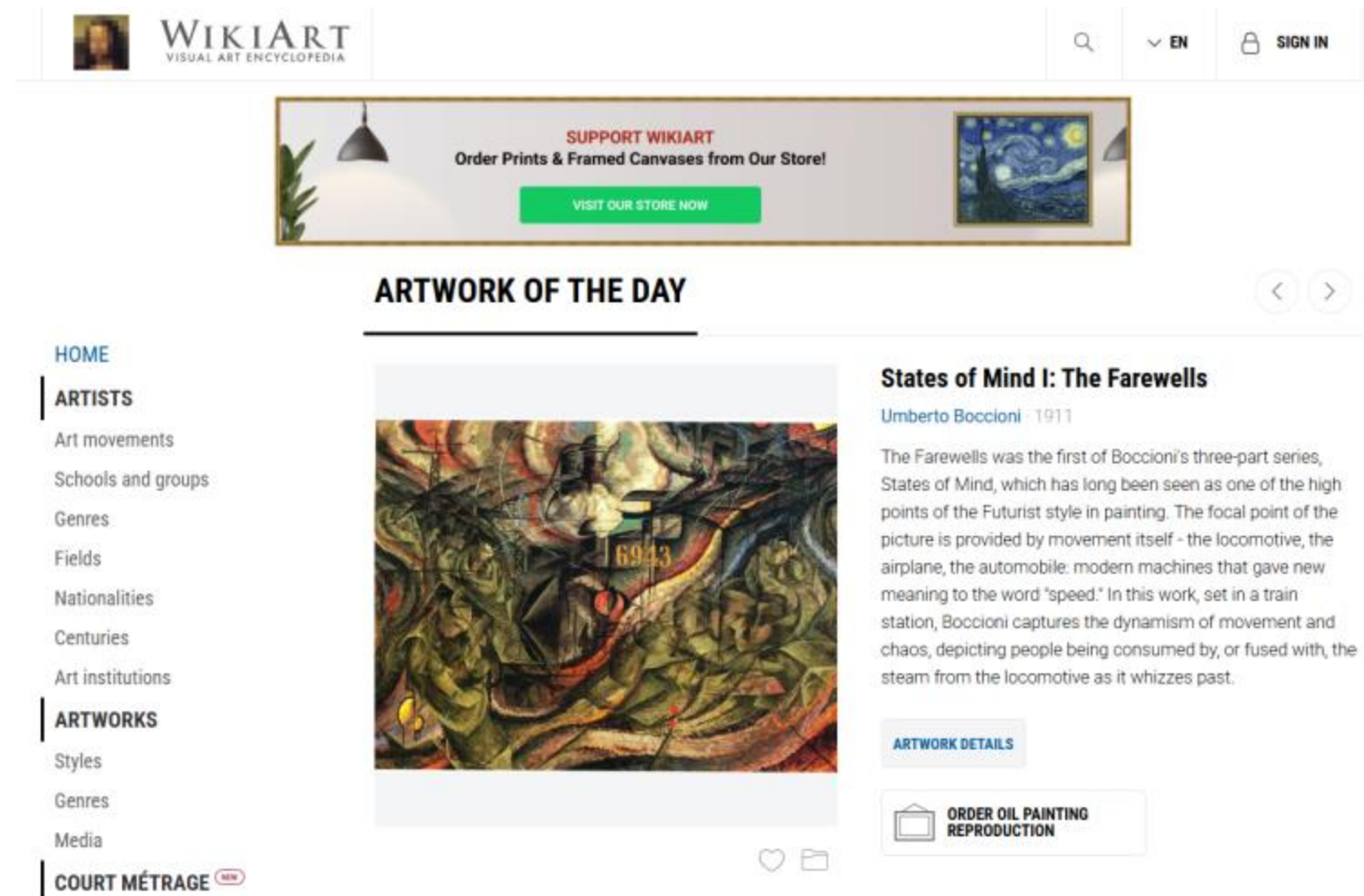
Open Source

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/>

csv 파일 가공

	file	artist	genre	style
0	Realism/vincent-van-gogh_pine-trees-in-the-fen...	22	133	161
1	Baroque/rembrandt_the-angel-appearing-to-the-s...	20	136	144
2	Post_Impressionism/paul-cezanne_portrait-of-th...	16	135	160
3	Impressionism/pierre-auguste-renoir_young-girl...	17	131	152
4	Romanticism/ivan-aivazovsky_morning-1851.jpg	9	139	163



