CS 548 Knowledge Discovery in Databases: Classification Project

Artem Frenk
Department of Data Science
Worcester Polytechnic Institute
afrenk@wpi.edu

Siddhartha Pradhan
Department of Data Science
Worcester Polytechnic Institute
sppradhan@wpi.edu

October 18, 2024

Abstract

In this project, we explore the WikiArt dataset to classify artwork based on its style using various machine learning models. The dataset, which contains high-resolution images and corresponding metadata such as artist, genre, and style, presents significant challenges related to memory usage, class imbalance, and image preprocessing. We performed an extensive exploratory data analysis (EDA) to investigate image properties, including image size, brightness distribution, and RGB channel distributions, to inform our preprocessing decisions. Preprocessing steps such as resizing, grayscale conversion, and data augmentation were employed to standardize the data. We then applied five classification models: Gaussian Naive Bayes, Support Vector Machines (SVM), K-Nearest Neighbors (KNN), XGBoost, and a Convolutional Neural Network (CNN). Hyperparameter tuning and model optimization strategies were explored, including grid search and cross-validation. Among all models, the CNN achieved the best performance, significantly outperforming traditional machine learning algorithms. This project highlights the importance of deep learning architectures for complex image classification tasks and identifies areas for further improvements in model tuning and resource optimization.

Exploratory Data Analysis

The WikiArts dataset is fairly large and requires large amounts of memory. This was mainly due to the high-resolution images included in the dataset. We opted for a map-reduce framework built into the HuggingFace datasets library for EDA and preprocessing. This enabled us only to load a portion of the dataset into memory. To perform some initial exploratory data analysis we examined the following data attributes:

1. Sample of Images + Styles

To explore the dataset, we first randomly sampled the dataset to explore the image data and the respective target variables, i.e. style. Five different observations have been given in Figure 1

2. Image Size

To better understand the input dimensions for our models, we checked whether the images were of the same size, and had the same aspect ratio. Figure 2 shows the distribution of the image widths and heights. The images do not have the same size and have different aspect ratios (i.e ratio of height and width).

3. Image Brightness Distribution

Next, to understand the distribution of the pixel values in the images, we examined the average pixel values for each image (aggregated across different

