50.038 Data Science Project: Style Analysis and Classification for Digital Art

Data Dragons
Lee Chien Shyong 1005903, Khoo Zi Qi 1006984

April 2025

Project Repository: `https://github.com/chienshyong/yukien`

Abstract

This project is about the analysis and classification of digital art styles using machine learning. We developed a model that can not only identify an artist by their artwork, but extract key visual features that make their style recognizable. By collecting a dataset of images from various artists, performing feature extraction, and applying a hybrid convolutional neural network (CNN) to classify the artworks, our model achieves an accuracy of 89% using standard metrics. This work can assist artists in learning a new style, contribute to tasks such as digital curation, and improving recommendation systems for online art platforms.

1 Motivation

In the 21st century, drawing has evolved from something done primarily on paper to a practice increasingly conducted on screens using digital tools. While traditional art forms remain foundational, digital art has emerged as a legitimate and widely accessible creative outlet. Leveraging technologies such as drawing tablets and software like Adobe Photoshop, artists can now simulate classical media or develop entirely new styles that were not feasible with physical materials. Digital drawing has become the norm in illustration, graphic design, architecture, concept art, and many more industries.

Digital artists operate in a rapidly evolving field where mastery of visual elements such as detail, color, and contrast is crucial for producing compelling artwork. To do so, studying other artists' work as a means of improving artistic skill is commonplace. By analysing artistic techniques of established artists, digital artists can refine their technical abilities.

Our goal is to develop an AI model that can identify visual patterns that define an artist's style, distinguish between artworks based on these patterns, and help artists like ourselves to explore why certain features are more recognizable than others.

Standard image recognition models are a "black box" where the user has no knowledge about how the model has identified the class of an image. In contrast, we hope to create a model that provides some insight to the user about which features of the artwork has made it identifiable.

2 Dataset and Collection

2.1 Dataset Type and Source

The dataset used in this project comprises high-resolution digital artworks collected for a multi-class image classification task. Each image is attributed to one of eight distinct artists, enabling the model to learn and differentiate unique visual styles. All images were sourced from **Danbooru**, an image board where users upload and tag artworks. While the platform offers rich metadata through tagging, this project focuses solely on the visual content and does not utilize the tags.

2.2 Data Collection Methodology

The code used for data collection can be found in `scrape.ipynb`. To collect the data, we used `gallery-dl`, a command-line scraping tool that supports various image hosting websites. For each artist, we filtered for general-rated images to ensure content appropriateness and quality consistency. The images were downloaded using the following command:

```
gallery-dl "https://danbooru.donmai.us/posts?tags=<artist_name>+rating:general"
```

The downloaded images were organized into separate folders labelled by artist name, which acted as class labels for training.

2.3 Dataset Diversity

Although all images originated from the same platform, we curated a dataset consisting of artworks from **eight different artists**. While this constitutes a single dataset, the inclusion of multiple artist classes introduces significant stylistic diversity. This approach allows the model to focus on distinguishing between various visual styles and reduces the risk of bias toward any one artist. Each class contains hundreds of images, resulting in a balanced and robust dataset for evaluating classification performance.

The artists were selected as some of our personal favorites, while also meeting the criteria of having enough (at least 100) works to train a model on.

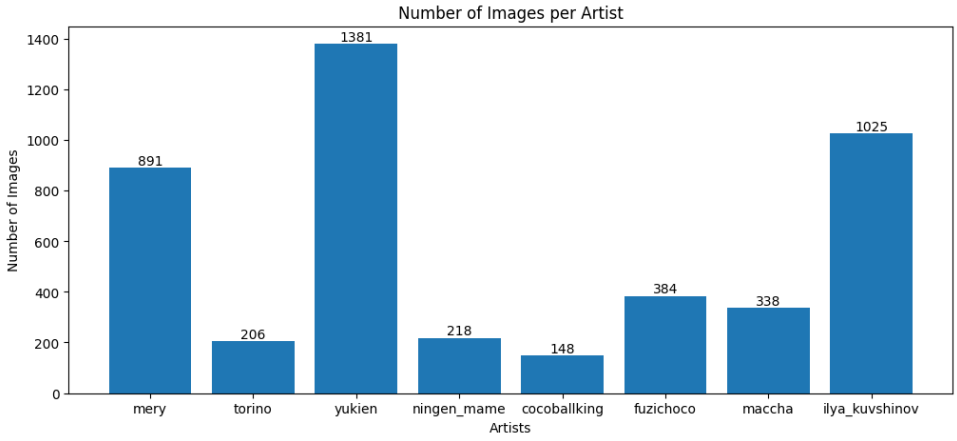


Figure 1: Number of Images per Artist.

