Task 1: Artwork Classification with Convolutional-Recurrent Networks

In [ ]:
```
import os
import numpy as np
import pandas as pd
from tensorflow.keras.preprocessing.image import load_img, img_to_a
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, ConvLSTM2D, MaxPooling2
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import regularizers
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.models import Model
from scipy.spatial import distance
from sklearn.preprocessing import normalize
from PIL import Image
from scipy.spatial.distance import cdist # 创建函数来查找测试图像的最近邻
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.preprocessing import LabelEncoder
import types
from tensorflow.keras.models import load_model
from PIL import Image
Image.MAX_IMAGE_PIXELS = None  # 取消大小限制
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import IsolationForest
from tensorflow.keras.utils import Sequence
```

In [ ]:
```
# 加载不同的 CSV 文件
try:
    # 读取训练和验证数据
    style_train_data = pd.read_csv('wikiart_csv/Style_train.csv', h
    style_train_data.columns = ['image_path', 'class_index']

    style_val_data = pd.read_csv('wikiart_csv/Style_val.csv', heade
    style_val_data.columns = ['image_path', 'class_index']

    # 查看数据的前几行
    print("成功加载 CSV 文件！")
    print(style_train_data.head())
    print(style_val_data.head())

except FileNotFoundError as e:
    print(f"找不到 CSV 文件: {e}")
except pd.errors.EmptyDataError:
    print("CSV 文件为空！")
```

```
except pd.errors.ParserError:
    print("CSV 文件解析错误！")
```

```
成功加载 CSV 文件!
                                          image_path  class_index
0  Impressionism/edgar-degas_landscape-on-the-orn...           12
1        Realism/camille-corot_mantes-cathedral.jpg            21
2  Abstract_Expressionism/gene-davis_untitled-197...            0
3     Symbolism/kuzma-petrov-vodkin_in-the-1920.jpg            24
4  Impressionism/maurice-prendergast_paris-boulev...           12
                                          image_path  class_index
0  Impressionism/edgar-degas_dancers-on-set-1880.jpg           12
1     Impressionism/claude-monet_water-lilies-6.jpg            12
2  Impressionism/giovanni-boldini_a-guitar-player...           12
3  Impressionism/john-singer-sargent_at-torre-gal...           12
4  Impressionism/john-singer-sargent_artist-in-th...           12
```

In [ ]:
```
data_dir = 'wikiart'  # 数据目录
if not os.path.exists(data_dir):
    print(f"数据目录不存在: {data_dir}")
else:
    print(f"数据目录存在: {data_dir}")
```

```
数据目录存在: wikiart
```

In [ ]:
```
# 调整图像大小
img_size = (128, 128)
batch_size = 32 # 每个训练批次会包含 32 张图像
```

In [ ]:
```
class ImageLabelGenerator(tf.keras.utils.Sequence):
    def __init__(self, data_frame, data_dir, batch_size, img_size,
        self.data_frame = data_frame
        self.data_dir = data_dir
        self.batch_size = batch_size
        self.img_size = img_size
        self.shuffle = shuffle
        self.num_classes = num_classes
        self.indexes = np.arange(len(self.data_frame))

        # 初始化 StandardScaler，用于标准化每个图像
        self.scaler = StandardScaler()

    def __len__(self):
        # 返回每个 epoch 需要多少步
        return int(np.floor(len(self.data_frame) / self.batch_size)

    def __getitem__(self, index):
        max_attempts = 3  # 最多尝试 3 次
        attempts = 0
        damaged_images = []  # 用于记录损坏的图片路径

        while attempts < max_attempts:
            batch_indexes = self.indexes[index * self.batch_size :
            batch_images = []
            batch_labels = []

            for i in batch_indexes:
```

```
                img_path = os.path.join(self.data_dir, self.data_fr
                try:
                    img = load_img(img_path, target_size=self.img_s
                    img = img_to_array(img)  # 读取图像并转为数组

                    # 进行标准化
                    img = img - np.mean(img)  # 减去均值
                    img = img / np.std(img)  # 除以标准差

                    label = self.data_frame.iloc[i, 1]  # 获取标签
                    batch_images.append(img)
                    batch_labels.append(label)
                except Exception as e:
                    print(f"无法加载图片: {img_path}, 错误信息: {e}")
                    damaged_images.append(img_path)
                    continue  # 跳过无法加载的图像

            # 如果成功加载了足够的图片，则返回
            if len(batch_images) == self.batch_size:
                return np.array(batch_images), to_categorical(np.ar

            attempts += 1
            print(f"尝试加载失败，重新尝试第 {attempts} 次...")

        # 如果尝试多次后仍无法加载足够的有效图片，则抛出异常
        if len(batch_images) < self.batch_size:
            print(f"连续 {max_attempts} 次尝试后，批次不足，损坏的图片列表
            raise ValueError("连续几次尝试后仍无法加载有效数据，请检查数据路

    def on_epoch_end(self):
        if self.shuffle:
            np.random.shuffle(self.indexes)
            print("已打乱数据顺序")
        # 当数据不足以完成当前批次时，确保生成器可以重新迭代数据集
        self.indexes = np.arange(len(self.data_frame))  # 确保索引每
```