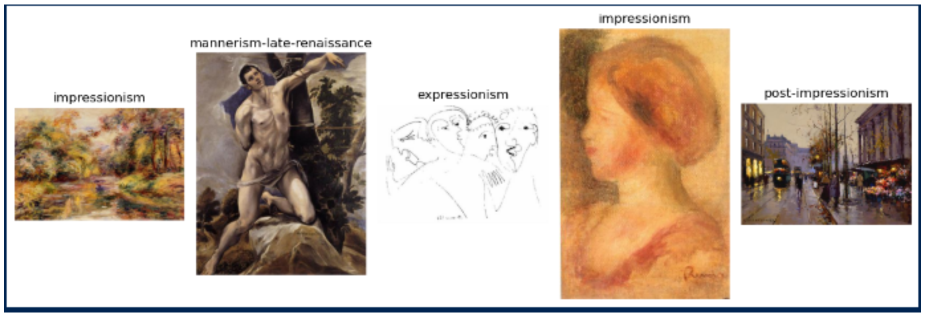


Figure 1: Sample of Images + Styles

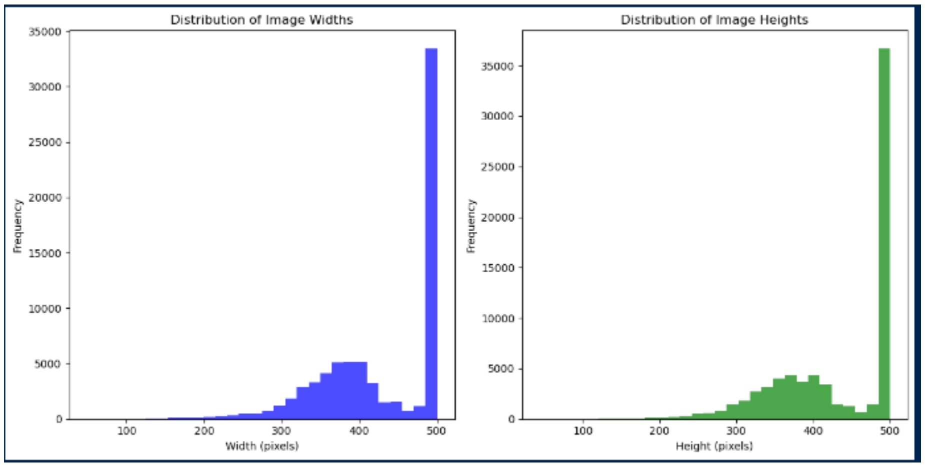


Figure 2: Image Width and Height Distributions

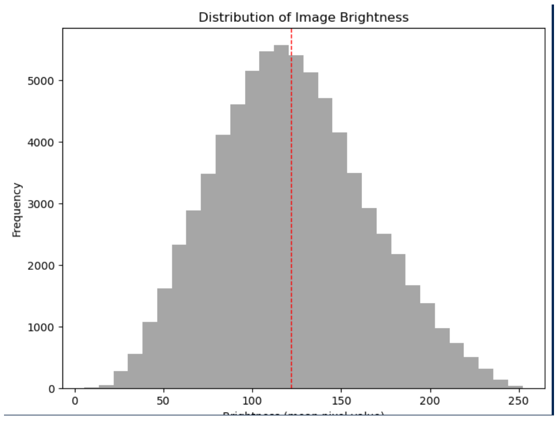


Figure 3: Image Brightness Distribution

image channels). Based on Figure 3, the image brightness follows a normal distribution with mean around 125. This result can be used to normalize the pixel values during the preprocessing phase.

4. Color Distribution (RGB Distribution)

Next, we examined the individual distributions of the three channels. Figure 4 shows the first 5 images in the train set and their respective RGB distributions. This analysis was performed to assess the trade off between the computational cost of training with 3 image channels and additional variation present in the 3 channels. In other words, we want to examine whether converting the images to black and white would be appropriate to simplify the model. Based on the results and given the high computational cost of training models with a large number of parameter (number of params for a single observation = Width * Height * number of channels), we chose to convert the images to black and white.

5. Class Imbalance

Since our target variable is the style of the image, we explored the distribution of the target categorical variable. Figure 5 suggests there is a large imbalance for the number of observations in each class. Image style of impressionism had the highest number of observations (n=11820), and action painting had the lowest (n=150). A potential solution to class imbalance is to combine categories with lower frequencies into a single ‘other’ category. However, for this initial project we chose to predict all classes regardless of their frequencies, as it is more realistic of real-world scenarios.