	file	title	artist	genre	style
0	Realism/vincent-van-gogh_pine-trees-in-the-fen...	pine-trees-in-the-fen-1884	vincent-van-gogh	landscape	Realism
1	Baroque/rembrandt_the-angel-appearing-to-the-s...	the-angel-appearing-to-the-shepherds-1634	rembrandt	religious painting	Baroque
2	Post_Impressionism/paul-cezanne_portrait-of-th...	portrait-of-the-artist-s-son	paul-cezanne	portrait	Post_Impressionism
3	Impressionism/pierre-auguste-renoir_young-girl...	young-girl-seated-in-a-meadow-1916	pierre-auguste-renoir	genre painting	Impressionism
4	Romanticism/ivan-aivazovsky_morning-1851.jpg	morning-1851	ivan-aivazovsky	marina	Romanticism

split, map, lambda 함수를 이용해 최종 painting.csv 파일 생성

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, ResNet50, EfficientNetB0 모델 사용

```
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)  
  
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)
```

```
painting_lst = np.load(os.path.join(filepath, 'data.npy'))  
painting_lst.shape
```

(80158, 128, 128, 3)

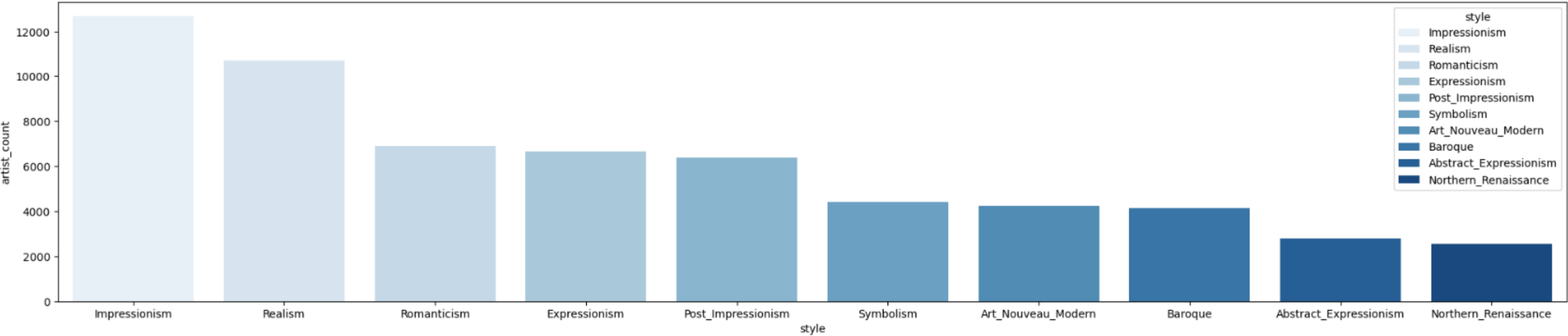


Extracting Features: 100% | 627/627 [04:58<00:00, 2.10it/s]
Final Feature Vector Shape: (80158, 512)