In [ ]:
```
# 设置类别数
num_classes_style = len(style_train_data['class_index'].unique())

# 使用生成器加载训练数据和验证数据
train_gen_style = ImageLabelGenerator(style_train_data, data_dir, b
val_gen_style = ImageLabelGenerator(style_val_data, data_dir, batch

# 查看数据
print("Style 类别数:", num_classes_style)
```

```
Style 类别数: 27
```

In [ ]:
```
# Check value for image pixel
# Notice that they already within 0-255
image_batch_original, labels_batch_original = next(iter(train_gen_s
first_image = image_batch_original[0]

print("The shape is", image_batch_original.shape,
      "\n i.e., There are",image_batch_original.shape[0],"images in
      "with size of",image_batch_original.shape[1],"by",image_batch
```

```
print("\n Check image",
      "\n Shape for label batch is", labels_batch_original.shape,
      "\n Minimum pixel is", np.min(first_image),
      "\n Maximum value for pixel is", np.max(first_image),
      "\n Mean value for pixel is", np.mean(first_image),
      "\n Standard derivative value for pixel is", np.std(first_ima
```

```
The shape is (32, 128, 128, 3)
 i.e., There are 32 images in a training batch, with size of 128 by
128

 Check image
 Shape for label batch is (32, 27)
 Minimum pixel is -1.65279
 Maximum value for pixel is 2.3076632
 Mean value for pixel is -1.9868216e-08
 Standard derivative value for pixel is 1.0
```

ConvLSTM

- 卷积部分 (Conv2D): 提取图像的空间特征。
- LSTM 部分: 图像的局部特征序列

In [ ]:
```
# 自定义生成器，扩展时间步维度
def add_time_step_dimension(generator):
    for images, labels in generator:
        #print(f"当前批次: {images.shape}, 标签: {labels.shape}")
        # 增加时间步维度: (batch_size, 128, 128, 3) -> (batch_size, 1,
        images = np.expand_dims(images, axis=1)  # 为了符合 ConvLSTM2
        yield images, labels

train_gen = add_time_step_dimension(train_gen_style)
val_gen = add_time_step_dimension(val_gen_style)

# 创建 ConvLSTM 模型
model = Sequential()

# ConvLSTM2D 层
time_steps = 1 #1  # 时间步
input_shape = (time_steps, img_size[0], img_size[1], 3)
model.add(ConvLSTM2D(filters=32,
                     kernel_size=(3, 3),
                     activation='relu',
                     input_shape=input_shape,
                     return_sequences=False))  # 不返回序列

# 批归一化
model.add(BatchNormalization())

# 卷积层
model.add(Conv2D(64, kernel_size=(5, 5), activation='relu', padding
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', padding
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Dropout 层
model.add(Dropout(0.3))

# 展平层
model.add(Flatten())  # 展平层，将多维的输入一维化
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.3))

# 输出层
model.add(Dense(num_classes_style, activation='softmax',
                kernel_regularizer=regularizers.L1L2(l1=0.001, l2=0

# 编译模型
model.compile(optimizer=Adam(learning_rate=0.0003),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/si
te-packages/keras/src/layers/rnn/rnn.py:205: UserWarning: Do not pas
s an `input_shape`/`input_dim` argument to a layer. When using Seque
ntial models, prefer using an `Input(shape)` object as the first lay
er in the model instead.
  super().__init__(**kwargs)
```

Model: "sequential_6"