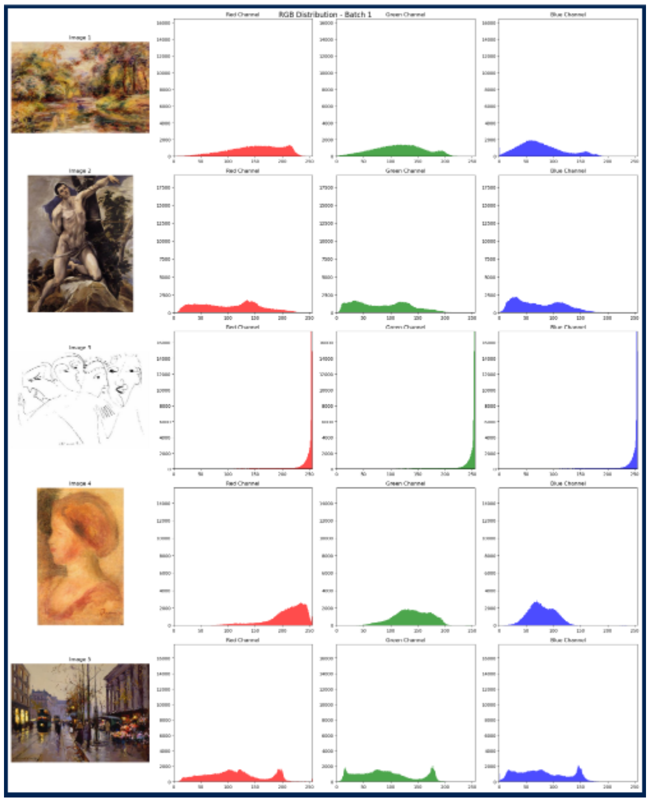


Figure 4: RGB Distribution (First 5 Entries)

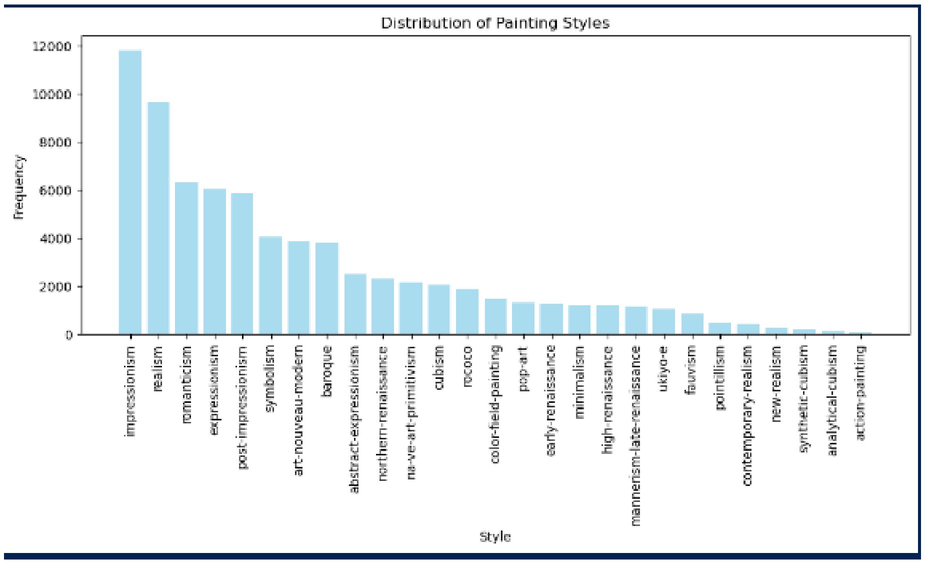


Figure 5: Class Imbalance

Problem Definition

Our dataset of choice is the `wikiart` dataset. This dataset features image data (`image`), with the name of the artist who drew them (`artist`), the genre of the painting (`genre`), and the style of the painting (`style`). Our goal is to leverage a variety of classification models to classify the genre of an image. The genre has a strong historical basis. As per WikiArt's website:

> A genre system divides artworks according to depicted themes and objects. A classical hierarchy of genres was developed in European culture by the 17th century. ... This hierarchy was based on the notion of man as the measure of all things. Landscape and still life were the lowest because they did not involve human subject matter. History was highest because it dealt with the noblest events of humanity. [The] [g]enre system is not so much relevant for a [sic] contemporary art; there are just two genre definitions that are usually applied to it: abstract or figurative.

We feel that we can use the lofty definition of genre to discard it entirely as a feature, and instead focus on the somewhat more objective metric, `style`. The website's style page states that

> A style of an artwork refers to its distinctive visual elements, techniques and methods. It usually corresponds with an art movement or a school (group) that its author is associated with.

Due to considerations of both performance and time constraints, we chose to sample a subset of the original dataset to allow for conclusions to be made. We still aim to target `style`, which contains the same amount of classes for styles (See Fig 1.), but discards the artist names and genres, which makes our objective simpler.