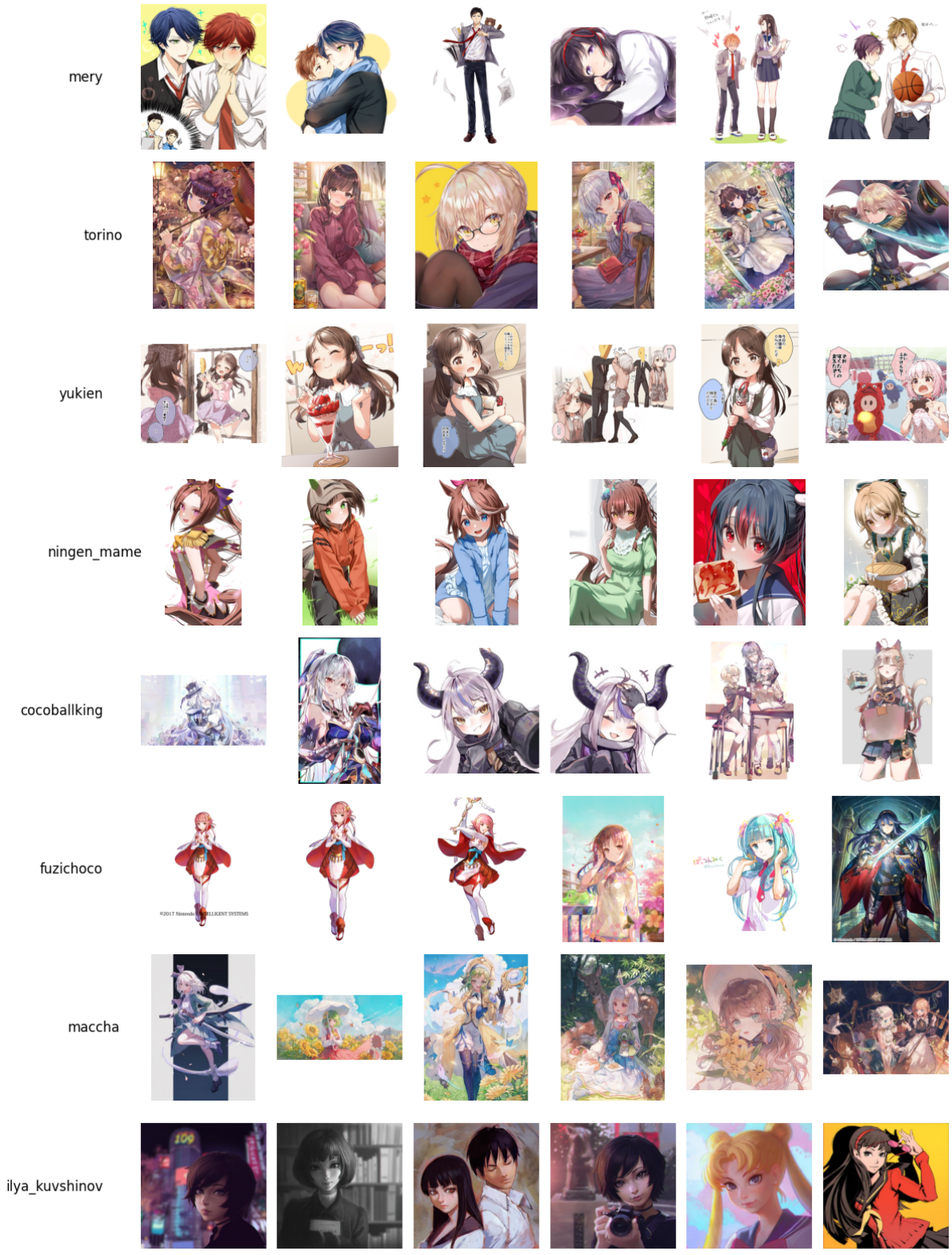


Figure 2: Example artworks from all eight artists. Each row corresponds to a different artist.

3 Data Pre-processing

3.1 Standardization

Before anything else, the downloaded images were standardized and shrunk to reduce computational time. All images were converted to PNG format, corrupted images removed, and resized to 400px wide (maintaining the aspect ratio). The code used for data collection can be found in `scrape.ipynb`.

After preprocessing, we save the images to the `/dataset` folder, separated further by artist into subfolders.

3.2 Feature Extraction

To achieve our objective of developing a model that provides insight into each artist's unique visual style, we perform feature extraction on the dataset. As digital artists ourselves, we have some intuition as to what features might differentiate one artist from another. In total, we extract 20 additional features: `edge_density`, `laplacian_variance`, `shannon_entropy`, `hs_colorfulness`, `color_spread`, `color_entropy`, `temp_mean`, `temp_stddev`, `gray_mean`, `gray_stddev`, `lbp_0`, `lbp_1`, `lbp_2`, `lbp_3`, `lbp_4`, `lbp_5`, `lbp_6`, `lbp_7`, `lbp_8`, `lbp_9`. The code can be found in `feature_extraction.ipynb`.

3.3 Explanation of Each Extracted Feature

3.3.1 Canny Edge Density

Canny is an algorithm that detects **edges**. The Canny edge detection algorithm is run, and the proportion of edge pixels compared to the whole image are calculated as a new feature. This gives a rough measurement for the amount of detail in an artwork.

3.3.2 Laplacian Variance

Laplacian variance measures **sharpness**. It is a second-derivative operator that detects areas of rapid intensity change. It highlights regions where the intensity shifts from light to dark and vice versa.

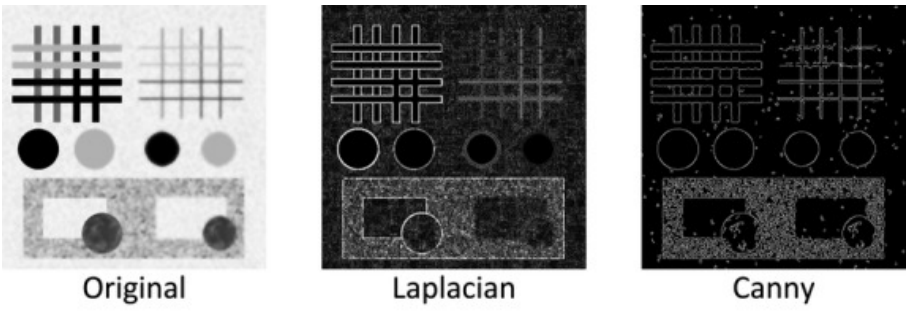


Figure 3: Output of the Canny and Laplacian algorithm. [1]

3.3.3 Shannon Entropy

In the context of images, Shannon entropy quantifies how much information is contained in the image based on the distribution of pixel intensities. The entropy gives a scalar measure of the **texture richness**.

For a discrete random variable X with possible outcomes $x_1, x_2, \ldots, x_n$ and corresponding probabilities $P(x_1), P(x_2), \ldots, P(x_n)$, the Shannon entropy $H(X)$ is defined as:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_2 P(x_i)$$

3.3.4 Hasler and Süsstrunk's Colorfulness Metric

The first of two metrics used to quantify **colorfulness**. To quantify the perceptual colorfulness of an image, Hasler and Süsstrunk (2003) proposed a metric based on the opponent color channels: [2]

$$rg = R - G, \quad yb = \frac{1}{2}(R + G) - B$$

Compute the mean and standard deviation of these two components:

$$\sigma_{rg},\ \mu_{rg},\ \sigma_{yb},\ \mu_{yb}$$

The final colorfulness score C is given by:

$$C = \sqrt{\sigma_{rg}^2 + \sigma_{yb}^2} + 0.3 \cdot \sqrt{\mu_{rg}^2 + \mu_{yb}^2}$$

This metric reflects both the variation and the magnitude of color components in the image.

3.3.5 Color Histogram Diversity

The second of two metrics used to quantify **colorfulness**. We calculate color histograms to generate two scalar features: **Color spread**, corresponding to many different color values appear, and **Color entropy**, corresponsing to how uniformly the colors are distributed.

The histograms are generated with numpy's `histogram` function.

```
def _histogram_spread(channel, bins=32):
    hist, _ = np.histogram(channel, bins=bins, range=(0, 1), density=False)
    non_zero_bins = np.count_nonzero(hist)
    return non_zero_bins / bins # value in [0, 1]

def _histogram_entropy(channel, bins=32):
    hist, _ = np.histogram(channel, bins=bins, range=(0, 1), density=True)
    return entropy(hist + 1e-10) # add epsilon to avoid log(0)
```

More intuitively, the **Color spread** describes how many colors an artist uses, and **Color entropy** describes whether they used a lot of one and little of the other, or an even amount of all.

3.3.6 Color Temperature Analysis

In classical art, **color temperature** refers to the perceived warmth or coolness of colors used in a painting or composition. Warm colors such as Red, orange, and yellow are associated with sunlight, fire, and heat and elicit emotions like energy, warmth, or passion. Cool colors like Blue, green, violet are associated with shade, water, and sky, and suggest calm, distance, or serenity.