Layer (type)	Output Shape	
conv_lstm2d_6 (ConvLSTM2D)	(None, 126, 126, 32)	
batch_normalization_6 (BatchNormalization)	(None, 126, 126, 32)	
conv2d_24 (Conv2D)	(None, 126, 126, 64)	
conv2d_25 (Conv2D)	(None, 126, 126, 64)	
max_pooling2d_12 (MaxPooling2D)	(None, 63, 63, 64)	
conv2d_26 (Conv2D)	(None, 61, 61, 128)	
conv2d_27 (Conv2D)	(None, 59, 59, 128)	
max_pooling2d_13 (MaxPooling2D)	(None, 29, 29, 128)	
dropout_12 (Dropout)	(None, 29, 29, 128)	
flatten_6 (Flatten)	(None, 107648)	
dense_12 (Dense)	(None, 256)	
dropout_13 (Dropout)	(None, 256)	
dense_13 (Dense)	(None, 27)	

Total params: 27,915,291 (106.49 MB)
Trainable params: 27,915,227 (106.49 MB)
Non-trainable params: 64 (256.00 B)

In [ ]:
```
train_sample_count = len(style_train_data)
val_sample_count = len(style_val_data)

print("训练集样本数:", train_sample_count)
print("验证集样本数:", val_sample_count)

# 计算每个 epoch 的步数
#steps_per_epoch = int(np.ceil(train_sample_count / batch_size))
#validation_steps = int(np.ceil(val_sample_count / batch_size))
steps_per_epoch = 50
validation_steps = 3


print("steps_per_epoch:", steps_per_epoch)
print("validation_steps:", validation_steps)
```

```
训练集样本数: 57025
验证集样本数: 24421
steps_per_epoch: 50
validation_steps: 3
```

In [ ]:
```
# 训练模型
history = model.fit(
```

```
    train_gen,
    epochs= 13,
    validation_data=val_gen,
    steps_per_epoch=steps_per_epoch,
    validation_steps=validation_steps
)
```

```
Epoch 1/13
50/50 ━━━━━━━━━━━━━━━━━━━━ 22s 385ms/step - accuracy: 0.1031 - loss:
4.2544 - val_accuracy: 0.5625 - val_loss: 3.7077
Epoch 2/13
50/50 ━━━━━━━━━━━━━━━━━━━━ 18s 364ms/step - accuracy: 0.1617 - loss:
3.3128 - val_accuracy: 0.7812 - val_loss: 3.6097
Epoch 3/13
50/50 ━━━━━━━━━━━━━━━━━━━━ 18s 365ms/step - accuracy: 0.1635 - loss:
3.2308 - val_accuracy: 0.9271 - val_loss: 3.4401
Epoch 4/13
50/50 ━━━━━━━━━━━━━━━━━━━━ 18s 367ms/step - accuracy: 0.1780 - loss:
3.1638 - val_accuracy: 0.8333 - val_loss: 3.4178
Epoch 5/13
50/50 ━━━━━━━━━━━━━━━━━━━━ 18s 363ms/step - accuracy: 0.1696 - loss:
3.0906 - val_accuracy: 0.9792 - val_loss: 3.1325
Epoch 6/13
50/50 ━━━━━━━━━━━━━━━━━━━━ 18s 358ms/step - accuracy: 0.2113 - loss:
2.9281 - val_accuracy: 0.5833 - val_loss: 3.1623
Epoch 7/13
50/50 ━━━━━━━━━━━━━━━━━━━━ 18s 363ms/step - accuracy: 0.1775 - loss:
2.9706 - val_accuracy: 0.7188 - val_loss: 2.8649
Epoch 8/13
50/50 ━━━━━━━━━━━━━━━━━━━━ 18s 366ms/step - accuracy: 0.2155 - loss:
2.8546 - val_accuracy: 0.8125 - val_loss: 2.7193
Epoch 9/13
50/50 ━━━━━━━━━━━━━━━━━━━━ 19s 374ms/step - accuracy: 0.1860 - loss:
2.8700 - val_accuracy: 0.5938 - val_loss: 2.7205
Epoch 10/13
50/50 ━━━━━━━━━━━━━━━━━━━━ 18s 360ms/step - accuracy: 0.1944 - loss:
2.8738 - val_accuracy: 0.4062 - val_loss: 2.9262
Epoch 11/13
50/50 ━━━━━━━━━━━━━━━━━━━━ 18s 358ms/step - accuracy: 0.2287 - loss:
2.7835 - val_accuracy: 0.6250 - val_loss: 2.1467
Epoch 12/13
50/50 ━━━━━━━━━━━━━━━━━━━━ 18s 361ms/step - accuracy: 0.2462 - loss:
2.7191 - val_accuracy: 0.8229 - val_loss: 1.7364
Epoch 13/13
50/50 ━━━━━━━━━━━━━━━━━━━━ 18s 358ms/step - accuracy: 0.2382 - loss:
2.6849 - val_accuracy: 0.6562 - val_loss: 2.1097
```