이용한 3가지 모델 중
가장 가벼운 VGG16 모델을 이용해 머신러닝에 적용

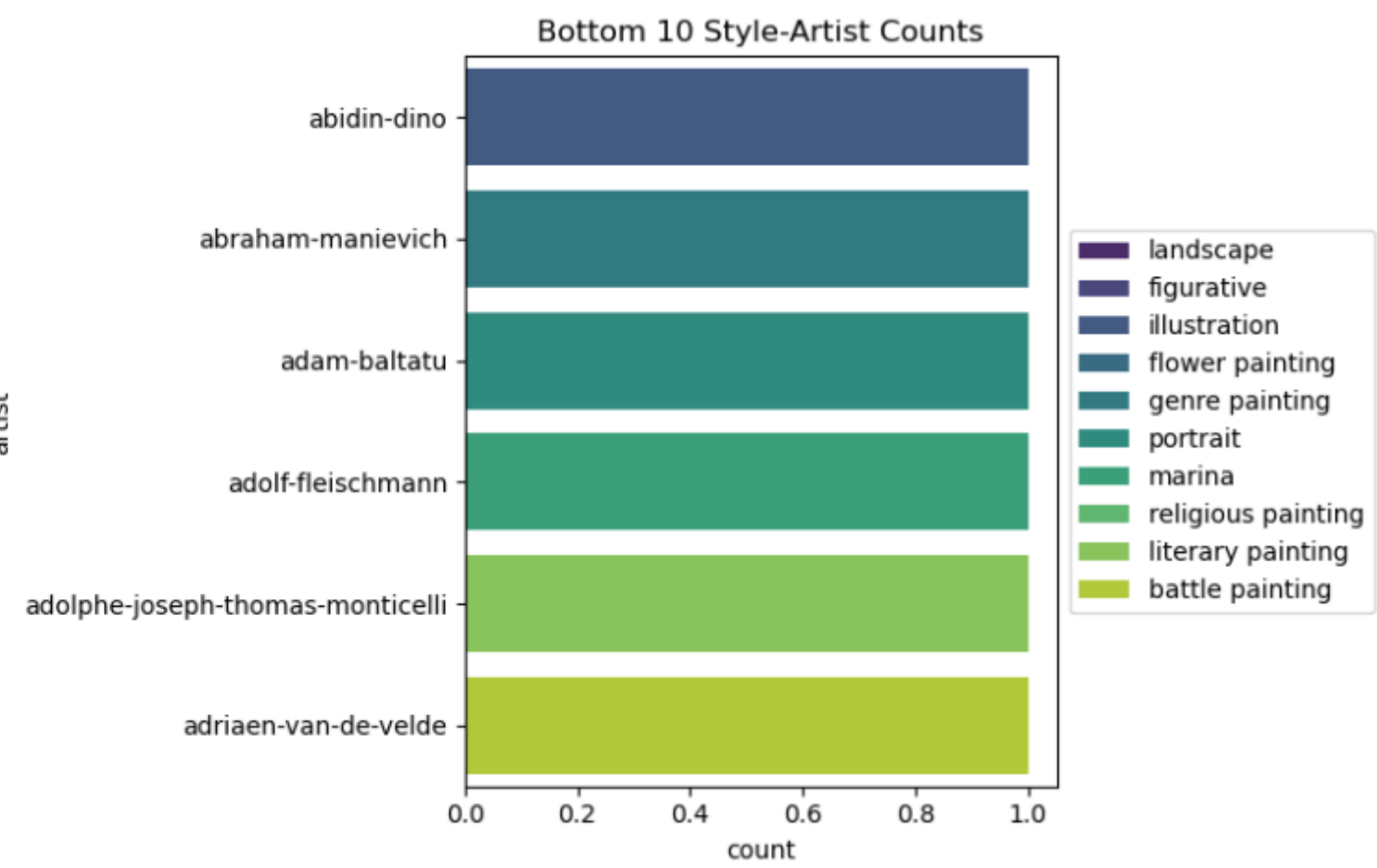
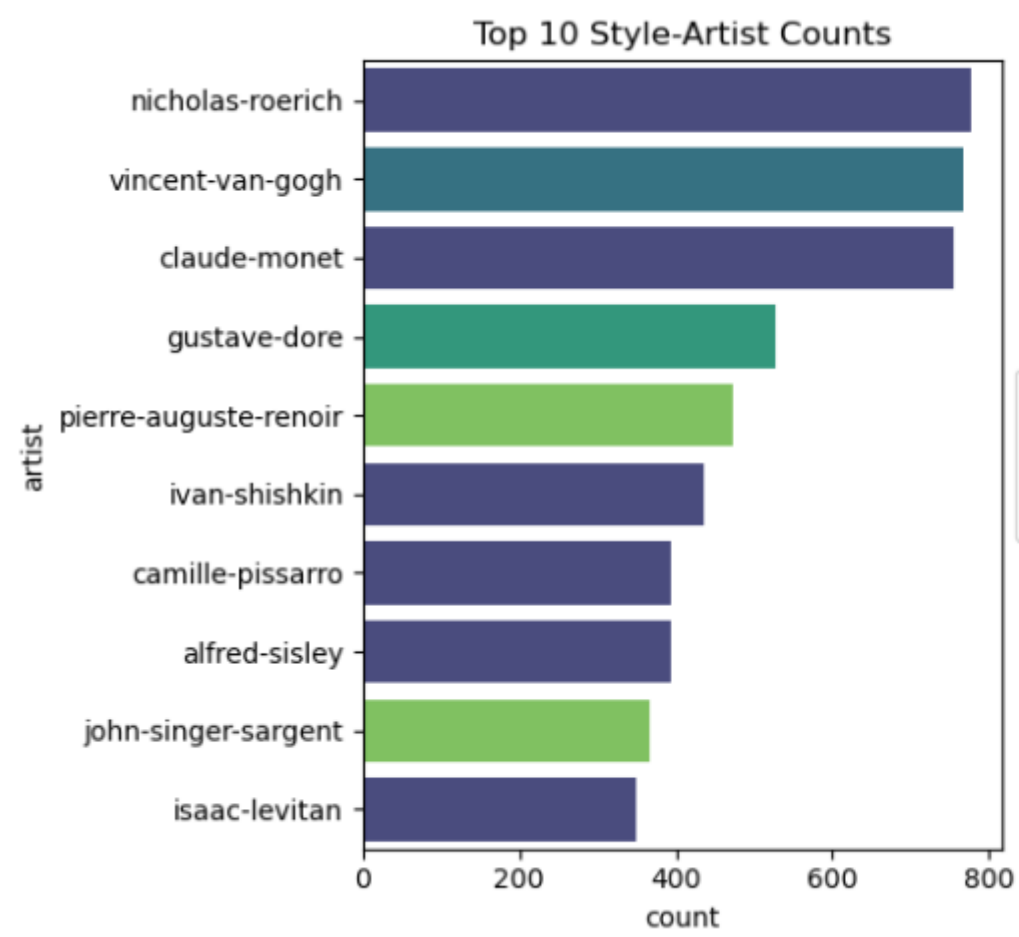
VGG_vectors.npy 생성