In a digital artwork, we can get a measure of the temperature by comparing the levels of the red channel and the blue channel. (The green channel is ignored because green is a neutral temperature color. The presence or absence of green, in theory, does not affect the perceived temperature of a color). To estimate perceptual color temperature, we use the mean and standard deviation of the difference between the red and blue color channels of an image:

Let $R(x, y)$ and $B(x, y)$ be the red and blue channel intensities at pixel (x, y), respectively. We define the temperature difference at each pixel as:

$$D(x, y) = R(x, y) - B(x, y)$$

The image-wide statistics are then computed as:

$$\mu_{\text{temp}} = \frac{1}{N} \sum_{x=1}^{W} \sum_{y=1}^{H} D(x, y)$$

$$\sigma_{\text{temp}} = \sqrt{\frac{1}{N} \sum_{x=1}^{W} \sum_{y=1}^{H} (D(x, y) - \mu_{\text{temp}})^2}$$

Where:	W, H	are the image width and height,
	$N = W \times H$	is the total number of pixels,
	μ_{temp}	is the mean red-blue difference,
	σ_{temp}	is the standard deviation of that difference.

The temperature mean describe the image's dominant color tone: higher positive values indicate a warmer tone (more red), while negative values suggest a cooler tone (more blue). The temperature standard deviation describe the range of temperatures that an artist uses.

Figure 4: Temperature analysis of one of our own artworks. Skin tones and the pink background are warm while the cyan rim light is cool.

Extracting the color temperature as a feature for machine learning is a **novelty** that comes from our insight as artists ourselves!

3.3.7 Brightness and Contrast

Converts the image to greyscale and computes the mean and standard deviation of grayscale intensities. This gives a measurement for the average brightness of an artwork and the amount of contrast, respectively.

3.3.8 Local Binary Patterns

Local Binary Pattern (LBP) is a **texture descriptor** used in computer vision and image processing. It captures the local structure around each pixel in an image by encoding how each pixel compares to its neighbors. Originally introduced for texture classification, LBP has become a standard method in applications like face recognition, surface inspection, and feature extraction in machine learning.

For a given pixel in a grayscale image, the LBP value is computed by comparing it to its P neighboring pixels spaced evenly on a circle of radius R. Each comparison yields a binary result - 0 if the neighboring pixel is lesser than the central picel, 1 otherwise. These are then concatenated into a binary number.

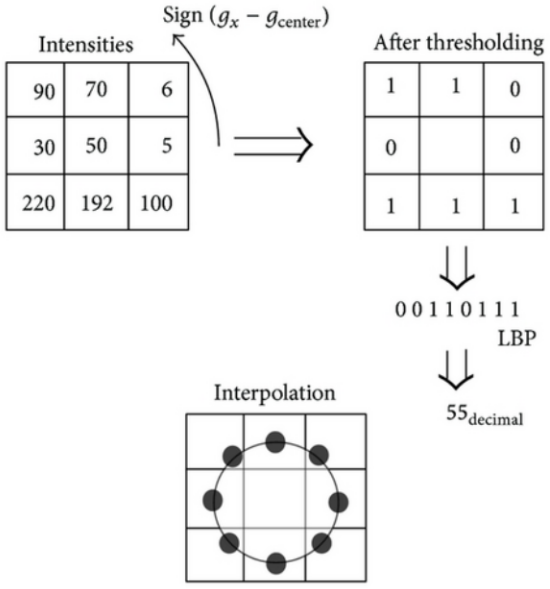


Figure 5: Conversion of a 3x3 section of an image into an LBP of 55. [3]

Mathematically: Let I_c be the intensity of the center pixel, and I_p the intensity of a neighboring pixel. The LBP code is computed as:

$$\text{LBP}_{P,R} = \sum_{p=0}^{P-1} s(I_p - I_c) \cdot 2^p$$

where the function $s(x)$ is defined as:

$$s(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

With this version of LBP, a total of 255 patterns are produced, from 00000000 to 11111111. However, a simpler version of LBP, the "uniform" LBP variant, exists with only 10 patterns, and performs with high discriminative capability (Lahdenoja 2013). Using the "uniform" LBP variant, only binary patterns with at most two bitwise transitions ($0 \rightarrow 1$ or $1 \rightarrow 0$) are considered, reducing the number of unique LBP codes. Anything with more than two bitwise transitions are grouped under one bucket.

In our implementation, we use scikit-image's `local_binary_pattern` function. The resulting LBP codes across the image are then summarized in a histogram, which serves as a feature vector of length 10 representing the texture of the image.

```
def compute_lbp_features(image_path, radius=1, n_points=8):
    # Load and convert to grayscale
    img = io.imread(image_path)
    # If the image has an alpha channel (RGBA), remove it and keep just RGB
    if img.shape[2] == 4:
        img = img[:, :, :3] # Keep only the RGB channels
    gray = color.rgb2gray(img)

    # Compute LBP
    lbp = feature.local_binary_pattern(gray, P=n_points, R=radius, method='uniform')

    # Compute normalized histogram
    n_bins = int(lbp.max() + 1)
    hist, _ = np.histogram(lbp.ravel(), bins=n_bins, range=(0, n_bins), density=True)

    return hist
```

3.4 Batch Processing and Data Storage

We developed a Python-based feature extraction pipeline that computes the described features for each image and saves them in a CSV file. The code can also be found in `feature_extraction.ipynb`. In this notebook, there are functions that compute each feature, followed by the function `feature_extraction(image_path)` that aggregates these features into a single vector:

```
def feature_extraction(image_path):
    density = edge_density(image_path)
    variance = laplacian_variance(image_path)
    entropy = image_entropy(image_path)
    colorful = colorfulness(image_path)
    color_histogram = color_histogram_diversity(image_path)
    temperature = temperature_analysis(image_path)
    grayscale_contrast = grayscale_contrast_analysis(image_path)
    lbp_features = compute_lbp_features(image_path)

    result = [density, variance, entropy, colorful, color_histogram['avg_spread'],
              color_histogram['avg_entropy'], temperature['temp_mean'],
              temperature['temp_stddev'],
              grayscale_contrast['mean_gray'], grayscale_contrast['std_gray']]
    result += lbp_features.tolist()
    return result
```

The images in our dataset were then processed using a batch function that extracts features from all images and stores the results in a CSV file (`dataset.csv`) for training, along with the

artist label and image path.

```
def process_folder(folder_path, artist_label, output_file='dataset.csv'):
    image_files = [f for f in os.listdir(folder_path)]
    for image_name in tqdm(image_files):
        image_path = os.path.join(folder_path, image_name)
        insert_into_csv(image_path, artist_label, output_file)
```

Thus, our feature extraction step creates a multi-modal dataset where each image is further supplemented by 20 numerical features.

4 Data Visualization

To better understand the dataset, we use box-and-whisker plots to explore the distribution of feature values for each artist. These visualizations help identify which features may contribute most to distinguishing artistic styles.

