Art_Extract_Test1

March 29, 2025

[98]:

```
# === IMPORTS ===
import os
import math
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.utils import class_weight
from tensorflow.keras import layers, models, losses, optimizers, applications,␣
  ↪mixed_precision
```

[184]:

```
np.random.seed(42)
tf.random.set_seed(42)

# Configuration
BATCH_SIZE = 32
IMG_HEIGHT = 224
IMG_WIDTH = 224
NUM_EPOCHS = 7  # Increase epochs when in production
LEARNING_RATE = 0.001
TASK = "style"  # Options: "style", "artist", "genre"
```

[101]:

```
# Paths
BASE_DIR = "C:/Users/Ace/Gsoc_HumanAI/wikiart_csv"
WIKIART_DIR = "C:/Users/Ace/Gsoc_HumanAI/wikiart" # artwork images here
MODELS_DIR = "C:/Users/Ace/Gsoc_HumanAI"
TRAIN_DATA_PATH = f"{BASE_DIR}/{TASK}_train.csv"
VAL_DATA_PATH = f"{BASE_DIR}/{TASK}_val.csv"
CLASSES_PATH = f"{BASE_DIR}/{TASK}_class.txt"
```

[104]:

```
from sklearn.model_selection import train_test_split
from sklearn.utils.class_weight import compute_class_weight

def load_data(data_path, subset_size=1.0, random_state=42):
    """Load data from CSV file and apply stratified sampling."""
    df = pd.read_csv(data_path)
    df.columns = ['image_path', 'label']
    df['image_path'] = df['image_path'].apply(lambda x: os.path.
 ↪join(WIKIART_DIR, x))

    df = df[df['image_path'].apply(os.path.exists)]

    if subset_size < 1.0:
        df = df.groupby('label', group_keys=False).apply(
            lambda x: x.sample(frac=subset_size, random_state=random_state)
        )

    # Print sample paths for verification
    sample_paths = df['image_path'].sample(min(5, len(df))).tolist()
    for path in sample_paths:
        print(f"Checking if path exists: {path}")
        print(f"Exists: {os.path.exists(path)}")

    return df

def load_classes(classes_path):
    """Load class names from text file."""
    with open(classes_path, 'r') as f:
        classes = [line.strip() for line in f.readlines()]
    return classes
```

[106]:

```
def preprocess_data(train_df, val_df, classes):
    """Preprocess data for training."""
    is_numeric_labels = isinstance(train_df['label'].iloc[0], (int, np.integer))

    if is_numeric_labels:
        train_df['label_encoded'] = train_df['label']
        val_df['label_encoded'] = val_df['label']

        label_map = {i: class_name for i, class_name in enumerate(classes)}

        train_df['label_name'] = train_df['label'].map(label_map)
        val_df['label_name'] = val_df['label'].map(label_map)

        le = LabelEncoder()
        le.fit(classes)
    else:
        le = LabelEncoder()
        le.fit(classes)

        unknown_train_labels = set(train_df['label']) - set(classes)
        unknown_val_labels = set(val_df['label']) - set(classes)
        if unknown_train_labels:
            print(f"Warning: Found {len(unknown_train_labels)} unknown labels␣
↪in training data")
            print(f"Sample unknown labels: {list(unknown_train_labels)[:5]}")

            train_df = train_df[train_df['label'].isin(classes)]

        if unknown_val_labels:
            print(f"Warning: Found {len(unknown_val_labels)} unknown labels in␣
↪validation data")
            print(f"Sample unknown labels: {list(unknown_val_labels)[:5]}")

            val_df = val_df[val_df['label'].isin(classes)]

        train_df['label_encoded'] = le.transform(train_df['label'])
        val_df['label_encoded'] = le.transform(val_df['label'])

    class_weights = compute_class_weight(
        'balanced',
        classes=np.unique(train_df['label_encoded']),
        y=train_df['label_encoded']
    )
    class_weights_dict = {i: weight for i, weight in enumerate(class_weights)}

    return train_df, val_df, le, class_weights_dict
```

[108]:

```
def create_data_generators(df, batch_size, task):
    """Create a tf.data.Dataset compatible with EfficientNetB3."""
    def load_and_preprocess_image(path):
        image = tf.io.read_file(path)
        image = tf.image.decode_jpeg(image, channels=3)
        image = tf.image.resize(image, [IMG_HEIGHT, IMG_WIDTH])
        image = tf.keras.applications.efficientnet.preprocess_input(image)
        return image

    def augment_image(image, label):
        if task == "style":
            image = tf.cast(image, tf.uint8)
            image = tf.image.random_flip_left_right(image)

            image = tf.image.random_brightness(image, max_delta=0.1)
            image = tf.image.random_contrast(image, lower=0.9, upper=1.1)
            image = tf.image.random_saturation(image, lower=0.9, upper=1.1)

            crop_factor = tf.random.uniform([], 0.9, 1.0)
            crop_size = tf.cast(
                tf.cast([IMG_HEIGHT, IMG_WIDTH], tf.float32) * crop_factor,
                tf.int32
            )
            crop_size = tf.minimum(crop_size, [IMG_HEIGHT, IMG_WIDTH])
            image = tf.image.random_crop(image, [crop_size[0], crop_size[1], 3])
            image = tf.image.resize(image, [IMG_HEIGHT, IMG_WIDTH])

        elif task == "artist":
            image = tf.image.random_flip_left_right(image)
            image = tf.image.random_brightness(image, max_delta=0.2)
            image = tf.image.random_contrast(image, lower=0.8, upper=1.2)
            image = tf.image.random_saturation(image, lower=0.8, upper=1.2)

        elif task == "genre":
            image = tf.image.random_flip_left_right(image)

            image = tf.image.random_brightness(image, max_delta=0.1)
            image = tf.image.random_contrast(image, lower=0.9, upper=1.1)
            image = tf.image.random_saturation(image, lower=0.9, upper=1.1)

            if tf.random.uniform([], 0, 1) > 0.5:
                crop_factor = tf.random.uniform([], 0.95, 1.0)
                crop_size = tf.cast(
                    tf.cast([IMG_HEIGHT, IMG_WIDTH], tf.float32) * crop_factor,
                    tf.int32
                )
                image = tf.image.random_crop(image, [crop_size[0], crop_size[1], 3])
                image = tf.image.resize(image, [IMG_HEIGHT, IMG_WIDTH])

        return image, label

    paths = df['image_path'].values
    labels = df['label_encoded'].values

    dataset = tf.data.Dataset.from_tensor_slices((paths, labels))

    dataset = dataset.map(
        lambda path, label: (load_and_preprocess_image(path), label),
        num_parallel_calls=tf.data.AUTOTUNE
    )

    dataset = dataset.map(augment_image, num_parallel_calls=tf.data.AUTOTUNE)

    dataset = dataset.shuffle(buffer_size=10000)

    dataset = dataset.batch(batch_size).prefetch(tf.data.AUTOTUNE)
    return dataset
```

[132]:

```
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.layers import Input, Dense, Dropout, BatchNormalization,␣
 ↪GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping,␣
 ↪ModelCheckpoint

def focal_loss(gamma=2., alpha=0.25):
    def focal_loss_fixed(y_true, y_pred):
        epsilon = tf.keras.backend.epsilon()
        y_pred = tf.clip_by_value(y_pred, epsilon, 1. - epsilon)
        y_true = tf.one_hot(tf.cast(y_true, tf.int32), tf.shape(y_pred)[-1])
        alpha_t = y_true * alpha + (1 - y_true) * (1 - alpha)
        loss = -alpha_t * (y_true * tf.math.pow(1. - y_pred, gamma) * tf.math.
 ↪log(y_pred))
        return tf.reduce_sum(loss, axis=-1)
    return focal_loss_fixed

class CosineAnnealingWithRestarts(tf.keras.optimizers.schedules.
 ↪LearningRateSchedule):
    def __init__(self, initial_lr, T_max, eta_min=0):
        self.initial_lr = initial_lr
        self.T_max = T_max
        self.eta_min = eta_min
        self.t = 0

    def __call__(self, step):
        cos_inner = tf.math.pi * (self.t % self.T_max) / self.T_max
        lr = self.eta_min + (self.initial_lr - self.eta_min) * (1 + tf.math.
 ↪cos(cos_inner)) / 2
        self.t += 1
        return lr

def build_conv_recurrent_model(num_classes, task, img_height=224,␣
 ↪img_width=224):
    """Build improved models for art classification with fixes for artist task.
 ↪"""
    if task == "style" or task == "genre":
        base_model = tf.keras.applications.EfficientNetB2(
            weights='imagenet',
            include_top=False,
            input_shape=(img_height, img_width, 3)
        )
    else:
        base_model = tf.keras.applications.EfficientNetB2(
            weights='imagenet',
            include_top=False,
            input_shape=(img_height, img_width, 3)
        )

    if task == "style":
        for layer in base_model.layers[:-30]:
            layer.trainable = False
    elif task == "genre":
        for layer in base_model.layers[:-60]:
            layer.trainable = False
    else:
        for layer in base_model.layers[:-40]:
            layer.trainable = False

    inputs = Input(shape=(img_height, img_width, 3))
    x = tf.keras.applications.efficientnet.preprocess_input(inputs)
    x = base_model(x)


    x = GlobalAveragePooling2D()(x)

    if task == "artist":
        x = Dense(1536, activation='relu')(x)
        x = BatchNormalization()(x)
        x = Dropout(0.4)(x)

        shortcut =Dense(768)(x)

        x = Dense(768, activation='relu')(x)
        x = BatchNormalization()(x)
        x = Dropout(0.3)(x)

        se = Dense(128, activation='relu')(x)
        se = Dense(768, activation='sigmoid')(se)
        x = x * se

        x = x + shortcut

    elif task == "style":
        x = Dense(1024, activation='relu')(x)
        x = BatchNormalization()(x)
        x = Dropout(0.3)(x)

        x = Dense(512, activation='relu')(x)
        x = BatchNormalization()(x)
        x = Dropout(0.3)(x)

        x = Dense(256, activation='relu')(x)
        x = BatchNormalization()(x)
        x = Dropout(0.2)(x)

    elif task == "genre":
        x = Dense(1024, activation='relu')(x)
        x = BatchNormalization()(x)
        x = Dropout(0.3)(x)

        x = Dense(512, activation='relu')(x)
        x = BatchNormalization()(x)
        x = Dropout(0.25)(x)

        x = Dense(256, activation='relu')(x)
        x = BatchNormalization()(x)
        x = Dropout(0.2)(x)

    outputs = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=inputs, outputs=outputs)

    if task == "style":
        optimizer = Adam(learning_rate=CosineAnnealingWithRestarts(2e-3, 1000),␣
↪weight_decay=1e-5)
    elif task == "genre":
        initial_learning_rate = 5e-4
        lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
            initial_learning_rate,
            decay_steps=2000,
            decay_rate=0.95,
            staircase=True
        )
        optimizer = Adam(learning_rate=lr_schedule, weight_decay=1e-5)
    else:
        optimizer = Adam(learning_rate=CosineAnnealingWithRestarts(5e-4, 2000),␣
↪weight_decay=1e-5)

    model.compile(
        optimizer=optimizer,
        loss=focal_loss(gamma=2.0, alpha=0.25),
        metrics=['accuracy']
    )

    return model
```