In [ ]:
```
test_loss, test_accuracy = model.evaluate(val_gen, steps=validation
print(f'验证集损失: {test_loss}, 验证集准确率: {test_accuracy}')
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 254ms/step - accuracy: 0.6185 - loss: 2.
0285
验证集损失: 2.0294737815856934, 验证集准确率: 0.6354166865348816
```

In [ ]:
```
# 确保预测时正确地收集 y_true 和 y_pred
y_true = []
y_pred = []
```

```
for batch_images, batch_labels in val_gen:
    # 只取 batch_labels 进行标签收集
    batch_labels = np.argmax(batch_labels, axis=1)  # 转化为类索引
    y_true.extend(batch_labels)  # 收集真实标签

    # 打印每个批次的预测结果，确保预测过程是有效的
    batch_predictions = model.predict(batch_images)
    batch_predictions = np.argmax(batch_predictions, axis=1)  # 获取
    y_pred.extend(batch_predictions)  # 收集预测标签

    # 打印每批次的预测和真实标签，调试用
    print(f"真实标签: {batch_labels}")
    print(f"预测标签: {batch_predictions}")

# 确保 y_true 和 y_pred 不为空
print(f"最终真实标签的数量: {len(y_true)}")
print(f"最终预测标签的数量: {len(y_pred)}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 271ms/step
真实标签: [12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 1
2 12 12 12 12
 12 12 12 12 12 12 12 12]
预测标签: [ 4 12 23 12 12 21 12 21 12 12 12 12 22 12 12 12 23 12 12 1
2 12 12 12 12
  4 12 12 12 24 12 12 12]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
真实标签: [12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 1
2 12 12 12 12
 12 12 12 12 12 12 12 12]
预测标签: [12 23 12 22 21 12 20 12 21 12 12 23 21 12 12 21 21 12  0 2
1 21 12 12  9
 12 21 21 20  9 21 12 21]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
真实标签: [12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 1
2 12 12 12 12
 12 12 12 12 12 12 12 12]
预测标签: [23 12 12 23 12 12 21 12 12 21 12 12 12 12 12 12 12 12 12 1
2 12  0 12 12
 12 21 20 12 24 12 12 20]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
真实标签: [12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 1
2 12 12 12 12
 12 12 12 12 12 12 12 12]
预测标签: [23 12 12 12 12 21 20 21 12 12  4 21 12 12 21 21 12 23 12 1
2 21 12 23 12
 12 12  4 21 12  4 12  4]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
真实标签: [12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 1
2 12 12 12 12
 12 12 12 12 12 12 12 12]
预测标签: [23 12 12 12 12 12 12 12 12 12 12 12 12 20 21 21 12 12 12 2
0  0 12 12 12
 12 12 12 12 21 12 12 12]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
真实标签: [12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 1
```

```
预测标签: [21 21  3  9 12 23  0 12 20 21 21 21 21  4 24 21 21 12 24
9 21 12 23 12
 12 12 21  0  0  3 21  4]
最终真实标签的数量: 22944
最终预测标签的数量: 22944
```