We have made charts for each of the 20 features mentioned in the previous section, which can be found in `data_visualization.ipynb`. For brevity, we will only look at Edge Density and Colorfulness in this report.

4.1 Edge Density Vizualization

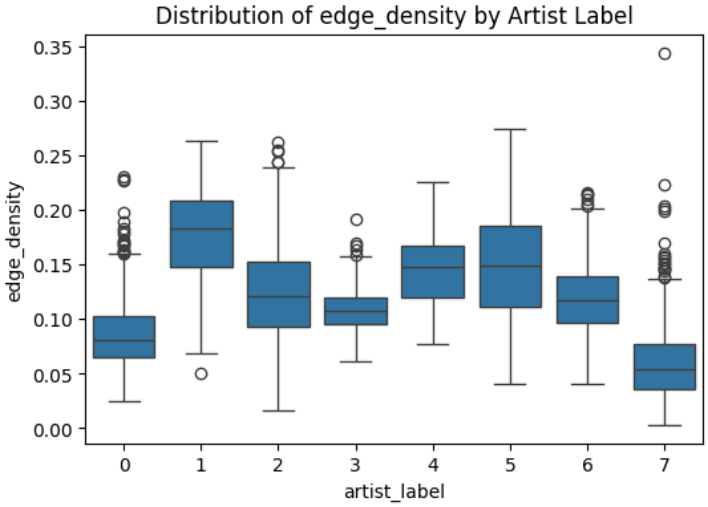


Figure 6: Distribution of Edge Density values across artists.

Edge Density measures the amount of detail in an artwork by counting the proportion of edge pixels after running the Canny algorithm. From this chart, we see that artist 1 has the most detailed artworks on average, and artist 7 has the least detailed artworks on average.

Let's take a look at these two artists' work to examine why that is.

Figure 7: Example artwork by Torino Aqua (artist 1).

Torino Aqua is one of the most popular fan-artists online, with over 1 million followers on Twitter/X. They are known for their highly detailed artwork of pop culture characters, like the ones above from *Genshin Impact*.

By insisting on meticulous rendering of every object, an expansive canvas jam-packed with visual interst is created. Additionally, 3D techniques are utilized to generate details and textures above what can be done with only 2D drawing.

While other artists in the dataset have artwork with similar detail levels, Torino is the only one that insists on extreme detail in every illustration, leading to by far the highest average.

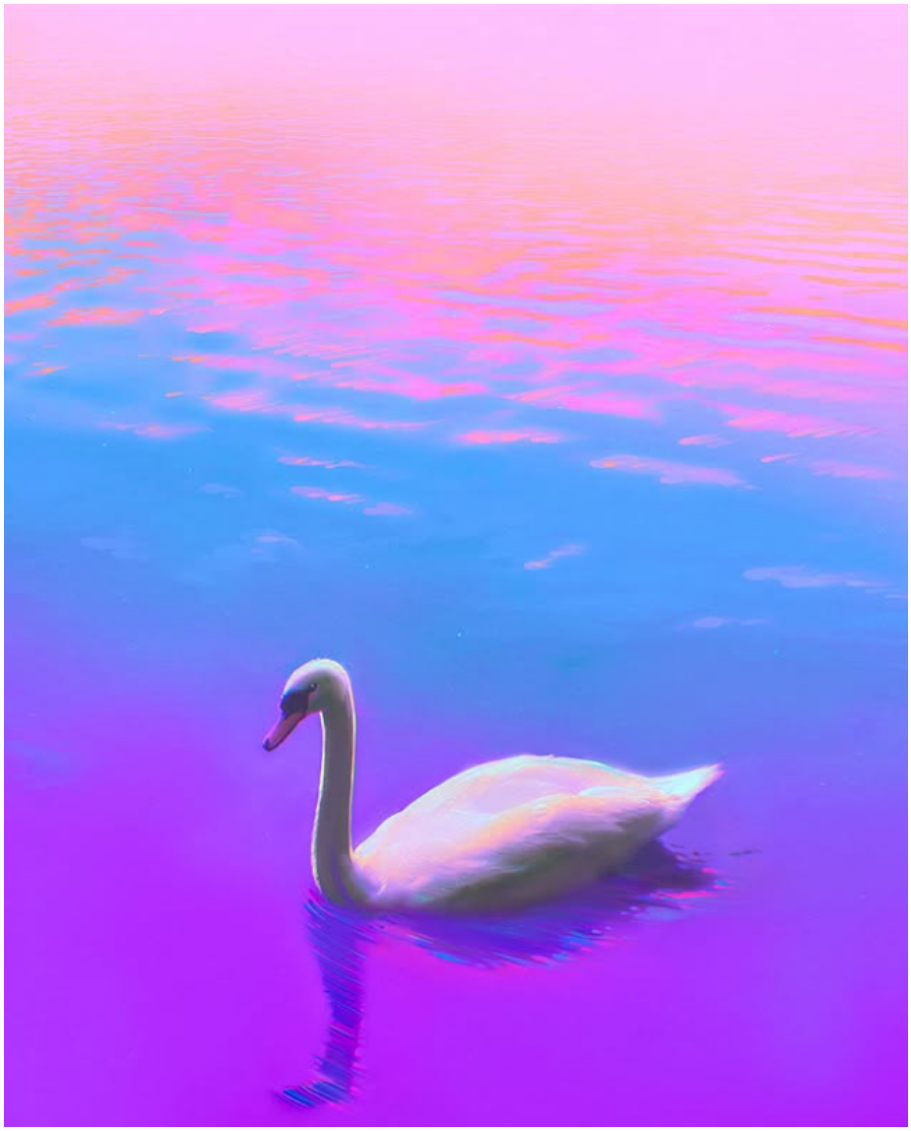


Figure 8: Example artwork by Ilya Kuvshinov (artist 7).

Artist 7, Kuvshinov Ilya, is a Russian artist and animation director. They gained online fame via Instagram by posting an artwork almost every day, totalling 3,370 posts as of writing! In our dataset, however, we are only using a measly 1,025 artworks.

Their work is often surreal, minimalist, and painterly with a soft digital aesthetic. The lack of detail is also a practicality to meet their goal of posting an artwork almost every day.

In the artwork shown above, the scene is centered on a solitary swan gliding across water, with heavy emphasis on reflection and negative space. Rather than texture and detail, the viewer is captured by atmosphere and color harmony.

However, if you look at the top right of the box-and-whisker plot, Ilya has produced an outlier with an edge density score of 0.344! Let's take a look at it.

Figure 9: Outlier artwork by Ilya Kuvshinov.

This stands out among his works. Instead of using the most detail in the focal area and a simple background, in this piece, attention is drawn to the face by giving every other area a striped pattern.