Feature	**Unique Values**
`Image`	81,444
`artist`	129
`genre`	11
`style`	27

Feature	**Unique Values**
`image`	73300
`text`	30133
`style`	27
`name`	73300
`gen_style`	179

Task 4: Preprocessing

The following preprocessing steps were performed on the dataset to ensure it is primed and ready for model training:

1. Data Cleaning

In this step, we aimed to address potential inconsistencies in the dataset, such as missing values, anomalies, and duplicate entries. While the dataset used was loaded from Hugging Face's `load_dataset` function, which handles these inconsistencies. Data had no missing features or duplicate images. However, image sizes featured several anomalies, which were handles in our transformation and normalization step.

2. Data Transformation and Normalization

All images were resized to a consistent size of 64x64 pixels to ensure uniformity in input dimensions across the dataset. Depending on the need, the images were also converted to grayscale using a flag. Grayscale conversion reduces data dimensionality, which can be beneficial for models that do not require color information. Additionally, the pixel values were normalized to a range between 0 and 1 to prevent models from being influenced by pixel intensity differences, thus ensuring smoother and more consistent training performance.

3. Data Augmentation

Though not enabled by default in the preprocessing script, data augmentation was incorporated as an optional flag. Techniques like random flipping, rotations, brightness adjustment, and contrast variations were applied to enhance the dataset and prevent overfitting. This process artificially increases dataset size by creating multiple versions of the same image, which ultimately improves the model's generalization capability to unseen data.

4. Label Encoding

To prepare the categorical target variable, two types of label encodings were applied. One-hot encoding was used for training convolutional neural networks (CNNs) to allow the network to predict probabilities for each class. For other models, integer encoding was applied to map the style of each artwork into numeric labels. This label encoding ensures that the categorical nature of the target variable is properly represented for model training.

5. Preparing for CNN and Other Models

For CNN models, images were retained in their 3D form ($64 \times 64 \times 3$ or 64×64 for grayscale). This format allows CNNs to capture the spatial structure of images. For other models, images were flattened into 1D vectors to match the input requirements of models that do not process spatial data. This ensures that the dataset is prepared for training on various model types.

Model Selection, Training, and Optimization

For our data, we chose the following 5 models:

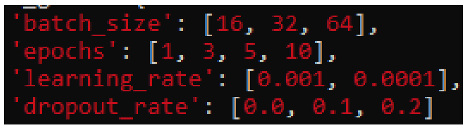
1. Gaussian Naive Bayes Classifier
2. Support Vector Machine + SGD
3. K-Nearest Neighbors (KNN) Classifier
4. XGBoost Classifier
5. Convolutional Neural Network

Table 2 outlines the performance of the base model, with default or best-guess hyperparameters:

The Gaussian Naive Bayes model is a probabilistic and simple model with no hyperparameters to tune. On the other hand, the SVM trained using Stochastic Gradient Descent had hyperparams such as learning rate (1e-5 or 1e-6) and regularization penalty (L1, L2, or elasticnet). Similarly, the KNN model uses hyperparameter K to determine the number of closest samples to consider for majority voting. We use K values 20, 50 and 100 during model tuning. Unlike

Model Type	Accuracy	Precision	Recall	F1 Score
Gaussian Naive Bayes	0.06	0.16	0.06	0.06
SVM (using SGD)	0.11	0.12	0.11	0.10
KNN (K=20)	0.15	0.19	0.15	0.14
XGBoost (Single Estimator)	0.21	0.16	0.21	0.15
CNN	0.33	0.31	0.33	0.31

Table 1: Results of the initial models on the test set (n=8145)



```
'batch_size': [16, 32, 64],
'epochs': [1, 3, 5, 10],
'learning_rate': [0.001, 0.0001],
'dropout_rate': [0.0, 0.1, 0.2]
```

Figure 6: CNN Hyperparameter Tuning Options for Grid Search & K-Fold CV

the other models, XGBoost and CNN have many hyperparameters to tune. For XGBoost, hyperparameters such as number of estimators (1, 5, 10), learning rate (0.1, 0.3), and regularization penalty strengths (alpha = [0, 0.01] and lambda = [0.9, 1]) were selected for tuning. In the case of the CNN, the hyperparameter tuning we performed was adjusting the batch size, number of epochs(iterations to run), learning rate, and the inclusion of a dropout layer (zeroing out a set percentage of neurons to prevent overfitting).