데이터 시각화



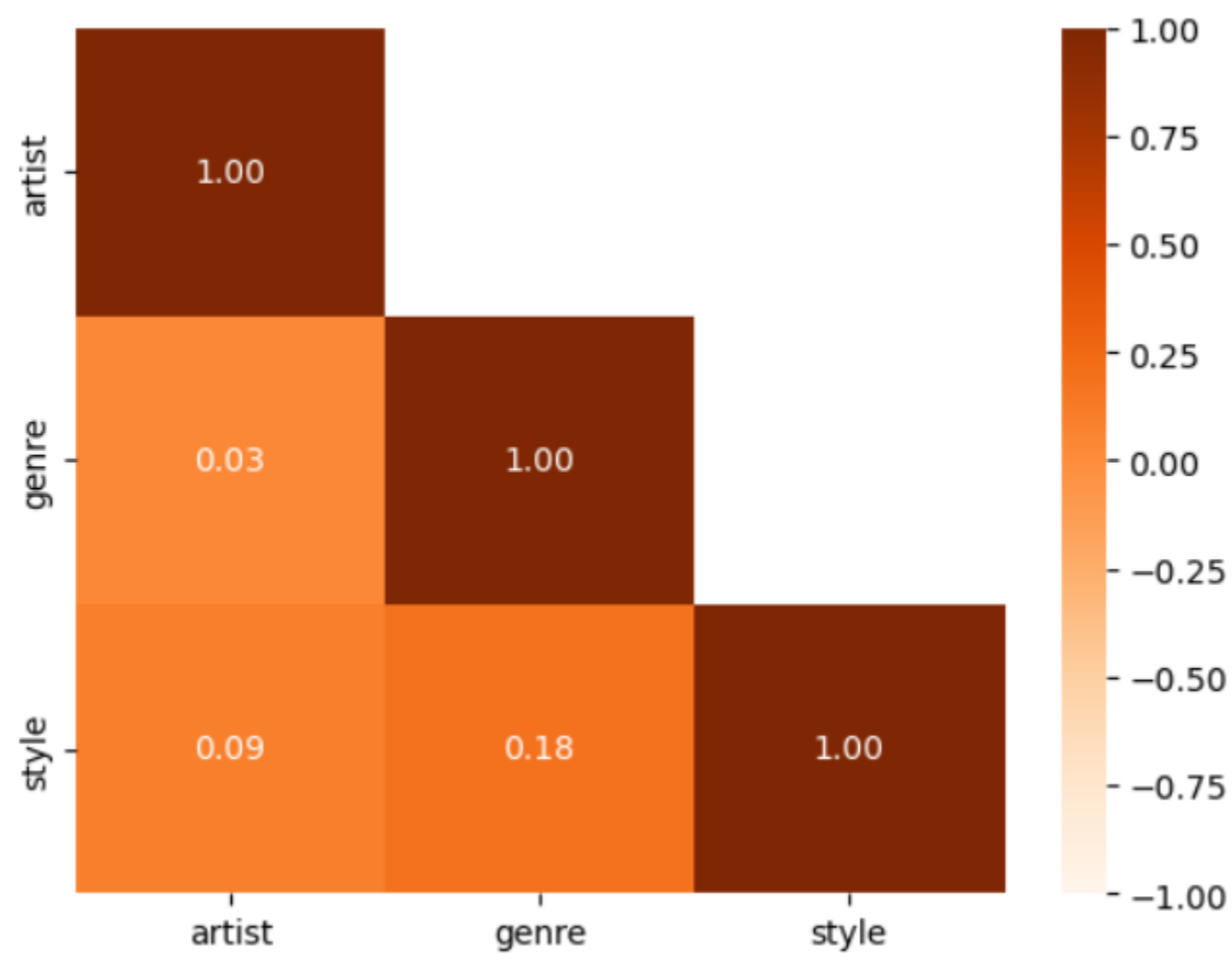
Style 별 Artist 수

데이터 시각화



Artist별 Genre 수

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



train 데이터 및 test 데이터 분리



ImageDataGenerator를
이용한 학습 데이터 증강

학습시키기

Model: "sequential_4"

Layer (type)	Output Shape	Param #
conv2d_16 (Conv2D)	(None, 128, 128, 16)	448
batch_normalization_12 (BatchNormalization)	(None, 128, 128, 16)	64
max_pooling2d_12 (MaxPooling2D)	(None, 64, 64, 16)	0
dropout_16 (Dropout)	(None, 64, 64, 16)	0
conv2d_17 (Conv2D)	(None, 64, 64, 32)	4,640
batch_normalization_13 (BatchNormalization)	(None, 64, 64, 32)	128