4.2 Colorfulness Vizualization

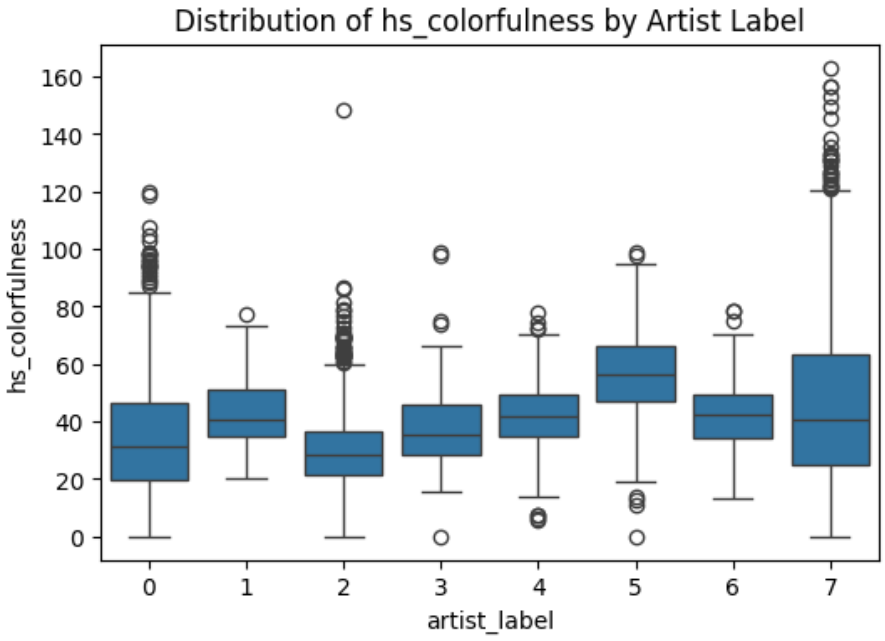


Figure 10: Distribution of Colourfulness values across artists.

For Hasler and Süsstrunk's Colorfulness Metric, we find artist 5 to have the clear highest average, while artists 0 and 2 are a little lower than the rest. Interestingly, we see artists 0, 2 and 7 having a large number of outliers, indicating their willingness to experiment with very colorful works once in a while, even though their average is lower than artist 5.

Figure 11: Example artwork by fuzichoco (artist 5).

Artist 5, known online as fuzichoco, is known for their highly detailed, character-driven illustration in a decorative anime/fantasy style.

In this example artwork, the artist combines jewel tones (reds, blues, golds) with high contrast and saturation. It is dense with visual elements — flowers, butterflies, ornate accessories, and patterned textiles — to frame a central figure.

The attention to detail in colorful elements make it clear why their average colorfulness score is the highest in our dataset.

Check out more of fuzichoco's work on their Instagram page!

Figure 12: Example artwork by Yukien (artist 2).

Artist 2 is another popular artist online known as Yukien or Kusakashi, with half a million followers on Twitter/X. While their recent works have increased in detail and color complexity, the average colorfulness of their portfolio is brought down by many of their older, simpler artworks.

Out of the 8 artists we have chosen, their style is the most **descriptive**, as opposed to **expressive**. This means that for this artist, telling a story by communicating characters' expressions and body language is more important than aiming for a beautiful, artistic illustration.

The example artwork above shows their typical muted palette and flat (cel) shading, without saturated primary colors. Interest is drawn to the expressive linework, typically in black. The ground tiles are neutral olive-beige, helping the character subtly stand out without sharp background separation.

By coloring a cartoonish drawing with a realistic color palette, viewers are able to connect better with the characters in the illustration while still feeling a sense of realism.

4.3 Principal Component Analysis (PCA)

We further analyse the extracted features using PCA. We find that artist 2 (green) stands out clearly—it has a distinct cluster that is more dispersed and offset from the others, indicating its features are notably different. The remaining artists show more overlap, suggesting that their style features are less distinguishable (at least in two dimenstions).

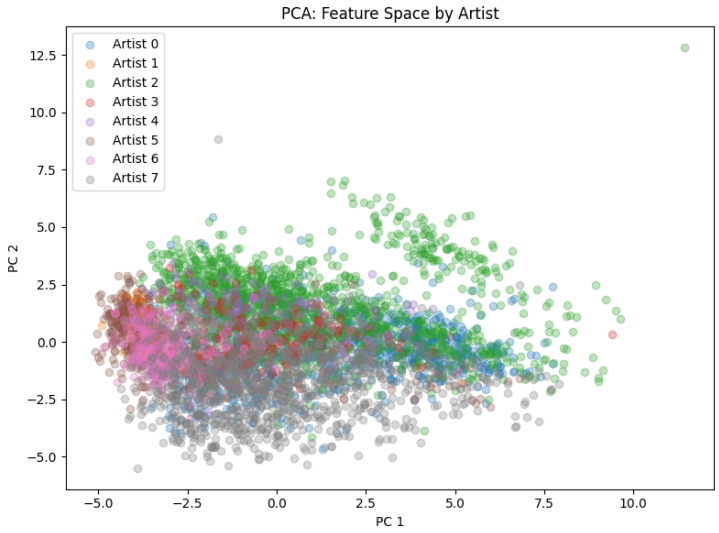


Figure 13: PCA Feature Space for the top 2 PC axes.

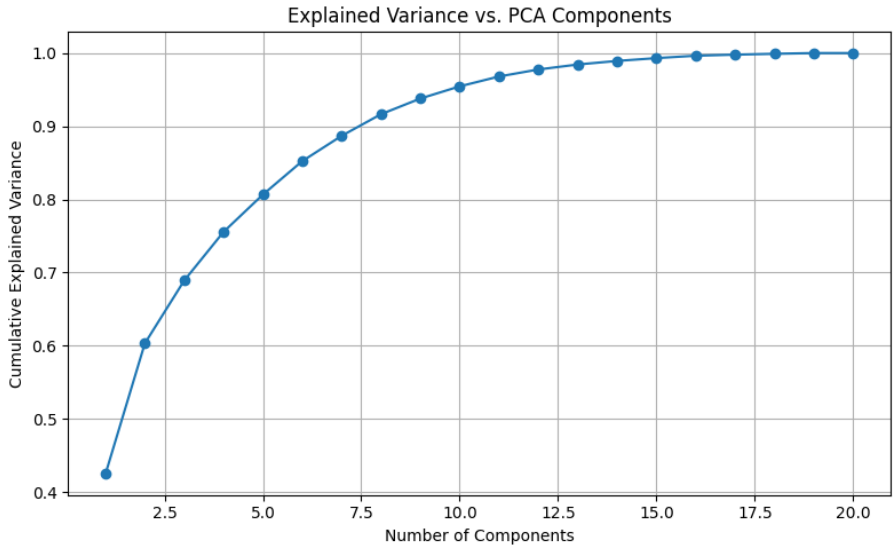


Figure 14: Cumulative explained variance for PCA components.

By the 10th component, over 95% of the total variance is captured, suggesting that we can reduce dimensionality significantly (e.g., to 10–12 dimensions) without losing much information.