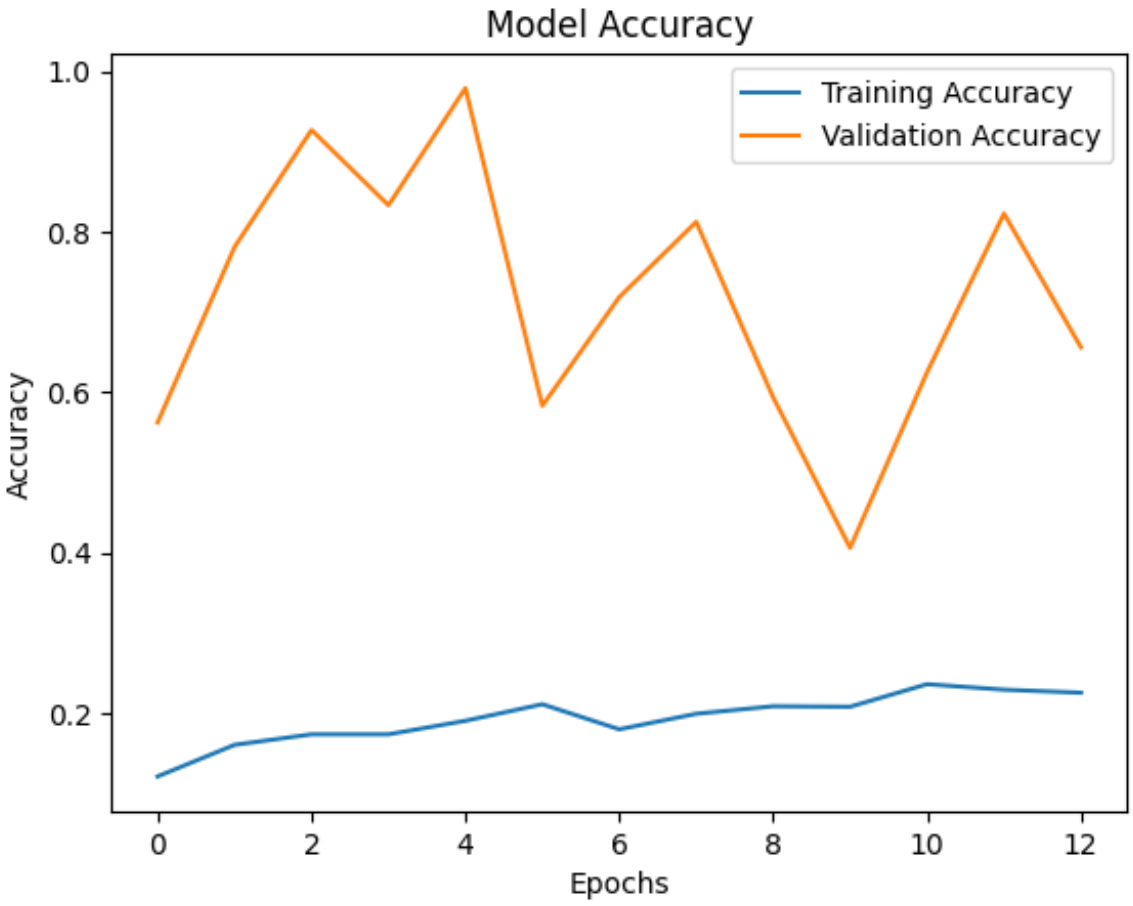
In [ ]:
```python
# 计算混淆矩阵
cm = confusion_matrix(y_true, y_pred)
# accuracy = (TP + TN) / (P + N)
accuracy = cm.diagonal()/cm.sum(axis=1)
# 打印混淆矩阵
print("混淆矩阵:")
print(cm)
print(accuracy)
```

```
混淆矩阵:
[[ 202    0    0   56   14   61    0    0    0  127    0    0   74
0
     2    1    0   15    0    0   20  193    0   24   45    0    0]
 [   2    0    0    2    1    3    0    0    0    5    0    0    4
0
     0    0    0    8    0    0    0    4    0    0    0    0    0]
 [   0    0    0    0    1    1    0    0    0    5    0    0   18
0
     0    0    0    0    0    0    1    6    0    1    0    0    0]
 [  83    0    0  199   31    5    0    2    0  161    0    0  331
0
     0    0    0   13    0    0   44  314    2   78   37    0    0]
 [   5    0    0   30  567    0    0    0    0   17    0    0  124
0
     0    0    0    5    0    1    3  312   16  186    6    0    0]
 [ 166    0    0   16    3  177    0    2    0   13    0    0   19
0
     8    0    0    1    0    0    5   46    0    7   21    0    0]
 [  10    0    0    3   10   13    0    0    0   11    0    0   36
0
     0    1    0    0    0    0    9   37    0    8    6    0    0]
 [  98    0    0   30   43    8    0    0    0  144    0    0  132
0
     0    0    0    3    0    0   28  135    0   14   35    0    0]
 [   0    0    0   11   68    1    0    0    0   50    0    0  159
0
     0    0    0    7    0    0    6   68    0   41    6    0    0]
 [ 160    0    0  106  149   22    0    0    0  279    0    0  507
0
     0    0    0   22    0    0   64  512    5  127   67    0    0]
 [  60    0    0    7    4   17    0    0    0   52    0    0   65
0
     0    0    0    0    0    0   26   23    0    4   22    0    0]
 [   1    0    0    9  107    1    0    0    0   24    0    0   96
0
     0    0    0    3    0    0    6  113    1   40    1    0    0]
 [  52    0    0    8  108    4    0    0    0   67    0    0 1462
0
     0    0    0    0    0    0  102  439    3  152   49    0    0]
 [   3    0    0    1  156    0    0    1    0   10    0    0   64
```

```
0