max_pooling2d_13 (MaxPooling2D)	(None, 32, 32, 32)	0
dropout_17 (Dropout)	(None, 32, 32, 32)	0
conv2d_18 (Conv2D)	(None, 32, 32, 64)	18,496
batch_normalization_14 (BatchNormalization)	(None, 32, 32, 64)	256
max_pooling2d_14 (MaxPooling2D)	(None, 16, 16, 64)	0
dropout_18 (Dropout)	(None, 16, 16, 64)	0
flatten_4 (Flatten)	(None, 16384)	0
dense_8 (Dense)	(None, 128)	2,097,280
dropout_19 (Dropout)	(None, 128)	0
dense_9 (Dense)	(None, 27)	3,483

Total params: 2,124,795 (8.11 MB)

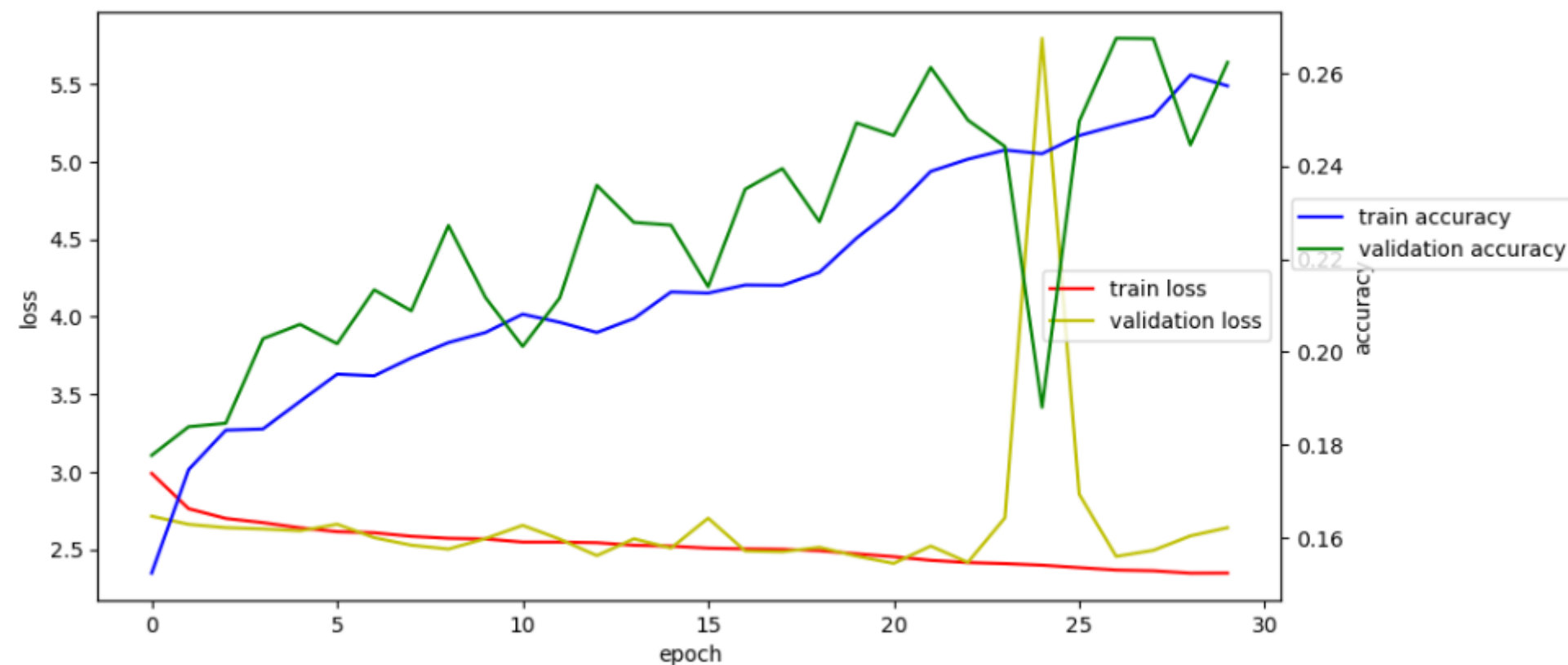
Trainable params: 2,124,571 (8.10 MB)