[122]:

```
import seaborn as sns
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report, f1_score
from tensorflow.keras.utils import plot_model

def evaluate_model(model, val_dataset, le,TASK):
    """Evaluate the model and handle both integer and one-hot encoded labels."""
    # Create results folder
    os.makedirs('results', exist_ok=True)

    plot_model(
        model,
        to_file=f'results/model_architecture_{TASK}.png',
        show_shapes=True,
        show_layer_names=True,
        expand_nested=True
    )
    print(f"Model architecture saved as 'results/model_architecture_{TASK}.
 ↪png'")

    predictions = model.predict(val_dataset, steps=len(val_dataset), verbose=1)
    predicted_classes = np.argmax(predictions, axis=1)

    true_classes = []
    for _, labels in val_dataset:
        labels = labels.numpy()
        if labels.ndim == 2:
            true_classes.extend(np.argmax(labels, axis=1))
        else:
            true_classes.extend(labels)

    true_classes = np.array(true_classes)

    accuracy = accuracy_score(true_classes, predicted_classes)
    f1 = f1_score(true_classes, predicted_classes, average='weighted')

    print(f"\nAccuracy: {accuracy:.4f}")
    print(f"F1 Score: {f1:.4f}\n")

    conf_matrix = confusion_matrix(true_classes, predicted_classes)
    class_report = classification_report(true_classes, predicted_classes,␣
 ↪target_names=le.classes_)


    with open(f'results/classification_report_{TASK}.txt', 'w') as f:
        f.write(class_report)
    plt.figure(figsize=(14, 12))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
                xticklabels=le.classes_, yticklabels=le.classes_,
                cbar_kws={'shrink': 0.8}, linewidths=0.5, linecolor='gray',
                annot_kws={"size": 7},
                vmin=0, vmax=conf_matrix.max())

    plt.title(f'Confusion Matrix for {TASK}')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')


    plt.xticks(ticks=np.arange(len(le.classes_)) + 0.5, labels=le.classes_,␣
↪rotation=45, ha='right', fontsize=8)
    plt.yticks(ticks=np.arange(len(le.classes_)) + 0.5, labels=le.classes_,␣
↪rotation=0, va='center', fontsize=8)

    plt.subplots_adjust(left=0.3, bottom=0.2)

    plt.savefig(f'results/confusion_matrix_{TASK}.png')
    plt.show()


    f1_scores = f1_score(true_classes, predicted_classes, average=None)
    plt.figure(figsize=(16, 6))
    sns.barplot(x=le.classes_, y=f1_scores, palette='viridis')

    plt.title(f'Per-Class F1 Scores for {TASK}')
    plt.ylabel('F1 Score')

    plt.xticks(ticks=np.arange(len(le.classes_)), labels=le.classes_,␣
↪rotation=45, ha='right')

    plt.tight_layout()
    plt.savefig(f'results/f1_scores_{TASK}.png')
    plt.show()

    with open(f'results/class_accuracy_{TASK}.txt', 'w') as f:
        f.write("Class-wise Evaluation:\n")
        for i, class_name in enumerate(le.classes_):
            class_acc = conf_matrix[i, i] / conf_matrix[i].sum() if␣
↪conf_matrix[i].sum() > 0 else 0
            f.write(f"{class_name} - Accuracy: {class_acc:.4f}\n")

    prediction_confidence = np.max(predictions, axis=1)
    low_confidence_indices = np.where(prediction_confidence < 0.5)[0]
    misclassified_indices = np.where(predicted_classes != true_classes)[0]
    outlier_indices = np.union1d(low_confidence_indices, misclassified_indices)

    print(f"\nFound {len(outlier_indices)} potential outliers")

    return outlier_indices, predictions, true_classes, predicted_classes
```

[124]:

```
def visualize_training_history(history,TASK):
    """Visualize training history"""
    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)
    plt.plot(history['accuracy'], label='Train Accuracy')
    plt.plot(history['val_accuracy'], label='Validation Accuracy')
    plt.title('Model Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(loc='upper left')

    plt.subplot(1, 2, 2)
    plt.plot(history['loss'], label='Train Loss')
    plt.plot(history['val_loss'], label='Validation Loss')
    plt.title('Model Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(loc='upper left')

    plt.tight_layout()
    plt.savefig(f'training_history_{TASK}.png')
    plt.show()



def visualize_outliers(outlier_indices, val_dataset, predictions, true_classes,␣
 ↪predicted_classes, le, task, num_examples=5):
    """Visualize outlier examples"""

    results_dir = f'results/{task}'
    os.makedirs(results_dir, exist_ok=True)

    images = []
    for img_batch, _ in val_dataset:
        images.extend(img_batch.numpy())

    if len(outlier_indices) > num_examples:
        sample_indices = np.random.choice(outlier_indices, num_examples,␣
 ↪replace=False)
    else:
        sample_indices = outlier_indices

    plt.figure(figsize=(12, 12))
    for i, idx in enumerate(sample_indices):
        if idx < len(images):
            image = images[idx]

            if image.max() <= 1.0:
                image = (image * 255).astype('uint8')
            else:
                image = np.clip(image, 0, 255).astype('uint8')

            true_label = le.classes_[true_classes[idx]]
            pred_label = le.classes_[predicted_classes[idx]]
            confidence = predictions[idx][predicted_classes[idx]]

            plt.subplot(3, 2, i + 1)
            plt.imshow(image)
            plt.title(f"True: {true_label}\nPred: {pred_label}\nConf:␣
↪{confidence:.2f}", fontsize=10)
            plt.axis('off')

    plt.tight_layout()
    outlier_path = os.path.join(results_dir, f'outliers_{task}.png')
    plt.savefig(outlier_path)
    plt.show()
    print(f"Outliers saved to: {outlier_path}")
```