     0    0    0    4    0    0    0  103    6   31    4    0    0]
 [  87    0    0   21   14   63    0    0    0   17    0    0   23
0
     9    0    0    4    0    0   10  112    1   24   16    0    0]
 [  80    0    0   38   33    9    0    0    0  109    0    0  170
0
     0    0    0    3    0    0   46  167    0   36   25    0    0]
 [   5    0    0    0    9    0    0    0    0    3    0    0   31
0
     0    0    0    0    0    0    3   29    0   11    3    0    0]
 [   6    0    0   32  149    0    0    0    0   63    0    0  158
0
     0    0    0   39    0    0    6  221    1   79   11    0    0]
 [   4    0    0    3    0    1    0    0    0   10    0    0   85
0
     0    0    0    0    0    0   20    7    0    6   17    0    0]
 [ 139    0    0   40   11   44    0    1    0   47    0    0   37
0
     0    0    0    3    0    0   11   74    0   16   21    0    0]
 [ 107    0    0   26   76    6    0    0    0  144    0    0  946
0
     0    2    0    7    0    0  151  344    1   74   51    0    0]
 [  31    0    0   34  343    2    0    0    0   58    0    0  844
0
     0    0    0   12    0    0   78 1418   19  346   34    0    0]
 [   0    0    0    6  198    0    0    0    0    3    0    0   75
0
     0    0    0    2    0    0    1  153   44  144    0    0    0]
 [  14    0    0   30  292    2    0    0    0   24    0    0  456
0
     0    0    0   10    0    0   18  679   24  527   29    0    0]
 [  59    0    0   29   82   31    0    1    0   58    0    0  439
0
     1    0    0   10    0    1   54  280    4  152  157    0    0]
 [   2    0    0    1   14    0    0    0    0    3    0    0   17
0
     0    0    0    0    0    0    3   16    0    5    3    0    0]
 [  12    0    0   50    3    0    0    0    0   71    0    0   96
0
     0    0    0    0    0    0   13   82    0   10   13    0    0]]
[0.24220624 0.         0.         0.15307692 0.44575472 0.36570248
 0.         0.         0.         0.13811881 0.         0.
 0.59771055 0.         0.02244389 0.         0.         0.05098039
 0.         0.         0.07803618 0.44050947 0.07028754 0.25035629
 0.11561119 0.         0.        ]
```