Non-trainable params: 224 (896.00 B)

8. 모델 평가

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

375/375 ----- 7s 20ms/step - accuracy: 0.2694 - loss: 2.4343
Test accuracy: 26.74%



학습시키기

Model: "sequential"

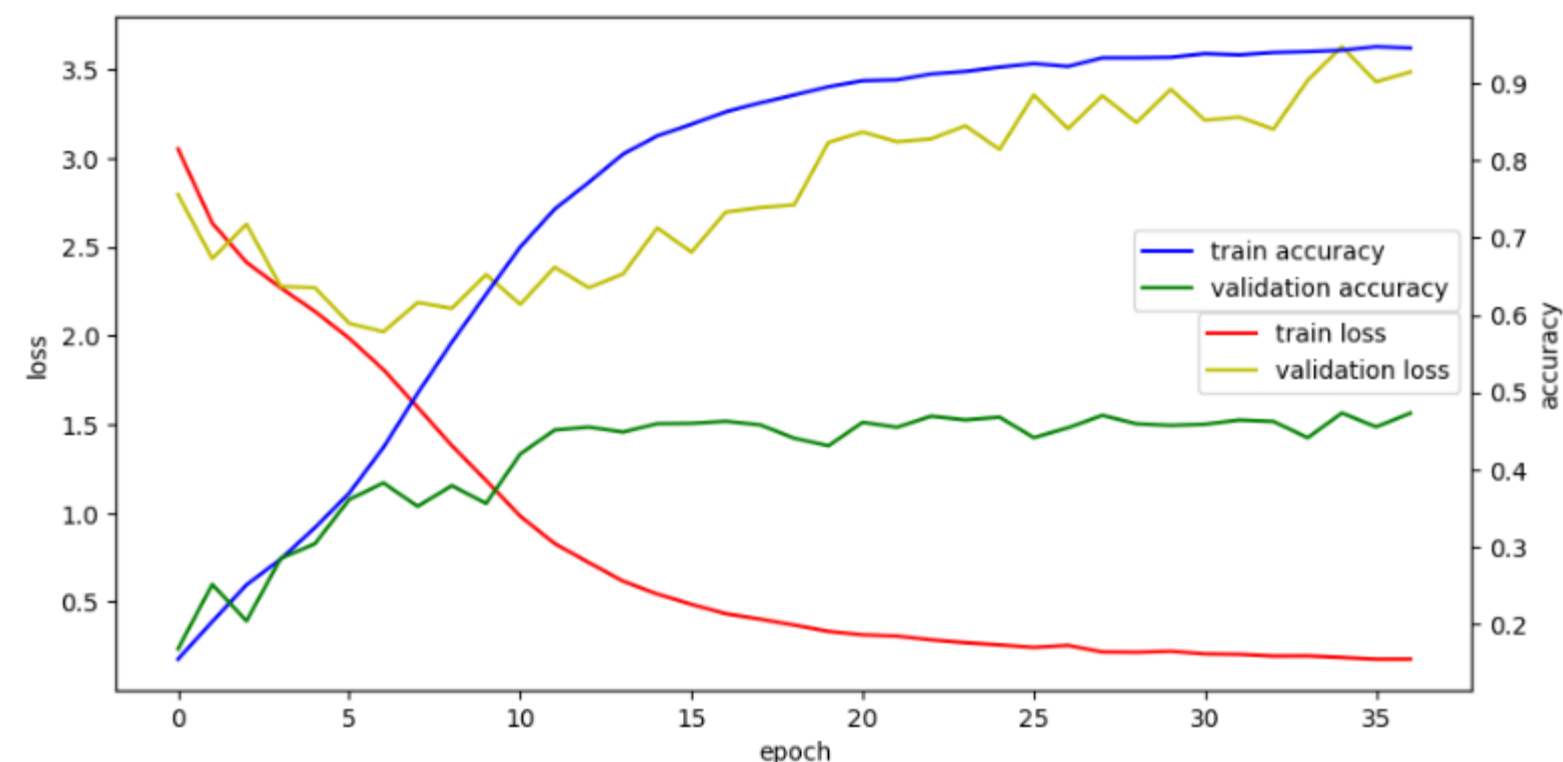
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 32)	896
conv2d_1 (Conv2D)	(None, 128, 128, 32)	9248
batch_normalization (Batch Normalization)	(None, 128, 128, 32)	128
max_pooling2d (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_2 (Conv2D)	(None, 64, 64, 64)	18496
conv2d_3 (Conv2D)	(None, 64, 64, 64)	36928
batch_normalization_1 (Batch Normalization)	(None, 64, 64, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 64)	0
conv2d_4 (Conv2D)	(None, 32, 32, 128)	73856
conv2d_5 (Conv2D)	(None, 32, 32, 128)	147584
batch_normalization_2 (Batch Normalization)	(None, 32, 32, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 128)	0