5 Problem and Algorithm/Model

5.1 Problem Statement

With the rise of digital art platforms, vast collections of artworks are now shared online. Each artist has a distinct visual identity, but manually identifying these styles can be challenging and subjective. This project seeks to leverage machine learning to automate the classification of artistic styles based on visual features.

Our main objective is to develop a model that can:

- Detect **unique visual characteristics** in artworks
- Accurately **classify artworks by artist**
- Analyse which **stylistic elements** contribute most to recognizability

We frame this as a supervised image classification task: given an image of an artwork, predict the artist who created it. This problem is relevant to areas such as digital archiving, personalized recommendations, and the study of visual aesthetics. It also provides insight into how style can be quantified and distinguished by computational means.

5.2 Developing the Model

We used a reduced dataset of only three artists (0, 1 and 2) to choose an architecture for our model, before training it on the full dataset. Using this reduced dataset, we start with a naive linear model, and slowly add complexity and measure the accuracy at each step.

- **Naive model:** Achieved accuracy of 13%.
- **Extracted features model:** Achieved accuracy of 33%.
- **Basic CNN:** Achieved accuracy of 90%.
- **Finetuning CNN from VGG16:** Achieved accuracy of 93%.
- **Hybrid CNN using extracted features (Final Model):** Achieved accuracy of 96%.

5.2.1 Naive Model

We start by looking at how a naive model performs, with two hidden layers and ReLU activation. Without any convultion layers, we simply flatten the image into a single vector as the first step. The code can be found in `model_naive.ipynb`.

```
def make_model(input_shape, num_classes):
    inputs = keras.Input(shape=input_shape)
    x = layers.Rescaling(1.0 / 255)(inputs)
    x = layers.Flatten()(x)
    x = layers.Dense(392, activation="relu")(x)
    x = Dropout(0.3)(x)
    x = layers.Dense(196, activation="relu")(x)
    x = Dropout(0.3)(x)
    activation = "softmax"
    units = num_classes
    outputs = layers.Dense(units, activation=activation)(x)
    return keras.Model(inputs, outputs)
```

	precision	recall	f1-score	support
Class 0	1.00	0.02	0.03	63
Class 1	0.12	1.00	0.22	30
Class 2	0.00	0.00	0.00	152
accuracy			0.13	245
macro avg	0.37	0.34	0.08	245
weighted avg	0.27	0.13	0.03	245

With this poor accuracy, it is obvious that a standard neural network will not be able to accomplish the task.

5.2.2 Extracted Features model

We also look at how a model that only uses our extracted features (a vector of length 20) performs, dropping the image as an input. The code can be found in `model_metrics_only.ipynb`.

```
# Using only extracted features as input
model = tf.keras.Sequential([
    tf.keras.layers.Dense(256, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

	precision	recall	f1-score	support
Class 0	0.89	0.65	0.75	188
Class 1	0.10	0.92	0.19	39
Class 2	0.00	0.00	0.00	260
accuracy			0.33	487
macro avg	0.33	0.53	0.31	487
weighted avg	0.35	0.33	0.31	487

It does not perform much better, with Class 2 not being predicted by the model at all.

5.2.3 Basic CNN

We move from a basic neural network to one that uses convolutional layers. 3 Convolutional layers with increase number of filters are used, followed by pooling layers each. After the last convolution, the output is flattened and passed through a final fully connected later. The code can be found in `model_cnn_basic.ipynb`.

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
```

```
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),

    tf.keras.layers.Dense(num_artists, activation='softmax')
])
```

	precision	recall	f1-score	support
Class 0	0.90	0.87	0.89	188
Class 1	0.88	0.74	0.81	39
Class 2	0.90	0.95	0.92	260
accuracy			0.90	487
macro avg	0.89	0.85	0.87	487
weighted avg	0.90	0.90	0.90	487

The accuracy jumps to an acceptable 90%. After this, we attempted to use data augmentation to increase the size of the dataset, but it did not result in any increase in accuracy. The code for this can be found in `model_cnn_dataaugment.ipynb`.

```
# Data augmentation to improve generalization
datagen = ImageDataGenerator(
    rotation_range=0,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

	precision	recall	f1-score	support
Class 0	0.93	0.86	0.90	188
Class 1	0.93	0.64	0.76	39
Class 2	0.87	0.95	0.91	260
accuracy			0.89	487
macro avg	0.91	0.82	0.85	487
weighted avg	0.90	0.89	0.89	487

We also attempted hyperparameter tuning with `keras_tuner`, and saw a slight improvement to 92% accuracy, but this may just be a lucky sample after running 50 trials on different parameters. The code for this can be found in `model_cnn_hyperparam_tuning.ipynb`.

```
Trial 50 Complete [00h 01m 34s]
val_accuracy: 0.8172484636306763

Best val_accuracy So Far: 0.9240246415138245
Total elapsed time: 01h 49m 08s

Value             |Best Value So Far |Hyperparameter
96                |128               |conv_1_filters
256               |64                |conv_2_filters
128               |128               |conv_3_filters
256               |256               |dense_units
0.2               |0.4               |dropout_rate
0.00026403        |0.00035343        |learning_rate
4                 |10                |tuner/epochs
0                 |0                 |tuner/initial_epoch
1                 |0                 |tuner/bracket
0                 |0                 |tuner/round
```

5.2.4 Finetuning CNN from VGG16

Next, we attempt transfer learning by employing a pre-trained model. We choose to finetune VGG16 as it is known to work well as a feature extractor that transfers well to other image tasks. We freeze training of the base model, but replace the top layer with our own trainable fully connected layer. The code can be found in `model_cnn_vgg16.ipynb`.

```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(128, 128, 3))
base_model.trainable = False

model = tf.keras.Sequential([
    base_model,
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(3, activation='softmax')
])
```

```
              precision    recall  f1-score   support

     Class 0       0.90      0.94      0.92       188
     Class 1       0.93      0.95      0.94        39
     Class 2       0.95      0.92      0.93       260

    accuracy                           0.93       487
   macro avg       0.93      0.94      0.93       487
weighted avg       0.93      0.93      0.93       487
```

With an improvement to 93% accuracy, it is the best model we have so far, but it still does not implement the custom features that we extracted before, and is still a "black box".

5.3 Hybrid CNN using extracted features (Final Model)

We develop our final model after learning from the results of our experimentation. We create a multi-modal hybrid model that takes both an image and our extracted feature vector to make a prediction. We continue to use VGG16 as a base model, but combine the output of the base model with the extracted features vector, and finally a fully connected layer and softmax. The code can be found in `model_cnn_vgg16_hybrid.ipynb`.