[126]:

```
# Data exploration function
def explore_dataset():
    """Explore the dataset structure"""
    for task in ["artist", "genre", "style"]:
        train_path = f"{BASE_DIR}/{task}_train.csv"
        val_path = f"{BASE_DIR}/{task}_val.csv"
        class_path = f"{BASE_DIR}/{task}_class.txt"

        print(f"\n{'='*40}")
        print(f"Exploring {task.upper()} dataset")
        print(f"{'='*40}")

        print(f"Train file exists: {os.path.exists(train_path)}")
        print(f"Val file exists: {os.path.exists(val_path)}")
        print(f"Class file exists: {os.path.exists(class_path)}")

        try:
            train_df = pd.read_csv(train_path)
            val_df = pd.read_csv(val_path)
```

```python
            print(f"\nTrain data shape: {train_df.shape}")
            print(f"Validation data shape: {val_df.shape}")

            print(f"\nTrain columns: {train_df.columns.tolist()}")

            print("\nSample train data (first 3 rows):")
            print(train_df.head(3))

            if len(train_df.columns) >= 2:
                label_col = train_df.iloc[:, 1]
                unique_labels = label_col.unique()
                print(f"\nNumber of unique labels in training data: {len(unique_labels)}")
                print(f"Sample labels: {unique_labels[:5].tolist()}")

            if len(train_df.columns) >= 1:
                img_col = train_df.iloc[:, 0]
                sample_paths = img_col.sample(min(3, len(img_col))).tolist()
                print("\nSample image paths:")
                for path in sample_paths:
                    print(f"  {path}")
                    full_path = os.path.join(WIKIART_DIR, path)
                    print(f"  Exists in wikiart folder: {os.path.exists(full_path)}")

        except Exception as e:
            print(f"Error exploring {task} dataset: {str(e)}")

print("Exploring dataset structure...")
explore_dataset()

print("\nStarting model training...")
```

```
Exploring dataset structure…

========================================
Exploring ARTIST dataset
========================================
Train file exists: True
Val file exists: True
Class file exists: True

Train data shape: (13345, 2)
Validation data shape: (5705, 2)

Train columns: ['Realism/vincent-van-gogh_pine-trees-in-the-fen-1884.jpg', '22']
```

```
Sample train data (first 3 rows):
  Realism/vincent-van-gogh_pine-trees-in-the-fen-1884.jpg  22
0  Baroque/rembrandt_the-angel-appearing-to-the-s…       20
1  Post_Impressionism/paul-cezanne_portrait-of-th…       16
2  Impressionism/pierre-auguste-renoir_young-girl…       17

Number of unique labels in training data: 23
Sample labels: [20, 16, 17, 9, 1]

Sample image paths:
  Romanticism/gustave-dore_paradise-lost-4.jpg
  Exists in wikiart folder: True
  Post_Impressionism/vincent-van-gogh_the-raising-of-lazarus-1890.jpg
  Exists in wikiart folder: True
  Impressionism/eugene-boudin_the-port-of-deauville-1.jpg
  Exists in wikiart folder: True

========================================
Exploring GENRE dataset
========================================
Train file exists: True
Val file exists: True
Class file exists: True

Train data shape: (45502, 2)
Validation data shape: (19491, 2)

Train columns: ['Northern_Renaissance/hieronymus-bosch_st-jacques-and-the-
magician-hermogenes.jpg', '7']

Sample train data (first 3 rows):
  Northern_Renaissance/hieronymus-bosch_st-jacques-and-the-magician-
hermogenes.jpg  \
0  Post_Impressionism/vincent-van-gogh_ears-of-wh…
1  Symbolism/theodor-severin-kittelsen_kvitebj-rn…
2  Expressionism/martiros-saryan_mother-of-the-ar…

   7
0  4
1  3
2  6

Number of unique labels in training data: 10
Sample labels: [4, 3, 6, 8, 0]

Sample image paths:
  Post_Impressionism/wassily-kandinsky_gabriele-munter-1905.jpg
  Exists in wikiart folder: True
  Impressionism/childe-hassam_clarissa.jpg
  Exists in wikiart folder: True
  Post_Impressionism/pyotr-konchalovsky_pine-tree-1921.jpg
  Exists in wikiart folder: True

========================================
Exploring STYLE dataset
========================================
Train file exists: True
Val file exists: True
Class file exists: True

Train data shape: (57024, 2)
Validation data shape: (24420, 2)

Train columns: ['Impressionism/edgar-degas_landscape-on-the-orne.jpg', '12']

Sample train data (first 3 rows):
  Impressionism/edgar-degas_landscape-on-the-orne.jpg  12
0         Realism/camille-corot_mantes-cathedral.jpg   21
1  Abstract_Expressionism/gene-davis_untitled-197…     0
2      Symbolism/kuzma-petrov-vodkin_in-the-1920.jpg   24

Number of unique labels in training data: 27
Sample labels: [21, 0, 24, 12, 7]

Sample image paths:
  Art_Nouveau_Modern/felix-vallotton_portrait-of-belgian-symbolist-poet-max-
elskamp-1898.jpg
  Exists in wikiart folder: True
  Impressionism/edmund-charles-tarbell_the-bath-1893.jpg
  Exists in wikiart folder: True
  Impressionism/pierre-auguste-renoir_rocks-with-shrimp-fishermen-1892.jpg
  Exists in wikiart folder: True

Starting model training…
```

[186]:
```
def train_model(task, subset_size=1.0):
    """Train the improved model with enhanced training strategy."""
    train_df = load_data(f"{BASE_DIR}/{task}_train.csv",␣
  ↪subset_size=subset_size)
    val_df = load_data(f"{BASE_DIR}/{task}_val.csv", subset_size=subset_size)
    classes = load_classes(f"{BASE_DIR}/{task}_class.txt")

    train_df, val_df, label_encoder, class_weights_dict =␣
  ↪preprocess_data(train_df, val_df, classes)
```

```
train_dataset = create_data_generators(train_df, BATCH_SIZE, task)
val_dataset = create_data_generators(val_df, BATCH_SIZE, task)

model = build_conv_recurrent_model(len(classes), task)
print(model.summary())

# Callbacks for better training
checkpoint = ModelCheckpoint(
    f'{MODELS_DIR}/models/best_model_{task}.keras',
    monitor='val_accuracy',
    save_best_only=True,
    mode='max',
    verbose=1
)

early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=12,
    restore_best_weights=True,
    verbose=1
)

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=4,
    min_lr=1e-7,
    verbose=1
)

model.compile(
    optimizer=Adam(learning_rate=LEARNING_RATE, weight_decay=1e-6),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

print("Phase 1: Initial training with mostly frozen base model...")
history1 = model.fit(
    train_dataset,
    epochs=NUM_EPOCHS,
    validation_data=val_dataset,
    callbacks=[checkpoint, early_stopping, reduce_lr],
    class_weight=class_weights_dict
)

print("Phase 2: Fine-tuning with more layers unfrozen...")
```

```
    base_model = model.layers[1]


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

    model.compile(
        optimizer=Adam(learning_rate=LEARNING_RATE/10, weight_decay=1e-6),
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )

    history2 = model.fit(
        train_dataset,
        epochs=10,
        initial_epoch=history1.epoch[-1] + 1,
        validation_data=val_dataset,
        callbacks=[checkpoint, early_stopping, reduce_lr],
        class_weight=class_weights_dict
    )

    # Evaluate the model
    print("Evaluating model...")
    outlier_indices, predictions, true_classes, predicted_classes =␣
 ↪evaluate_model(
        model, val_dataset, label_encoder,task
    )

    print("Visualizing results...")
    combined_history = {}
    for k in history1.history.keys():
        combined_history[k] = history1.history[k] + history2.history[k]

    visualize_training_history(combined_history,task)
    visualize_outliers(
        outlier_indices, val_dataset, predictions,
        true_classes, predicted_classes, label_encoder,task
    )

    return model, combined_history
```

```
[ ]: def train_all_models_improved():
         """Train improved models for all tasks with stratified sampling."""
         results = {}
         subset_sizes = {
             "artist": 1.0,  # Use 100% of the dataset for Artist
             "genre": 0.5,   # Use 50% of the dataset for Genre
```

```
        "style": 0.3    # Use 30% of the dataset for Style
    }

    for task in ["style","genre","artist"]:
        print(f"\n\n{'='*60}")
        print(f"Training improved model for {task.upper()}")
        print(f"{'='*60}\n")

        model, history = train_model(task, subset_size=subset_sizes[task])
        results[task] = (model, history)

    return results


if __name__ == "__main__":
    # Training all models
    all_results = train_all_models_improved()
```

[164]:

```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
```

[166]:

```
def load_trained_model(task):
    """Load a trained model for a specific task"""
    model_path = f'{MODELS_DIR}/models/best_model_{task}.keras'
    if not os.path.exists(model_path):
        print(f"Error: Model for {task} not found at {model_path}")
        return None

    try:
        model = load_model(model_path)
        print(f"Successfully loaded {task} model")
        return model
    except Exception as e:
        print(f"Error loading {task} model: {str(e)}")
        return None

def load_class_labels(task):
    """Load class labels for a specific task"""
    classes_path = f'{BASE_DIR}/{task}_class.txt'
    try:
        with open(classes_path, 'r') as f:
            classes = [line.strip() for line in f.readlines()]
        print(f"Loaded {len(classes)} {task} classes")
        return classes
    except Exception as e:
        print(f"Error loading {task} classes: {str(e)}")
        return None
```

[168]:

```
def preprocess_image(img_path):
    """Preprocess an image for model prediction"""
    try:
        if not os.path.exists(img_path):
            print(f"Error: Image not found at {img_path}")
            return None

        # Load and preprocess the image
        img = image.load_img(img_path, target_size=(224, 224))
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array = img_array / 255.0

        return img_array, img
    except Exception as e:
        print(f"Error preprocessing image: {str(e)}")
        return None, None
```

[170]:

```
def predict_artwork(img_path, tasks=None):
    """
    Predict artist, style, and genre for a given artwork image

    Parameters:
    img_path (str): Path to the artwork image
    tasks (list): List of tasks to perform, default ["artist", "genre", "style"]

    Returns:
    dict: Dictionary with predictions for each task
    """
    if tasks is None:
        tasks = ["artist", "genre", "style"]

    img_array, original_img = preprocess_image(img_path)
    if img_array is None:
        return None

    results = {}

    models = {task: load_trained_model(task) for task in tasks}
    class_labels = {task: load_class_labels(task) for task in tasks}

    for task in tasks:
        print(f"\nPredicting {task}...")

        model = models.get(task)
        classes = class_labels.get(task)
        if model is None or classes is None:
            results[task] = {"error": f"Could not load model or classes for {task}"}
            continue

        try:
            img_tensor = tf.convert_to_tensor(img_array)
            img_tensor = tf.ensure_shape(img_tensor, (1, 224, 224, 3))  # Example shape (adjust to your model)

            @tf.function(reduce_retracing=True)
            def predict_step(input_tensor):
                return model(input_tensor)

            predictions = predict_step(img_tensor)

            top_indices = tf.argsort(predictions[0], direction="DESCENDING")[:3]
            top_predictions = [(classes[i.numpy()], float(predictions[0][i].numpy())) for i in top_indices]

            results[task] = {
                "top_predictions": top_predictions,
                "prediction": classes[int(tf.argmax(predictions[0]))],
                "confidence": float(tf.reduce_max(predictions[0]))
            }

            print(f"Top {task} predictions:")
            for class_name, prob in top_predictions:
                print(f"  {class_name}: {prob:.4f}")

        except Exception as e:
            print(f"Error making prediction for {task}: {str(e)}")
            results[task] = {"error": str(e)}

    visualize_prediction_results(img_path, original_img, results)

    return results
```