flatten (Flatten)	(None, 32768)	0
dense (Dense)	(None, 512)	16777728
batch_normalization_3 (Batch Normalization)	(None, 512)	2048
leaky_re_lu (LeakyReLU)	(None, 512)	0
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328
batch_normalization_4 (Batch Normalization)	(None, 256)	1024
leaky_re_lu_1 (LeakyReLU)	(None, 256)	0
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 27)	6939

 Total params: 17,206,971
 Trainable params: 17,204,987
 Non-trainable params: 1,984

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

375/375 [=====] - 6s 16ms/step - loss: 3.4846 - accuracy: 0.4729
 accuracy : 47.29%



같은 VGGNet을 사용해도 model을 어떻게 만드느냐에 따라 accuracy가 달라짐

데이터 준비

STEP 01

독립 변수

이미지 벡터 추출
VGG_vectors.npy

STEP 02

종속 변수

artist, genre, style 3종류

STEP 03

스케일 조정

SMOTE와
NearMiss 사용

STEP 04

데이터 분리




train 데이터
test 데이터

분류분석

DecisionTreeClassifier

◆ Classification Report:




	precision	recall	f1-score	support
Abstract_Expressionism	0.5375	0.5419	0.5397	3805
Action_painting	0.9597	0.9756	0.9675	3805
Analytical_Cubism	0.9659	0.9821	0.9739	3805
Art_Nouveau_Modern	0.4398	0.4202	0.4298	3805
Baroque	0.4895	0.4781	0.4837	3805
Color_Field_Painting	0.7928	0.8034	0.7981	3805
Contemporary_Realism	0.7781	0.8268	0.8017	3805
Cubism	0.5943	0.5921	0.5932	3805
Early_Renaissance	0.6797	0.6870	0.6833	3805
Expressionism	0.2819	0.2668	0.2741	3805
Fauvism	0.6853	0.7057	0.6953	3805
High_Renaissance	0.6618	0.6857	0.6736	3805
Impressionism	0.2487	0.2229	0.2351	3805
Mannerism_Late_Renaissance	0.6880	0.7224	0.7048	3804
Minimalism	0.8169	0.8163	0.8166	3805
Naive_Art_Primitivism	0.5331	0.5351	0.5341	3805
New_Realism	0.8258	0.8710	0.8478	3805
Northern_Renaissance	0.5502	0.5648	0.5574	3805
Pointillism	0.7926	0.8607	0.8252	3805
Pop_Art	0.6664	0.6762	0.6713	3805
Post_Impressionism	0.3120	0.2926	0.3020	3804
Realism	0.2329	0.2079	0.2197	3805
Rococo	0.6188	0.6420	0.6302	3805
Romanticism	0.3552	0.3372	0.3460	3805
Symbolism	0.4195	0.3997	0.4094	3805
Synthetic_Cubism	0.9374	0.9566	0.9469	3805
Ukiyo_e	0.7704	0.7777	0.7740	3805
accuracy			0.6240	102733
macro avg	0.6161	0.6240	0.6198	102733
weighted avg	0.6161	0.6240	0.6198	102733