```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(128, 128, 3))
base_model.trainable = False

# Image input branch
image_input = Input(shape=(128, 128, 3), name='image_input')
x = base_model(image_input)
x = Flatten()(x)

# Extracted features input branch
additional_input = Input(shape=(X_additional.shape[1],), name='additional_input')
y = Dense(64, activation='relu')(additional_input)

# Concatenate image features with the processed additional features
z = Concatenate()([x, y])

# Fully connected layers for classification
z = Dense(128, activation='relu')(z)
z = Dropout(0.5)(z)
output = Dense(num_artists, activation='softmax')(z)

# Define the complete model
model = Model(inputs=[image_input, additional_input], outputs=output)
```

Layer (type)	Output Shape	Param #	Connected to
image_input (InputLayer)	(None, 128, 128, 3)	0	-
vgg16 (Functional)	(None, 4, 4, 512)	14,714,688	image_input[0][0]
additional_input (InputLayer)	(None, 20)	0	-
flatten (Flatten)	(None, 8192)	0	vgg16[0][0]
dense (Dense)	(None, 64)	1,344	additional_input…
concatenate (Concatenate)	(None, 8256)	0	flatten[0][0], dense[0][0]
dense_1 (Dense)	(None, 128)	1,056,896	concatenate[0][0]
dropout (Dropout)	(None, 128)	0	dense_1[0][0]
dense_2 (Dense)	(None, 8)	1,032	dropout[0][0]

Figure 15: Final Model Architecture.

```
              precision    recall  f1-score   support

     Class 0       0.96      0.94      0.95       188
     Class 1       0.97      0.95      0.96        39
     Class 2       0.95      0.97      0.96       260

    accuracy                           0.96       487
   macro avg       0.96      0.95      0.96       487
weighted avg       0.96      0.96      0.96       487
```

We find that the accuracy is even better than before, jumping to 96%! The increase in accuracy after utilizing the extracted features proves that the metrics intuitively used to separate art styles are indeed useful from a computational standpoint. We describe the evaluation strategy in detail in the next section, where we train a model on the full dataset of 8 artists.

6 Evaluation Methodology

The effectiveness of the proposed algorithm is evaluated using a systematic approach, including a train-test split, performance metrics, and comparison with baseline models.

6.1 Train-Test Split

We divide the dataset into training and testing sets to assess the model's generalization ability. A typical 80-20 split is applied to the data:

```
X_train_images, X_test_images, X_train_additional, X_test_additional,
y_train, y_test = train_test_split(X_images, X_additional, y, test_size=0.2,
    ↪ random_state=1)
```

This ensures that the model is trained on a subset of the data and evaluated on unseen data to gauge its performance effectively.

6.2 Evaluation Metrics

The model's performance is measured using various evaluation metrics, including accuracy, precision, recall, and F1-score. The classification report provides a comprehensive evaluation of the model's performance across multiple classes:

```
report = classification_report(y_true, y_pred, target_names=
    [f"Class␣{i}" for i in np.unique(y_true)])
```

These metrics help to determine how well the model distinguishes between different artists.

- **Accuracy:** The proportion of total predictions that are correct.
- **Precision:** The proportion of positive predictions that are actually correct.
- **Recall:** The proportion of actual positives that were correctly predicted.
- **F1-score:** The harmonic mean of precision and recall, balancing both.

6.3 Comparison with Existing Models

To assess the effectiveness of our model, we compare its performance with existing models and baseline approaches. We consider the state-of-the-art baseline as `cnn_vgg16`, which applies transfer learning to the VGG16 model.

We compare our proposed model `cnn_vgg16_hybrid`, that incorporates both image features and additional extracted features, to `cnn_vgg16`. Both models will be trained with the same dataset, the same batch size (32) and the same epoch length (10).

7 Results and Discussion

7.1 Finetuning CNN from VGG16 (8 artists)

Result summary of finetuning the baseline VGG model without any extracted features, achieving an overall accuracy of 85%.

	precision	recall	f1-score	support
Class 0	0.89	0.78	0.83	195
Class 1	0.92	0.85	0.88	40
Class 2	0.88	0.92	0.90	259
Class 3	0.88	0.81	0.84	43
Class 4	0.96	0.69	0.80	35
Class 5	0.68	0.83	0.75	72
Class 6	0.81	0.82	0.82	62
Class 7	0.85	0.90	0.87	201
accuracy			0.85	907
macro avg	0.86	0.83	0.84	907
weighted avg	0.86	0.85	0.85	907

7.2 Hybrid CNN (8 artists)

Result summary of our proposed Hybrid CNN model, achieving an overall accuracy of 89%. Even with additional artists, our feature extraction improves upon the state-of-the-art by a significant margin.

	precision	recall	f1-score	support
Class 0	0.87	0.88	0.87	195
Class 1	0.95	0.90	0.92	40
Class 2	0.89	0.94	0.92	259
Class 3	0.88	0.84	0.86	43
Class 4	1.00	0.54	0.70	35
Class 5	0.73	0.85	0.79	72
Class 6	0.91	0.82	0.86	62
Class 7	0.93	0.93	0.93	201
accuracy			0.89	907
macro avg	0.90	0.84	0.86	907
weighted avg	0.89	0.89	0.89	907

7.3 UI Implementation

To complement model evaluation, a user interface (UI) was developed using Streamlit to provide an interactive experience. The UI enables users to upload an artwork and receive predictions on the most likely artist behind the style, along with visual explanations and feature breakdowns.

The trained hybrid CNN model (from `model_cnn_vgg16_hybrid.ipynb`) was saved using:

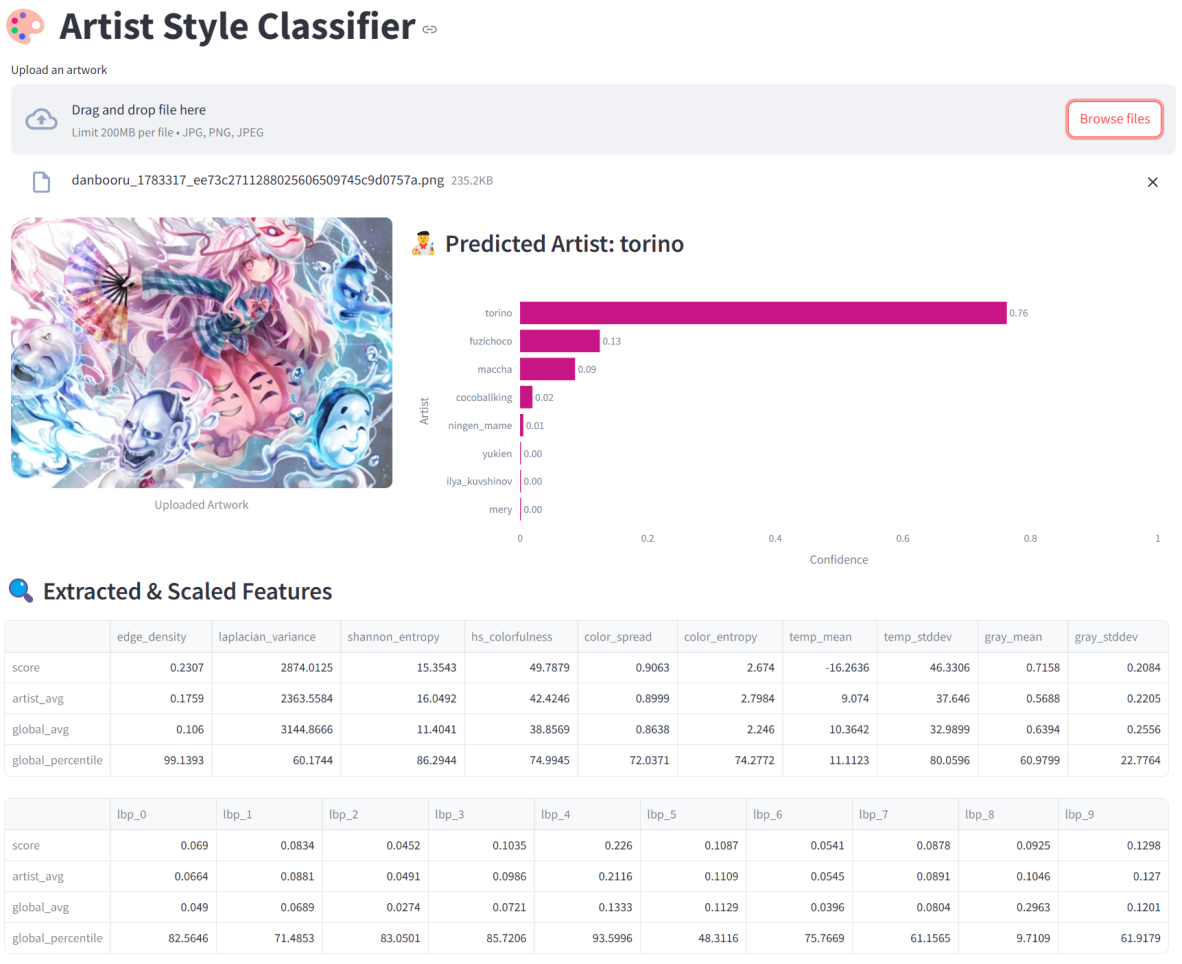
```
model.save("model.h5")
joblib.dump(scaler, "scaler.pkl")
```

Upon uploading an image, it is resized and normalized before being passed to the CNN model. Simultaneously, handcrafted features are extracted from the image using a custom `feature_extraction` module and scaled using the pre-fitted scaler.

Predictions are visualized through a horizontal bar chart displaying the confidence levels for each artist class. The UI also presents extracted and scaled features, compared against both global and artist-specific averages, along with percentile ranks derived from the dataset.

The app is launched using the command:

```
streamlit run ./app.py
```



	edge_density	laplacian_variance	shannon_entropy	hs_colorfulness	color_spread	color_entropy	temp_mean	temp_stddev	gray_mean	gray_stddev
score	0.2307	2874.0125	15.3543	49.7879	0.9063	2.674	-16.2636	46.3306	0.7158	0.2084
artist_avg	0.1759	2363.5584	16.0492	42.4246	0.8999	2.7984	9.074	37.646	0.5688	0.2205
global_avg	0.106	3144.8666	11.4041	38.8569	0.8638	2.246	10.3642	32.9899	0.6394	0.2556
global_percentile	99.1393	60.1744	86.2944	74.9945	72.0371	74.2772	11.1123	80.0596	60.9799	22.7764

	lbp_0	lbp_1	lbp_2	lbp_3	lbp_4	lbp_5	lbp_6	lbp_7	lbp_8	lbp_9
score	0.069	0.0834	0.0452	0.1035	0.226	0.1087	0.0541	0.0878	0.0925	0.1298
artist_avg	0.0664	0.0881	0.0491	0.0986	0.2116	0.1109	0.0545	0.0891	0.1046	0.127
global_avg	0.049	0.0689	0.0274	0.0721	0.1333	0.1129	0.0396	0.0804	0.2963	0.1201
global_percentile	82.5646	71.4853	83.0501	85.7206	93.5996	48.3116	75.7669	61.1565	9.7109	61.9179

Figure 16: Screenshot of the User Interface for Artwork Classification

7.4 Discussion of Results

The results of our experiments highlight the effectiveness of incorporating handcrafted features into the artist classification task. Our initial baseline, which involved fine-tuning a VGG16 model without any additional features, achieved a respectable accuracy of 85%. This demonstrates the strong representational power of convolutional neural networks when applied to image data alone.

However, we observed a notable performance improvement when we included extracted features: edge density, Laplacian variance, Shannon entropy, HS colorfulness, color spread, color entropy, temporal mean and standard deviation, grayscale statistics, and local binary patterns (LBP 0–9). These features capture important low-level image characteristics that are complementary to those learned by the CNN.

Although it is difficult to pinpoint a single most informative feature, the collective contribution of these handcrafted features helped the model better differentiate between artist styles, especially for subtle variations not easily captured by deep learning alone.

Guided by these insights, we developed a hybrid multi-modal architecture that takes both the image and its corresponding extracted feature vector as input. This final model architecture, implemented in `model_cnn_vgg16_hybrid.ipynb`, uses VGG16 as the base image encoder. The output of the CNN is concatenated with the feature vector and passed through a fully connected layer followed by a softmax activation for classification.

Our hybrid model achieved an improved overall accuracy of 89%, even when more artists were added to the dataset. This result demonstrates that the integration of statistical features with deep visual representations leads to a significant improvement in classification performance and robustness, setting a new benchmark for artist style classification in our dataset.

These findings suggest that a hybrid approach combining deep learning with domain-specific features can be a powerful strategy in other image classification tasks where subtle stylistic or structural variations matter.

From an artist's perspective, the results also prompt reflection on how we recognize artistic styles. It is often difficult to articulate why a particular piece feels like it belongs to a certain artist — the reasoning is usually based on a vague sense of "vibe," use of color, or how features like faces are drawn. This ambiguity opens interesting directions for future exploration. For example, we could crop to just the facial region and evaluate whether the model can still correctly attribute the artwork, which may reveal which visual components carry the strongest stylistic signals.

For future research, incorporating a large multimodal model (LMM) to explain the prediction in natural language could enhance interpretability. When explanations are verbalized, they offer a clearer framework for analytical artists who seek structure and guidance to emulate or improve upon the style of their favorite artists.

References

[1] Seiichi Uchida. Image processing and recognition for biological images. *Development, growth differentiation*, 55, 04 2013.

[2] David Hasler and Sabine Suesstrunk. Measuring colourfulness in natural images. *Proceedings of SPIE - The International Society for Optical Engineering*, 5007:87–95, 06 2003.

[3] Olli Lahdenoja, Jonne Poikonen, and Mika Laiho. Towards understanding the formation of uniform local binary patterns. *International Scholarly Research Notices*, 2013(1):429347, 2013.