[172]:

```
def visualize_prediction_results(img_path, original_img, results):
    """Visualize the prediction results"""
    plt.figure(figsize=(12, 8))

    plt.subplot(1, 2, 1)
    plt.imshow(original_img)
    plt.title(f"Artwork: {os.path.basename(img_path)}")
    plt.axis('off')
    plt.subplot(1, 2, 2)
    plt.axis('off')

    result_text = "Prediction Results:\n\n"

    for task in results:
        result_text += f"{task.capitalize()}:\n"

        if "error" in results[task]:
            result_text += f"  Error: {results[task]['error']}\n"
        else:
            for i, (class_name, prob) in enumerate(results[task]["top_predictions"]):
                result_text += f"  {i+1}. {class_name}: {prob:.2%}\n"

        result_text += "\n"

    plt.text(0.1, 0.5, result_text, fontsize=12, verticalalignment='center')

    plt.tight_layout()
    plt.savefig('artwork_prediction.png')
    plt.show()
```

[174]:

```
def batch_predict_artworks(folder_path, tasks=None):
    """
    Predict artist, style, and genre for all artwork images in a folder

    Parameters:
    folder_path (str): Path to the folder containing artwork images
    tasks (list): List of tasks to perform, default ["artist", "genre", "style"]
    """
    if tasks is None:
        tasks = ["artist", "genre", "style"]

    if not os.path.exists(folder_path):
        print(f"Error: Folder not found at {folder_path}")
        return

    image_extensions = ['.jpg', '.jpeg', '.png']
    image_files = [f for f in os.listdir(folder_path)
                  if os.path.isfile(os.path.join(folder_path, f)) and
                  any(f.lower().endswith(ext) for ext in image_extensions)]

    if not image_files:
        print(f"No image files found in {folder_path}")
        return
    print(f"Found {len(image_files)} image files. Starting batch prediction...")

    models = {}
    class_labels = {}

    for task in tasks:
        models[task] = load_trained_model(task)
        class_labels[task] = load_class_labels(task)

        if models[task] is None or class_labels[task] is None:
            print(f"Warning: Could not load model or classes for {task}")

    all_results = {}
    for img_file in image_files:
        img_path = os.path.join(folder_path, img_file)
        print(f"\nProcessing {img_file}...")

        img_array, _ = preprocess_image(img_path)
        if img_array is None:
            all_results[img_file] = {"error": "Failed to preprocess image"}
            continue

        img_results = {}
        for task in tasks:
            if models[task] is None or class_labels[task] is None:
                img_results[task] = {"error": f"Model or classes not available␣
↪for {task}"}
                continue

            try:
                predictions = models[task].predict(img_array)

                top_indices = np.argsort(predictions[0])[-3:][::-1]
                top_predictions = [(class_labels[task][i],␣
↪float(predictions[0][i])) for i in top_indices]

                img_results[task] = {
                    "top_predictions": top_predictions,
                    "prediction": class_labels[task][np.argmax(predictions)],
                    "confidence": float(np.max(predictions))
                }

            except Exception as e:
                print(f"Error making {task} prediction for {img_file}:␣
↪{str(e)}")
                img_results[task] = {"error": str(e)}
        all_results[img_file] = img_results

    export_results_to_csv(all_results, folder_path)

    print(f"\nCompleted batch prediction for {len(image_files)} images")
    return all_results
```

[176]:

```
def export_results_to_csv(all_results, folder_path):
    """Export batch prediction results to CSV"""
    import pandas as pd

    rows = []
    for img_file, img_results in all_results.items():
        row = {'image': img_file}

        for task in img_results:
            if "error" in img_results[task]:
                row[f'{task}_prediction'] = "ERROR"
                row[f'{task}_confidence'] = 0.0
            else:
                row[f'{task}_prediction'] = img_results[task]["prediction"]
                row[f'{task}_confidence'] = img_results[task]["confidence"]

                # Add top 3 predictions
                for i, (class_name, prob) in␣
 ↪enumerate(img_results[task]["top_predictions"]):
                    row[f'{task}_top{i+1}'] = class_name
                    row[f'{task}_top{i+1}_confidence'] = prob

        rows.append(row)

    df = pd.DataFrame(rows)
    csv_path = os.path.join(folder_path, 'artwork_predictions.csv')
    df.to_csv(csv_path, index=False)
    print(f"Results exported to {csv_path}")

def analyze_single_image(img_path):
    """
    Analyze a single artwork image for artist, style, and genre

    Parameters:
    img_path (str): Path to the artwork image
    """
    print(f"Analyzing artwork: {img_path}")
    results = predict_artwork(img_path)

    if results:
        print("\nSummary of predictions:")
        for task, task_results in results.items():
            if "error" in task_results:
                print(f"  {task.capitalize()}: Error - {task_results['error']}")
            else:
                print(f"  {task.capitalize()}: {task_results['prediction']}␣
 ↪({task_results['confidence']:.2%})")

    return results
```

[ ]:

```
if __name__ == "__main__":
    # Single image prediction
    print("\n===== Single Image Prediction =====")
    analyze_single_image(test_image_path)

    # Batch prediction example - uncomment to use
    # test_folder_path = "./test_images"  # Change this to your test folder path
    # print("\n===== Batch Prediction =====")
    # batch_results = batch_predict_artworks(test_folder_path)
```

[ ]:

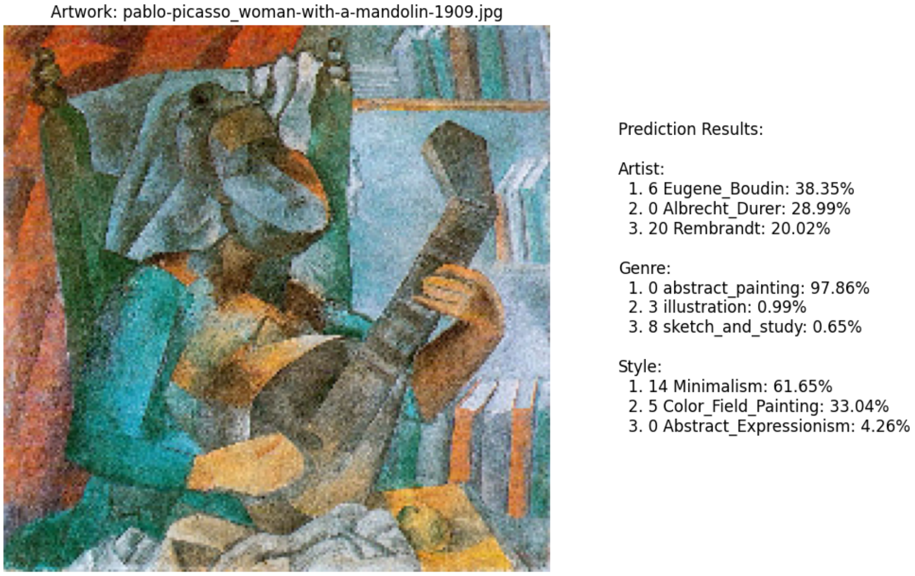
```

```

[150]:

```
# Example usage
if __name__ == "__main__":
[redacted]

    # Single image prediction
    print("\n===== Single Image Prediction =====")
    analyze_single_image(test_image_path)

    # Batch prediction example - uncomment to use
    # test_folder_path = "./test_images"  # Change this to your test folder path
    # print("\n===== Batch Prediction =====")
    # batch_results = batch_predict_artworks(test_folder_path)

[redacted]
```

```
./test_artwork.jpg C:/Users/Ace/Gsoc_HumanAI/wikiart/Analytical_Cubism/pablo-
picasso_woman-with-a-mandolin-1909.jpg


===== Single Image Prediction =====
Analyzing artwork: C:/Users/Ace/Gsoc_HumanAI/wikiart/Analytical_Cubism/pablo-
picasso_woman-with-a-mandolin-1909.jpg
Successfully loaded artist model
Successfully loaded genre model
Successfully loaded style model
Loaded 23 artist classes
Loaded 10 genre classes
Loaded 27 style classes

Predicting artist…
Top artist predictions:
  6 Eugene_Boudin: 0.3835
  0 Albrecht_Durer: 0.2899
  20 Rembrandt: 0.2002

Predicting genre…
WARNING:tensorflow:5 out of the last 5 calls to <function
predict_artwork.<locals>.predict_step at 0x0000029D2FD958A0> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True option that can avoid unnecessary
retracing. For (3), please refer to
```

```
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
Top genre predictions:
  0 abstract_painting: 0.9786
  3 illustration: 0.0099
  8 sketch_and_study: 0.0065

Predicting style…
WARNING:tensorflow:6 out of the last 6 calls to <function
predict_artwork.<locals>.predict_step at 0x0000029D3489C180> triggered
tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors.
For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True option that can avoid unnecessary
retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
Top style predictions:
  14 Minimalism: 0.6165
  5 Color_Field_Painting: 0.3304
  0 Abstract_Expressionism: 0.0426
```



```
Artwork: pablo-picasso_woman-with-a-mandolin-1909.jpg

Prediction Results:

Artist:
  1. 6 Eugene_Boudin: 38.35%
  2. 0 Albrecht_Durer: 28.99%
  3. 20 Rembrandt: 20.02%

Genre:
  1. 0 abstract_painting: 97.86%
  2. 3 illustration: 0.99%
  3. 8 sketch_and_study: 0.65%

Style:
  1. 14 Minimalism: 61.65%
  2. 5 Color_Field_Painting: 33.04%
  3. 0 Abstract_Expressionism: 4.26%
```

```
Summary of predictions:
  Artist: 6 Eugene_Boudin (38.35%)
  Genre: 0 abstract_painting (97.86%)
  Style: 14 Minimalism (61.65%)
```

```
[ ]:
```

```python
        "style": 0.3    # Use 30% of the dataset for Style
    }

    for task in ["style","genre","artist"]:
        print(f"\n\n{'='*60}")
        print(f"Training improved model for {task.upper()}")
        print(f"{'='*60}\n")

        model, history = train_model(task, subset_size=subset_sizes[task])
        results[task] = (model, history)

    return results


if __name__ == "__main__":
    # Training all models
    all_results = train_all_models_improved()
```

```
============================================================
Training improved model for STYLE
============================================================


C:\Users\Ace\AppData\Local\Temp\ipykernel_37128\2906373448.py:14:
DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns.
This behavior is deprecated, and in a future version of pandas the grouping
columns will be excluded from the operation. Either pass `include_groups=False`
to exclude the groupings or explicitly select the grouping columns after groupby
to silence this warning.
  df = df.groupby('label', group_keys=False).apply(

Checking if path exists:
C:/Users/Ace/Gsoc_HumanAI/wikiart\Baroque/rembrandt_young-woman-with-a-broom.jpg
Exists: True
Checking if path exists: C:/Users/Ace/Gsoc_HumanAI/wikiart\Romanticism/cornelis-
springer_coming-out-of-church.jpg
Exists: True
Checking if path exists: C:/Users/Ace/Gsoc_HumanAI/wikiart\Realism/salvador-
dali_reclining-girl-in-sheep.jpg
Exists: True
Checking if path exists: C:/Users/Ace/Gsoc_HumanAI/wikiart\Expressionism/laszlo-
moholy-nagy_self-portrait-1919.jpg
Exists: True
Checking if path exists: C:/Users/Ace/Gsoc_HumanAI/wikiart\Realism/ivan-
shishkin_forest-into-the-frost.jpg
Exists: True
```

```
C:\Users\Ace\AppData\Local\Temp\ipykernel_37128\2906373448.py:14:
DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns.
This behavior is deprecated, and in a future version of pandas the grouping
columns will be excluded from the operation. Either pass `include_groups=False`
to exclude the groupings or explicitly select the grouping columns after groupby
to silence this warning.
  df = df.groupby('label', group_keys=False).apply(
```

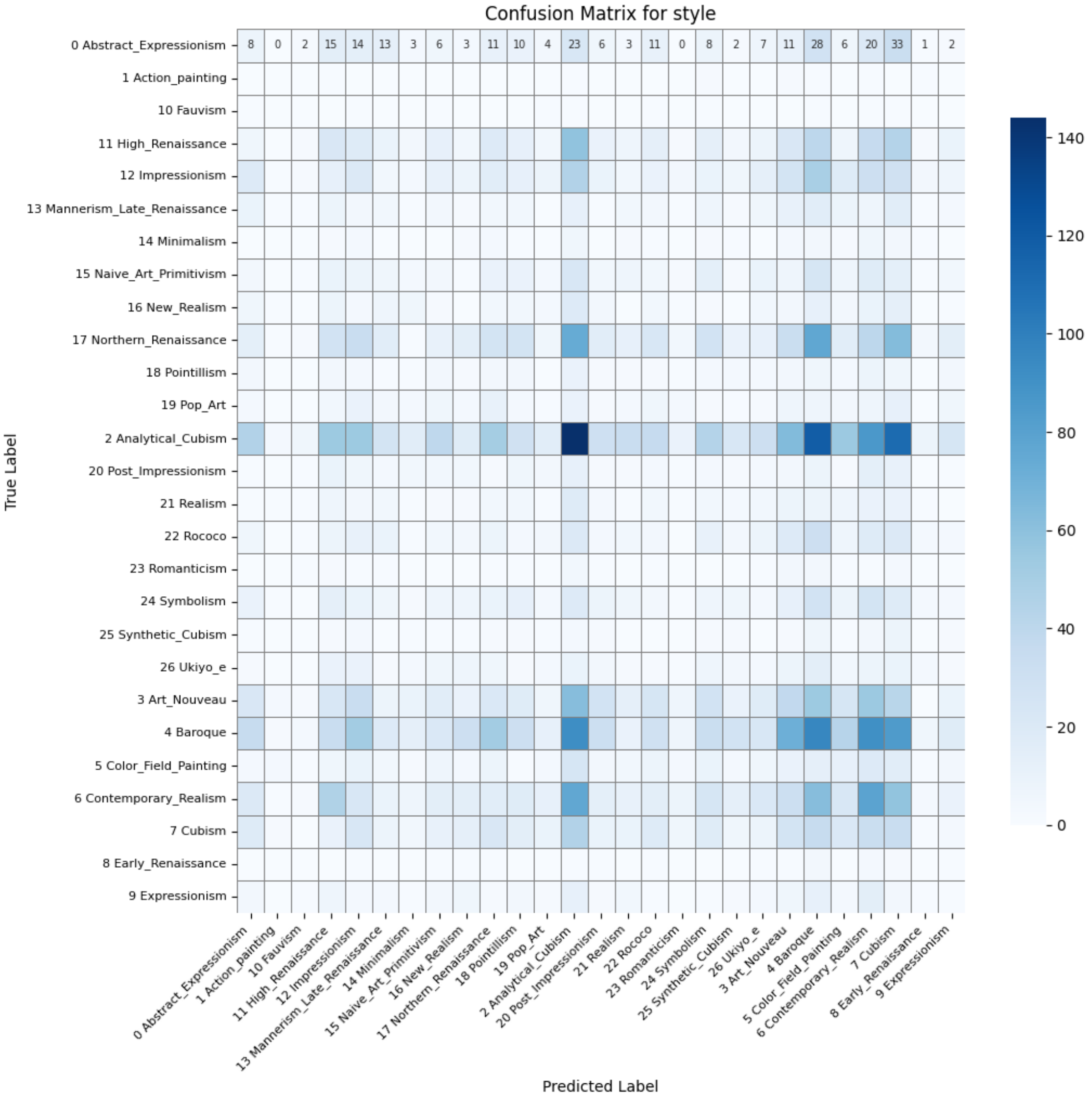
```
Checking if path exists:
C:/Users/Ace/Gsoc_HumanAI/wikiart\Post_Impressionism/rene-magritte_the-staging-
post-1948(1).jpg
Exists: True
Checking if path exists: C:/Users/Ace/Gsoc_HumanAI/wikiart\Rococo/maurice-
quentin-de-la-tour_face-of-the-man-after-alexis-grimou.jpg
Exists: True
Checking if path exists:
C:/Users/Ace/Gsoc_HumanAI/wikiart\Color_Field_Painting/ellsworth-
kelly_rectangle-from-the-series-line-form-color-1951.jpg
Exists: True
Checking if path exists: C:/Users/Ace/Gsoc_HumanAI/wikiart\Realism/john-singer-
sargent_mrs-frederick-mead-mary-eliza-scribner.jpg
Exists: True
Checking if path exists:
C:/Users/Ace/Gsoc_HumanAI/wikiart\Post_Impressionism/jan-sluyters_the-white-
tree.jpg
Exists: True
```

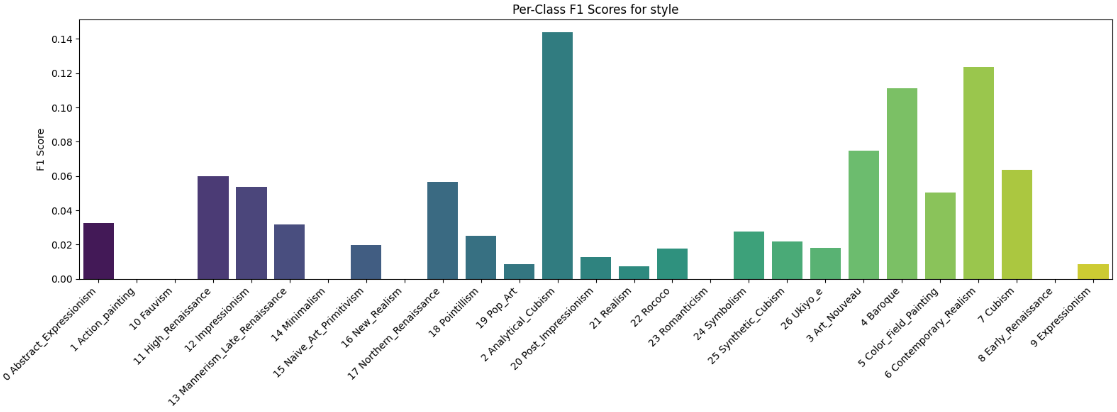
```
Model: "functional_5"
```

```
 Layer (type)                     Output Shape                  Param #

 input_layer_11 (InputLayer)      (None, 224, 224, 3)                 0

 efficientnetb2 (Functional)      (None, 7, 7, 1408)          7,768,569

 global_average_pooling2d_5       (None, 1408)                        0
 (GlobalAveragePooling2D)

 dense_20 (Dense)                 (None, 1024)                1,442,816

 batch_normalization_15           (None, 1024)                    4,096
 (BatchNormalization)

 dropout_15 (Dropout)             (None, 1024)                        0

 dense_21 (Dense)                 (None, 512)                   524,800
 batch_normalization_16           (None, 512)                    2,048
 (BatchNormalization)

 dropout_16 (Dropout)             (None, 512)                        0

 dense_22 (Dense)                 (None, 256)                  131,328

 batch_normalization_17           (None, 256)                    1,024
 (BatchNormalization)

 dropout_17 (Dropout)             (None, 256)                        0

 dense_23 (Dense)                 (None, 27)                     6,939


 Total params: 9,881,620 (37.70 MB)

 Trainable params: 5,084,135 (19.39 MB)

 Non-trainable params: 4,797,485 (18.30 MB)

None
Phase 1: Initial training with mostly frozen base model…
Epoch 1/7
535/535              0s 1s/step -
accuracy: 0.1519 - loss: 3.2685
Epoch 1: val_accuracy improved from -inf to 0.23492, saving model to
C:/Users/Ace/Gsoc_HumanAI/models/best_model_style.keras
535/535              984s 2s/step -
accuracy: 0.1520 - loss: 3.2675 - val_accuracy: 0.2349 - val_loss: 2.6823 -
learning_rate: 0.0010
Epoch 2/7
535/535              0s 1s/step -
accuracy: 0.2718 - loss: 2.1875
Epoch 2: val_accuracy improved from 0.23492 to 0.29252, saving model to
C:/Users/Ace/Gsoc_HumanAI/models/best_model_style.keras
535/535              898s 2s/step -
accuracy: 0.2718 - loss: 2.1872 - val_accuracy: 0.2925 - val_loss: 2.2492 -
learning_rate: 0.0010
Epoch 3/7
535/535              0s 1s/step -
accuracy: 0.3474 - loss: 1.8038
Epoch 3: val_accuracy improved from 0.29252 to 0.30713, saving model to
C:/Users/Ace/Gsoc_HumanAI/models/best_model_style.keras
535/535              888s 2s/step -
accuracy: 0.3474 - loss: 1.8036 - val_accuracy: 0.3071 - val_loss: 2.3143 -
```