 Precision: 0.6161
 Recall: 0.6240
 F1-score: 0.6198

MLPClassifier

◆ Classification Report:

	precision	recall	f1-score	support
Abstract_Expressionism	0.6690	0.6980	0.6832	3805
Action_painting	0.9880	0.9963	0.9921	3805
Analytical_Cubism	0.9961	0.9942	0.9951	3805
Art_Nouveau_Modern	0.5048	0.4297	0.4642	3805
Baroque	0.5135	0.5448	0.5287	3805
Color_Field_Painting	0.9092	0.8838	0.8963	3805
Contemporary_Realism	0.9350	0.9493	0.9421	3805
Cubism	0.7073	0.7451	0.7257	3805
Early_Renaissance	0.7920	0.8728	0.8305	3805
Expressionism	0.3102	0.2717	0.2897	3805
Fauvism	0.8202	0.8778	0.8480	3805
High_Renaissance	0.7903	0.7982	0.7942	3805
Impressionism	0.4596	0.3782	0.4149	3805
Mannerism_Late_Renaissance	0.7972	0.8247	0.8107	3804
Minimalism	0.9120	0.9204	0.9162	3805
Naive_Art_Primitivism	0.6341	0.6662	0.6498	3805
New_Realism	0.9583	0.9551	0.9567	3805
Northern_Renaissance	0.7130	0.6360	0.6723	3805
Pointillism	0.9315	0.9721	0.9514	3805
Pop_Art	0.7608	0.8770	0.8148	3805
Post_Impressionism	0.3550	0.3699	0.3623	3804
Realism	0.3741	0.2978	0.3316	3805
Rococo	0.6149	0.8394	0.7099	3805
Romanticism	0.5259	0.3204	0.3982	3805
Symbolism	0.4457	0.4715	0.4582	3805
Synthetic_Cubism	0.9834	0.9947	0.9890	3805
Ukiyo_e	0.8974	0.9472	0.9216	3805
accuracy			0.7234	102733
macro avg	0.7148	0.7234	0.7166	102733
weighted avg	0.7148	0.7234	0.7166	102733

 Precision: 0.7148
 Recall: 0.7234
 F1-score: 0.7166

앙상블 모형

RandomForestClassifier, XGBClassifier, LGBMClassifier

```
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)
```

```
# ✅ 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_model_hard.predict(test_X[0].reshape(1, -1))
```

```
array(['Northern_Renaissance'], dtype=object)
```

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

```
array(['Northern_Renaissance'], dtype=object)
```

```
test_y[0]
```

```
17
```

```
inverse_test_y = le.inverse_transform(test_y)
inverse_test_y[0]
```

```
'Northern_Renaissance'
```

```
model_measure(voting_model_hard)
```

```
'정확도:0.588, 정밀도:0.576, 재현율:0.588, f1_score:0.568'
```

```
model_measure(voting_model_soft)
```

```
'정확도:0.619, 정밀도:0.604, 재현율:0.619, f1_score:0.605'
```

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 : [Romanticism]

Wikipedia Description :

Romanticism (also known as the Romantic movement or Romantic era) was an artistic and intellectual movement that originated in Europe towards the end of the 18th century. The purpose of the movement was to advocate for the importance of subjectivity, imagination, and appreciation of nature in society and culture in response to the Age of Enlightenment and the Industrial Revolution.

Style 예측 결과



예측결과 : landscape

artist_name
Li Cheng
artist_info
Chinese ,919 - 967
painting_count
13 artworks

artist_name
Kanō Masanobu
artist_info
Japanese ,c.1434 - c.1530
painting_count
15 artworks

artist_name
Kanō Motonobu
artist_info
Japanese ,1476 - 1559
painting_count
35 artworks

artist_name
Joachim Patinir
artist_info
Flemish ,c.1480 - 1524
painting_count
24 artworks

artist_name
Jacopo Bassano
artist_info
Italian ,c.1510 - 1592
painting_count
47 artworks

artist_name
Kanō Eitoku
artist_info
Japanese ,1543 - 1590
painting_count
12 artworks

Genre 예측 결과



Abdullah Suriosubroto

- Born: 1878; Semarang, Indonesia
- Died: 1941; Yogyakarta, Indonesia
- Nationality: Indonesian
- Art Movement: Realism
- Genre: landscape
- Field: painting
- Wikipedia:
[id.wikipedia.org/wiki/Abdullah Suriosubroto](https://id.wikipedia.org/wiki/Abdullah_Suriosubroto)

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