In [ ]:
```
# 绘制准确率曲线
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accurac
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
# 绘制损失曲线
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [ ]:
```
class_indices = {v: k for k, v in enumerate(style_val_data['class_i
class_names = list(class_indices.keys())
```

In [ ]:
```
print(f"最大真实标签值: {max(y_true)}")
print(f"最大预测标签值: {max(y_pred)}")
```

最大真实标签值: 26
最大预测标签值: 24

In [ ]:
```
# 计算哪些图像被错误分类
incorrect_idx = [i for i in range(len(y_true)) if y_true[i] != y_pr

# 获取类别名称
if isinstance(val_gen, ImageLabelGenerator):
    class_names = list(val_gen.class_indices.keys())
else:
    # 手动指定类别名称
    # 设定正确的类别名称（假设有 27 个类别）
    class_names = [
    '0', '1', '2', '3', '4', '5', '6',
    '7', '8', '9', '10', '11', '12', '13',
    '14', '15', '16', '17', '18', '19', '20',
    '21', '22', '23', '24', '25', '26'
]
```

In [ ]:
```
# 3. 计算并打印分类报告
print(classification_report(y_true, y_pred, target_names=class_name
```

```
              precision    recall  f1-score   support

           0       0.15      0.24      0.18       834
           1       0.00      0.00      0.00        29
           2       0.00      0.00      0.00        33
           3       0.25      0.15      0.19      1300
           4       0.23      0.45      0.30      1272
           5       0.38      0.37      0.37       484
           6       0.00      0.00      0.00       144
           7       0.00      0.00      0.00       670
           8       0.00      0.00      0.00       417
           9       0.18      0.14      0.16      2020
          10       0.00      0.00      0.00       280
          11       0.00      0.00      0.00       402
          12       0.23      0.60      0.33      2446
          13       0.00      0.00      0.00       383
          14       0.45      0.02      0.04       401
          15       0.00      0.00      0.00       716
          16       0.00      0.00      0.00        94
          17       0.23      0.05      0.08       765
          18       0.00      0.00      0.00       153
          19       0.00      0.00      0.00       444
          20       0.21      0.08      0.11      1935
          21       0.24      0.44      0.31      3219
          22       0.35      0.07      0.12       626
          23       0.25      0.25      0.25      2105
          24       0.23      0.12      0.15      1358
          25       0.00      0.00      0.00        64
          26       0.00      0.00      0.00       350

    accuracy                           0.23     22944
   macro avg       0.12      0.11      0.10     22944
weighted avg       0.19      0.23      0.18     22944
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/si
te-packages/sklearn/metrics/_classification.py:1565: UndefinedMetric
Warning: Precision is ill-defined and being set to 0.0 in labels wit
h no predicted samples. Use `zero_division` parameter to control thi
s behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(resu
lt))
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/si
te-packages/sklearn/metrics/_classification.py:1565: UndefinedMetric
Warning: Precision is ill-defined and being set to 0.0 in labels wit
h no predicted samples. Use `zero_division` parameter to control thi
s behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(resu
lt))
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/si
te-packages/sklearn/metrics/_classification.py:1565: UndefinedMetric
Warning: Precision is ill-defined and being set to 0.0 in labels wit
h no predicted samples. Use `zero_division` parameter to control thi
s behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(resu
lt))
```

In [ ]:
```
# 查看前 5 个错误分类的图像
for idx in incorrect_idx[:5]:
    img_path = os.path.join(data_dir, style_val_data.iloc[idx]['ima
    img_name = style_val_data.iloc[idx]['image_path']
    img = load_img(img_path, target_size=img_size)
    plt.imshow(img)
    true_label = class_names[y_true[idx]]
    predicted_label = class_names[y_pred[idx]]
    plt.title(f"True: {true_label}, Predicted: {predicted_label}")
    plt.title(f"Image: {img_name}\nTrue: {true_label}, Predicted: {
    plt.axis('off')
    plt.show()
    plt.pause(0.1)  # 让显示过程有一个小的延迟，确保图像渲染
```

Image: Impressionism/giovanni-boldini_a-guitar-player-1873.jpg
True: 12, Predicted: 23

Image: Impressionism/william-merritt-chase_shinnecock-landscape-1.jpg
True: 12, Predicted: 21

Image: Impressionism/paul-gauguin_dahlias-in-a-copper-vase-1885.jpg
True: 12, Predicted: 21

Image: Impressionism/eugene-boudin_beach-scene.jpg
True: 12, Predicted: 22

In [ ]:
```
# model.save('ConvLSTM_model.h5')

# # 确保模型已编译
# model = load_model('ConvLSTM_model.h5')
# model.compile(optimizer='adam', loss='categorical_crossentropy',
```

In [ ]:
```
for layer in model.layers:
    print(layer.name)
```

```
conv_lstm2d_6
batch_normalization_6
conv2d_24
conv2d_25
max_pooling2d_12
conv2d_26
conv2d_27
max_pooling2d_13
dropout_12
flatten_6
dense_12
dropout_13
dense_13
```

In [ ]:
```
# #提取训练集和验证集的特征
# train_features = feature_extractor.predict(train_gen, batch_size=
# val_features = feature_extractor.predict(val_gen, batch_size=batc

# # 标准化特征
# train_features = normalize(train_features.reshape(train_features.
# val_features = normalize(val_features.reshape(val_features.shape[

# # 存储文件名以便后续可视化
# train_filenames = train_gen.filenames
# val_filenames = val_gen.filenames
```

In [ ]:
```
# # 3. 从训练集提取特征
# features = []
# labels = []