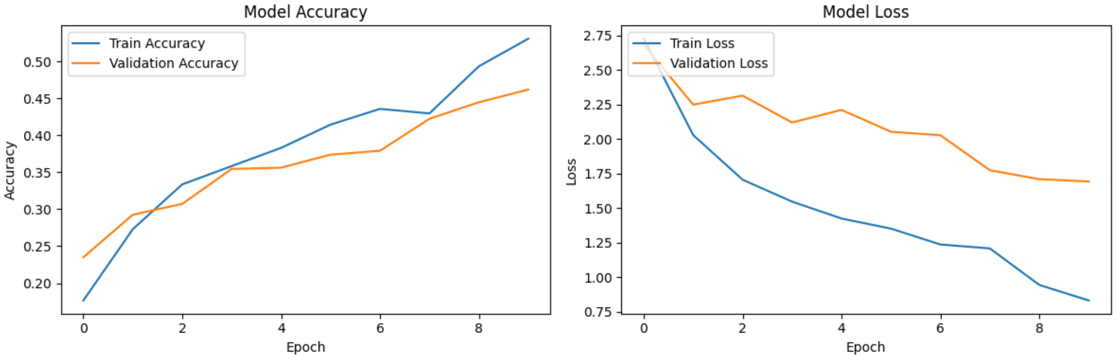
```
learning_rate: 0.0010
Epoch 4/7
535/535            0s 1s/step -
accuracy: 0.3711 - loss: 1.6134
Epoch 4: val_accuracy improved from 0.30713 to 0.35449, saving model to
C:/Users/Ace/Gsoc_HumanAI/models/best_model_style.keras
535/535            899s 2s/step -
accuracy: 0.3710 - loss: 1.6133 - val_accuracy: 0.3545 - val_loss: 2.1200 -
learning_rate: 0.0010
Epoch 5/7
535/535            0s 1s/step -
accuracy: 0.3985 - loss: 1.4677
Epoch 5: val_accuracy improved from 0.35449 to 0.35613, saving model to
C:/Users/Ace/Gsoc_HumanAI/models/best_model_style.keras
535/535            888s 2s/step -
accuracy: 0.3985 - loss: 1.4676 - val_accuracy: 0.3561 - val_loss: 2.2112 -
learning_rate: 0.0010
Epoch 6/7
535/535            0s 1s/step -
accuracy: 0.4395 - loss: 1.4128
Epoch 6: val_accuracy improved from 0.35613 to 0.37374, saving model to
C:/Users/Ace/Gsoc_HumanAI/models/best_model_style.keras
535/535            889s 2s/step -
accuracy: 0.4394 - loss: 1.4127 - val_accuracy: 0.3737 - val_loss: 2.0529 -
learning_rate: 0.0010
Epoch 7/7
535/535            0s 1s/step -
accuracy: 0.4591 - loss: 1.2794
Epoch 7: val_accuracy improved from 0.37374 to 0.37906, saving model to
C:/Users/Ace/Gsoc_HumanAI/models/best_model_style.keras
535/535            882s 2s/step -
accuracy: 0.4590 - loss: 1.2794 - val_accuracy: 0.3791 - val_loss: 2.0281 -
learning_rate: 0.0010
Restoring model weights from the end of the best epoch: 7.
Phase 2: Fine-tuning with more layers unfrozen…
Epoch 8/10
535/535            0s 5s/step -
accuracy: 0.4083 - loss: 1.3818
Epoch 8: val_accuracy improved from 0.37906 to 0.42219, saving model to
C:/Users/Ace/Gsoc_HumanAI/models/best_model_style.keras
535/535            3269s 6s/step -
accuracy: 0.4083 - loss: 1.3814 - val_accuracy: 0.4222 - val_loss: 1.7743 -
learning_rate: 1.0000e-04
Epoch 9/10
535/535            0s 5s/step -
accuracy: 0.4902 - loss: 1.0179
Epoch 9: val_accuracy improved from 0.42219 to 0.44458, saving model to
C:/Users/Ace/Gsoc_HumanAI/models/best_model_style.keras
```

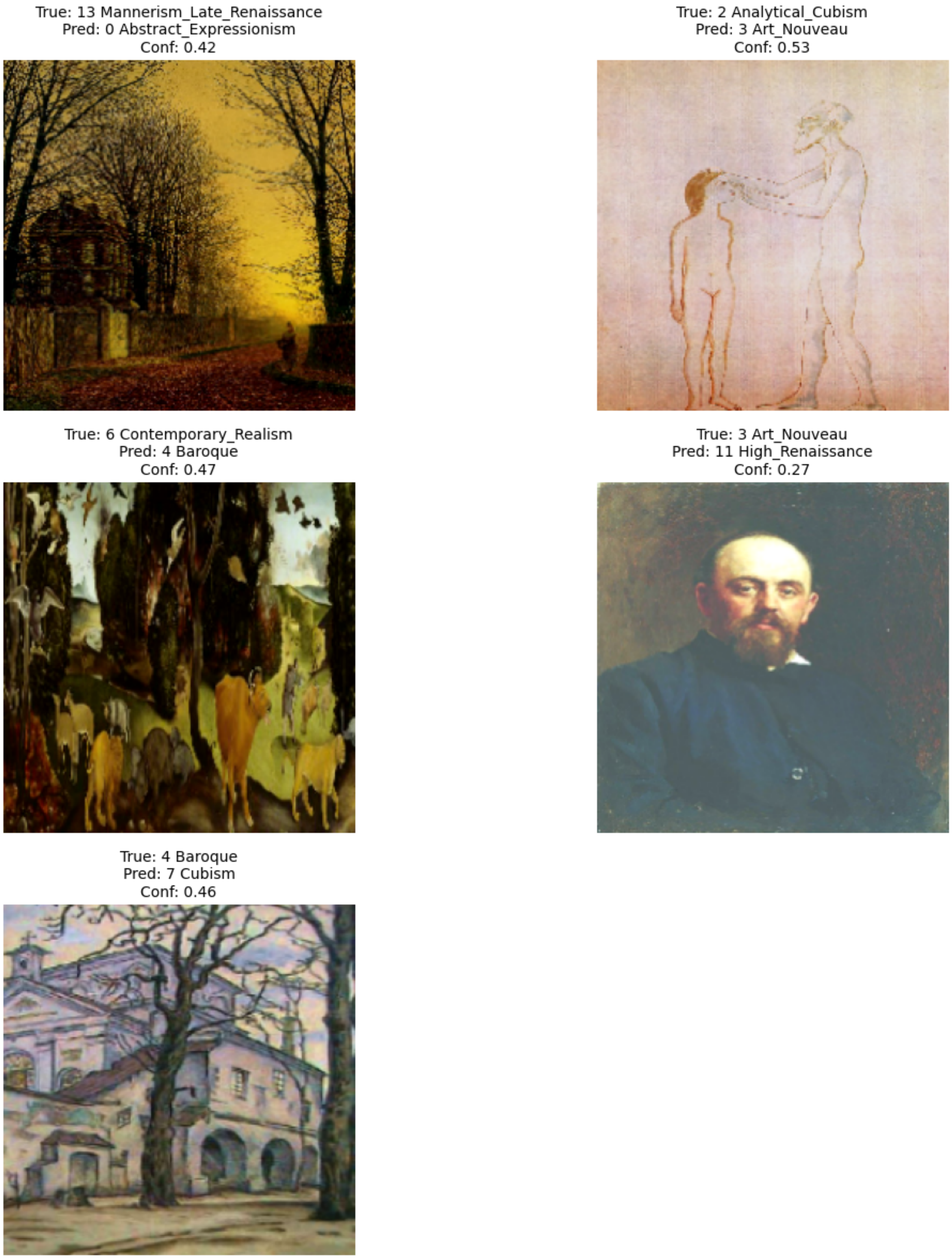
```
535/535             3135s 6s/step -
accuracy: 0.4902 - loss: 1.0178 - val_accuracy: 0.4446 - val_loss: 1.7096 -
learning_rate: 1.0000e-04
Epoch 10/10
535/535             0s 6s/step -
accuracy: 0.5310 - loss: 0.8715
Epoch 10: val_accuracy improved from 0.44458 to 0.46178, saving model to
C:/Users/Ace/Gsoc_HumanAI/models/best_model_style.keras
535/535             3479s 6s/step -
accuracy: 0.5310 - loss: 0.8714 - val_accuracy: 0.4618 - val_loss: 1.6927 -
learning_rate: 1.0000e-04
Restoring model weights from the end of the best epoch: 10.
Evaluating model…
Model architecture saved as 'results/model_architecture_style.png'
229/229             240s 879ms/step

Accuracy: 0.0700
F1 Score: 0.0749
```


Confusion Matrix for style
True Label
Predicted Label
0 Abstract_Expressionism
1 Action_painting
10 Fauvism
11 High_Renaissance
12 Impressionism
13 Mannerism_Late_Renaissance
14 Minimalism
15 Naive_Art_Primitivism
16 New_Realism
17 Northern_Renaissance
18 Pointillism
19 Pop_Art
2 Analytical_Cubism
20 Post_Impressionism
21 Realism
22 Rococo
23 Romanticism
24 Symbolism
25 Synthetic_Cubism
26 Ukiyo_e
3 Art_Nouveau
4 Baroque
5 Color_Field_Painting
6 Contemporary_Realism
7 Cubism
8 Early_Renaissance
9 Expressionism
8 0 2 15 14 13 3 6 3 11 10 4 23 6 3 11 0 8 2 7 11 28 6 20 33 1 2
140
120
100
80
60
40
20
0


Per-Class F1 Scores for style
F1 Score
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0 Abstract_Expressionism
1 Action_painting
10 Fauvism
11 High_Renaissance
12 Impressionism
13 Mannerism_Late_Renaissance
14 Minimalism
15 Naive_Art_Primitivism
16 New_Realism
17 Northern_Renaissance
18 Pointillism
19 Pop_Art
2 Analytical_Cubism
20 Post_Impressionism
21 Realism
22 Rococo
23 Romanticism
24 Symbolism
25 Synthetic_Cubism
26 Ukiyo_e
3 Art_Nouveau
4 Baroque
5 Color_Field_Painting
6 Contemporary_Realism
7 Cubism
8 Early_Renaissance
9 Expressionism

```
Found 7107 potential outliers
Visualizing results…
```




True: 13 Mannerism_Late_Renaissance
Pred: 0 Abstract_Expressionism
Conf: 0.42
True: 2 Analytical_Cubism
Pred: 3 Art_Nouveau
Conf: 0.53
True: 6 Contemporary_Realism
Pred: 4 Baroque
Conf: 0.47
True: 3 Art_Nouveau
Pred: 11 High_Renaissance
Conf: 0.27
True: 4 Baroque
Pred: 7 Cubism
Conf: 0.46

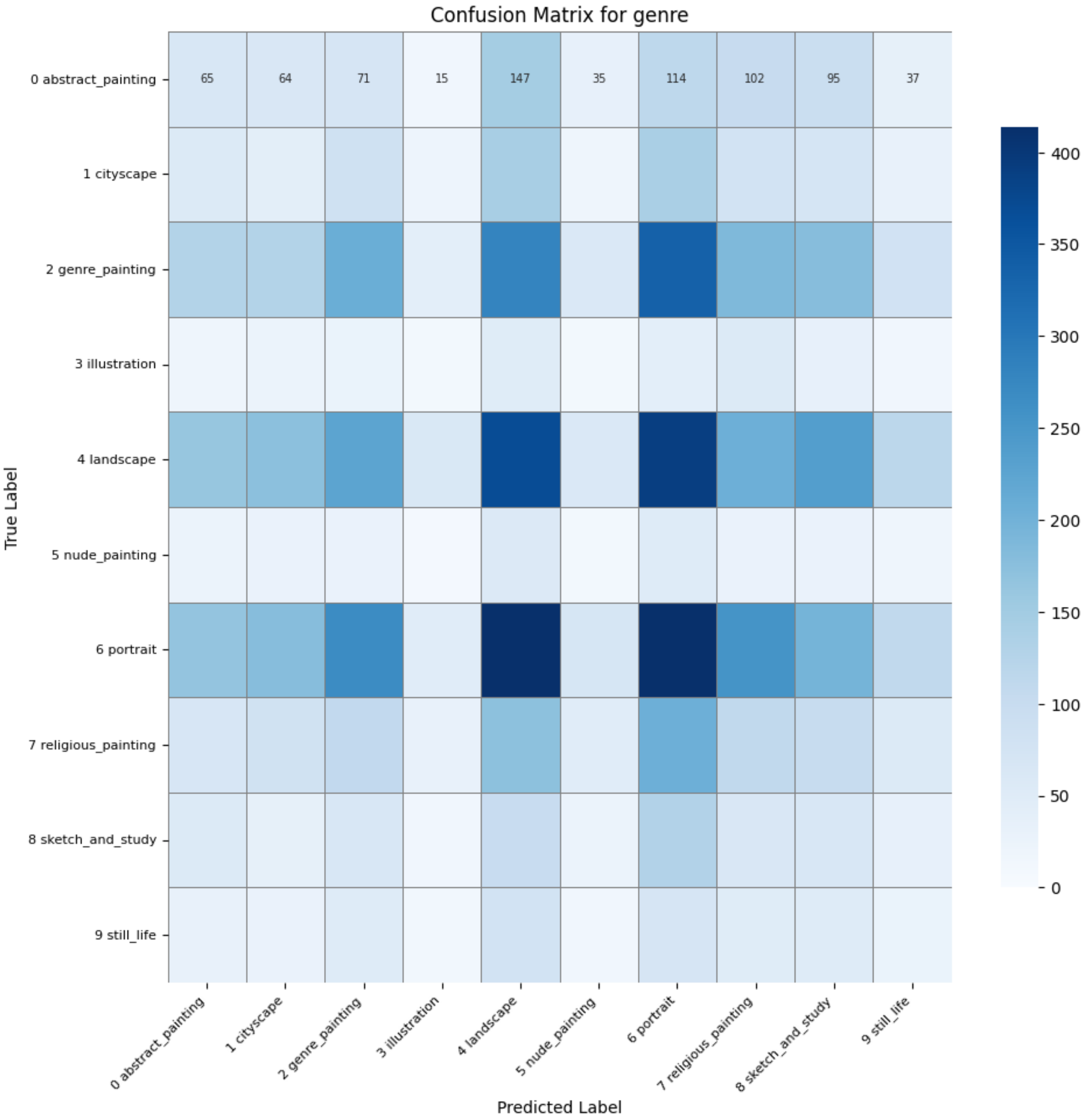
```
Outliers saved to: results/style\outliers_style.png
```

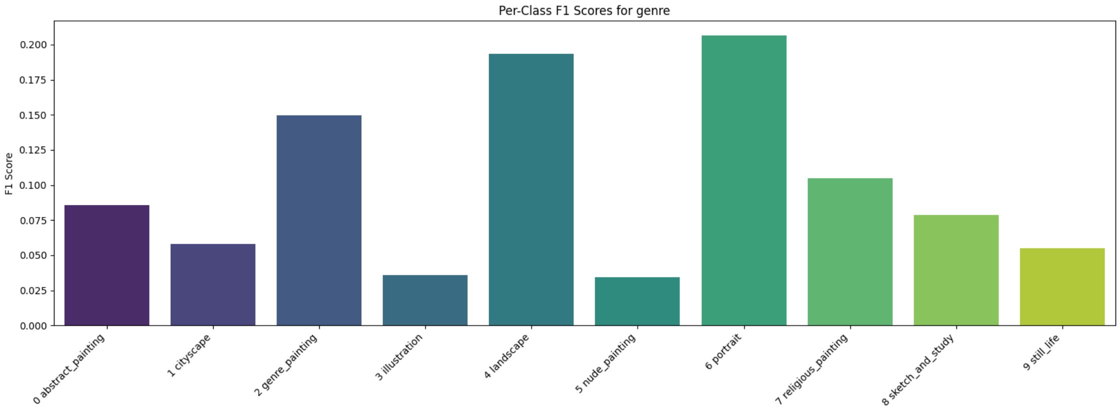
```
============================================================
Training improved model for GENRE
============================================================


C:\Users\Ace\AppData\Local\Temp\ipykernel_37128\2906373448.py:14:
DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns.
This behavior is deprecated, and in a future version of pandas the grouping
columns will be excluded from the operation. Either pass `include_groups=False`
to exclude the groupings or explicitly select the grouping columns after groupby
to silence this warning.
  df = df.groupby('label', group_keys=False).apply(

Checking if path exists: C:/Users/Ace/Gsoc_HumanAI/wikiart\Rococo/thomas-
gainsborough_portrait-of-george-iii-1781.jpg
Exists: True
Checking if path exists:
C:/Users/Ace/Gsoc_HumanAI/wikiart\Post_Impressionism/vincent-van-gogh_the-
bridge.jpg
Exists: True
Checking if path exists:
C:/Users/Ace/Gsoc_HumanAI/wikiart\Northern_Renaissance/lucas-cranach-the-
elder_christ-carrying-the-cross-1538.jpg
Exists: True
Checking if path exists: C:/Users/Ace/Gsoc_HumanAI/wikiart\Romanticism/jan-
matejko_sigmund-and-barbara.jpg
Exists: True
Checking if path exists: C:/Users/Ace/Gsoc_HumanAI/wikiart\Symbolism/kuzma-
petrov-vodkin_portrait-of-ria-portrait-of-a-a-kholopova-1915.jpg
Exists: True

C:\Users\Ace\AppData\Local\Temp\ipykernel_37128\2906373448.py:14:
DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns.
This behavior is deprecated, and in a future version of pandas the grouping
columns will be excluded from the operation. Either pass `include_groups=False`
to exclude the groupings or explicitly select the grouping columns after groupby
to silence this warning.
  df = df.groupby('label', group_keys=False).apply(

Checking if path exists:
C:/Users/Ace/Gsoc_HumanAI/wikiart\High_Renaissance/vittore-carpaccio_portrait-
of-a-lady-1.jpg
Exists: True
Checking if path exists: C:/Users/Ace/Gsoc_HumanAI/wikiart\Realism/ivan-
shishkin_pines-sunny-day.jpg
Exists: True
Checking if path exists:
C:/Users/Ace/Gsoc_HumanAI/wikiart\Post_Impressionism/georges-braque_nude.jpg
Exists: True
Checking if path exists: C:/Users/Ace/Gsoc_HumanAI/wikiart\Realism/adolf-
```

```
hitler_the-st-charles-church.jpg
Exists: True
Checking if path exists: C:/Users/Ace/Gsoc_HumanAI/wikiart\Realism/winslow-
homer_the-bridal-path-white-mountains-1868.jpg
Exists: True
```