# for batch_images, batch_labels in train_gen:
#     # 提取特征
#     batch_features = feature_extractor.predict(batch_images)
#     batch_features = batch_features.reshape(batch_features.shape[
#     features.append(batch_features)
#     labels.append(batch_labels)

# # 转换为 numpy 数组
# features = np.vstack(features)
# labels = np.concatenate(labels)

# # 4. 使用 PCA 进行降维 (2D)
# pca = PCA(n_components=2)
# reduced_features = pca.fit_transform(features)

# # 5. 使用 Isolation Forest 进行异常检测
# clf = IsolationForest(n_estimators=100, contamination=0.1)  # 假设
# outliers = clf.fit_predict(reduced_features)  # 返回 1 表示正常，-1

# # 找到异常的索引
# outlier_indices = np.where(outliers == -1)[0]

# # 6. 可视化
```

```
# plt.scatter(reduced_features[:, 0], reduced_features[:, 1], c='bl
# plt.scatter(reduced_features[outlier_indices, 0], reduced_feature
# plt.legend()
# plt.title("PCA Visualization with Outliers")
# plt.show()

# 7. 打印异常画作的名字（或索引）
# print("异常画作的索引:", outlier_indices)
```

In [ ]:
```
# def pictures(test_idx):
#     test_feature = val_features[test_idx].reshape(1, -1)
#     my_distance = cdist(test_feature, train_features, metric='euc
#     nearest_neighbors_idx = np.argpartition(my_distance, 5)[:5]

#     plt.figure(figsize=(8, 8))
#     plt.subplot(2, 3, 1)
#     img = plt.imread(os.path.join(data_dir, style_val_data.iloc[t
#     plt.imshow(img)
#     plt.title(f"Test Image: {style_val_data.iloc[test_idx]['image
#     plt.axis('off')

#     for p in range(len(nearest_neighbors_idx)):
#         plt.subplot(2, 3, p + 2)
#         neighbor_image = plt.imread(os.path.join(data_dir, style_
#         plt.imshow(neighbor_image)
#         plt.title(f"Neighbor {p+1}")
#         plt.axis('off')

#     plt.show()
```

In [ ]:
```
# # 加载模型
# model = load_model('ConvLSTM_model.h5')

# # 打印模型的层，确保它已正确加载
# model.summary()

# # 检查所有层的名称
# for layer in model.layers:
#     print(f"Layer Name: {layer.name}")

# # 假设输入的形状是 (batch_size, time_steps, height, width, channels)
# dummy_input = np.random.random((1, 1, 128, 128, 3))  # 模拟输入数据

# # 激活模型进行前向传播，确保每层都有正确的输入
# model.predict(dummy_input)

# # 获取 'conv2d_1' 层并创建新模型
# conv2d_1_layer = model.get_layer('conv2d_1')  # 获取目标层
# model2 = Model(inputs=model.inputs, outputs=conv2d_1_layer.output

# # 查看提取模型的结构
# model2.summary()

# # 进行预测以获取 'conv2d_1' 层的输出
# output = model2.predict(dummy_input)
```

```
# print(output.shape)  # 打印输出形状
```

In [ ]:
```
# # 通过CNN提取的特征
# # 示例：画作特征数据的模拟
# features = np.random.rand(100, 128)  # 100个画作，每个画作有128个特征

# # 标准化特征数据
# scaler = StandardScaler()
# features_scaled = scaler.fit_transform(features)

# # PCA降维到2D
# pca = PCA(n_components=2)
# reduced_features = pca.fit_transform(features_scaled)

# # 使用K-means进行聚类
# kmeans = KMeans(n_clusters=3, random_state=42)
# kmeans.fit(reduced_features)
# labels = kmeans.labels_

# # 计算每个点到其所属簇中心的距离（距离越大，越可能是异常值）
# distances = np.linalg.norm(reduced_features - kmeans.cluster_cent

# # 假设距离大于某个阈值的点为异常点
# threshold = np.percentile(distances, 95)  # 取95%分位数作为阈值
# outliers = np.where(distances > threshold)[0]

# # 可视化
# plt.scatter(reduced_features[:, 0], reduced_features[:, 1], c=lab
# plt.scatter(reduced_features[outliers, 0], reduced_features[outli
# plt.legend()
# plt.title('PCA and K-means Outlier Detection')
# plt.show()

# print("异常点的索引:", outliers)
```