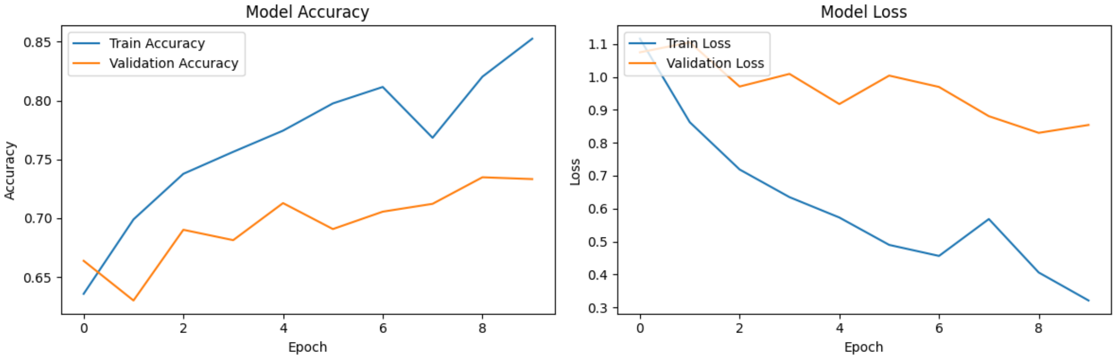
```
Model: "functional_6"


 Layer (type)                        Output Shape                  Param #

 input_layer_13 (InputLayer)         (None, 224, 224, 3)                 0

 efficientnetb2 (Functional)         (None, 7, 7, 1408)          7,768,569

 global_average_pooling2d_6          (None, 1408)                        0
 (GlobalAveragePooling2D)

 dense_24 (Dense)                    (None, 1024)                1,442,816

 batch_normalization_18              (None, 1024)                    4,096
 (BatchNormalization)

 dropout_18 (Dropout)                (None, 1024)                        0

 dense_25 (Dense)                    (None, 512)                   524,800

 batch_normalization_19              (None, 512)                     2,048
 (BatchNormalization)

 dropout_19 (Dropout)                (None, 512)                         0

 dense_26 (Dense)                    (None, 256)                   131,328

 batch_normalization_20              (None, 256)                     1,024
 (BatchNormalization)

 dropout_20 (Dropout)                (None, 256)                         0

 dense_27 (Dense)                    (None, 10)                      2,570


 Total params: 9,877,251 (37.68 MB)


 Trainable params: 6,453,502 (24.62 MB)
 Non-trainable params: 3,423,749 (13.06 MB)

None
Phase 1: Initial training with mostly frozen base model…
Epoch 1/7
711/711            0s 1s/step -
accuracy: 0.6132 - loss: 1.1923
Epoch 1: val_accuracy improved from -inf to 0.66383, saving model to
C:/Users/Ace/Gsoc_HumanAI/models/best_model_genre.keras
711/711            1449s 2s/step -
accuracy: 0.6132 - loss: 1.1922 - val_accuracy: 0.6638 - val_loss: 1.0750 -
learning_rate: 0.0010
Epoch 2/7
711/711            0s 1s/step -
accuracy: 0.6929 - loss: 0.8629
Epoch 2: val_accuracy did not improve from 0.66383
711/711            1305s 2s/step -
accuracy: 0.6929 - loss: 0.8629 - val_accuracy: 0.6301 - val_loss: 1.1054 -
learning_rate: 0.0010
Epoch 3/7
711/711            0s 1s/step -
accuracy: 0.7378 - loss: 0.7055
Epoch 3: val_accuracy improved from 0.66383 to 0.69020, saving model to
C:/Users/Ace/Gsoc_HumanAI/models/best_model_genre.keras
711/711            1321s 2s/step -
accuracy: 0.7378 - loss: 0.7055 - val_accuracy: 0.6902 - val_loss: 0.9710 -
learning_rate: 0.0010
Epoch 4/7
711/711            0s 1s/step -
accuracy: 0.7599 - loss: 0.6266
Epoch 4: val_accuracy did not improve from 0.69020
711/711            1303s 2s/step -
accuracy: 0.7599 - loss: 0.6267 - val_accuracy: 0.6814 - val_loss: 1.0094 -
learning_rate: 0.0010
Epoch 5/7
711/711            0s 1s/step -
accuracy: 0.7829 - loss: 0.5395
Epoch 5: val_accuracy improved from 0.69020 to 0.71288, saving model to
C:/Users/Ace/Gsoc_HumanAI/models/best_model_genre.keras
711/711            1307s 2s/step -
accuracy: 0.7829 - loss: 0.5396 - val_accuracy: 0.7129 - val_loss: 0.9178 -
learning_rate: 0.0010
Epoch 6/7
711/711            0s 1s/step -
accuracy: 0.8027 - loss: 0.4495
Epoch 6: val_accuracy did not improve from 0.71288
711/711            1299s 2s/step -
```

```
accuracy: 0.8027 - loss: 0.4496 - val_accuracy: 0.6908 - val_loss: 1.0042 -
learning_rate: 0.0010
Epoch 7/7
711/711             0s 1s/step -
accuracy: 0.8107 - loss: 0.4533
Epoch 7: val_accuracy did not improve from 0.71288
711/711             1314s 2s/step -
accuracy: 0.8107 - loss: 0.4533 - val_accuracy: 0.7056 - val_loss: 0.9698 -
learning_rate: 0.0010
Restoring model weights from the end of the best epoch: 5.
Phase 2: Fine-tuning with more layers unfrozen…
Epoch 8/10
711/711             0s 5s/step -
accuracy: 0.7141 - loss: 0.6954
Epoch 8: val_accuracy did not improve from 0.71288
711/711             4568s 6s/step -
accuracy: 0.7142 - loss: 0.6952 - val_accuracy: 0.7123 - val_loss: 0.8809 -
learning_rate: 1.0000e-04
Epoch 9/10
711/711             0s 5s/step -
accuracy: 0.7961 - loss: 0.4513
Epoch 9: val_accuracy improved from 0.71288 to 0.73484, saving model to
C:/Users/Ace/Gsoc_HumanAI/models/best_model_genre.keras
711/711             4495s 6s/step -
accuracy: 0.7962 - loss: 0.4512 - val_accuracy: 0.7348 - val_loss: 0.8303 -
learning_rate: 1.0000e-04
Epoch 10/10
711/711             0s 5s/step -
accuracy: 0.8349 - loss: 0.3517
Epoch 10: val_accuracy did not improve from 0.73484
711/711             4314s 6s/step -
accuracy: 0.8349 - loss: 0.3516 - val_accuracy: 0.7333 - val_loss: 0.8543 -
learning_rate: 1.0000e-04
Restoring model weights from the end of the best epoch: 9.
Evaluating model…
Model architecture saved as 'results/model_architecture_genre.png'
305/305             326s 908ms/step

Accuracy: 0.1356
F1 Score: 0.1401
```


Confusion Matrix for genre
True Label
Predicted Label
0 abstract_painting
1 cityscape
2 genre_painting
3 illustration
4 landscape
5 nude_painting
6 portrait
7 religious_painting
8 sketch_and_study
9 still_life
65
64
71
15
147
35
114
102
95
37
400
350
300
250
200
150
100
50
0


Per-Class F1 Scores for genre
F1 Score
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0 abstract_painting
1 cityscape
2 genre_painting
3 illustration
4 landscape
5 nude_painting
6 portrait
7 religious_painting
8 sketch_and_study
9 still_life

```
Found 8521 potential outliers
Visualizing results…
```



True: 2 genre_painting
Pred: 1 cityscape
Conf: 0.99

True: 2 genre_painting
Pred: 3 illustration
Conf: 0.93

True: 2 genre_painting
Pred: 9 still_life
Conf: 0.99

True: 0 abstract_painting
Pred: 7 religious_painting
Conf: 0.70

True: 0 abstract_painting
Pred: 4 landscape
Conf: 0.64

```
Outliers saved to: results/genre\outliers_genre.png
```