We chose three distinct methods for tuning and improving model performance:

1. Exhaustive Grid Search on a Validation set (size = 0.3 * train set size). This is a popular method for tuning hyperparameters as it uses a small validation set for tuning, resulting in less computational resources used.

2. Exhaustive Grid Search with 3-Fold Cross Validation. This is a method to tune hyperparameters that changes the validation and train set 3 times for each hyperparameter combination. While this process is extremely computational-resource intensive, it results in models that generalize better.

3. Alternative preprocessing step–Using RGB pixel data rather than Black and White image data. Instead of only using black and white images, we pass in all three color channels to the models (for CNN input size = 64, 64, 3 and for other models input size = 12288, i.e. 64*64*3)

Model Evaluation

For evaluation, we chose

1. Accuracy, defined as

$$\frac{\text{True Positive} \times \text{True Negative}}{\text{True Positive} \times \text{False Positive} \times \text{False Negative} \times \text{True Negative}}$$

2. Precision, defined as

$$\frac{\text{True Positive}}{\text{True Positive} \times \text{False Positive}}$$

3. Recall, defined as

$$\frac{\text{True Positive}}{\text{True Positive} \times \text{False Negative}}$$

4. F1 Score, defined as

$$2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Model Type	**Optimization**	**Accuracy**	**Precision**	**Recall**	**F1 Score**
SVM_SGD	1	0.10	0.13	0.10	0.09
SVM_SGD	2	0.12	0.13	0.12	0.09
SVM_SGD	3	0.06	0.08	0.06	0.06
KNN	1	0.16	0.15	0.16	0.14
KNN	2	0.17	0.22	0.17	0.13
KNN	3	0.11	0.09	0.11	0.10
XGBoost	1	0.21	0.18	0.21	0.16
XGBoost	2	0.26	0.22	0.26	0.24
XGBoost	3	0.15	0.10	0.14	0.11
CNN	1	0.33	0.33	0.32	0.31

Table 2: Results of the models on the test set (n=8145)

Due to time constraints and lack of high powered computing resources, we were unable to complete training for K-Fold Cross Validation and Image Augumentation for the CNN, due both to an outdated CUDA version and mismatched drivers. Moving forward, measures will be taken to ensure that more robust pipelines are created to avoid memory and other computational issues. However, even with the inclusion of Grid Search CV for Hyperparameter Tuning, we saw only a slight increase in Precision relative to the "Vanilla" model, and slight decreases or no changes to the other evaluation metrics for the model. The optimal parameters were clocked at ['batch_size': 64, 'epochs': 5, 'learning_rate': 0.001, 'dropout_rate': 0.0]. The worst performersj were those that had a higher dropout rate, leading us to conclude that in the case of this CNN architecture on image data, the inclusion of a dropout layer negatively affected our model's performance on the test set. The highest observed F1 in a test that featured dropout was only 0.29. We also observe that smaller epoch sizes rightfully led to decreased accuracies, due to models not being able to fully fit to the data with less epochs. Even with optimizations, the CNN greatly outperformed all other models we tried, leading us to tentatively conclude that Deep Learning architecture is preferable for our task and dataset.

Code

The code can be found in this GitHub repository.

References

Ideas for EDA 1

Ideas for EDA 2

Ideas for EDA 3

Sci-Kit Learn KNN Hyperparameters

CNN Guide 1

XGBoost for Image Data

Waskom, M. L. (2021). Seaborn: statistical data visualization. Journal of Open Source Software, 6(60), 3021.

Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. Journal of Big Data, 6(1), 1-48.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13(Feb), 281-305.

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In ICLR.

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785-794).

- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.
- Dumoulin, V., & Visin, F. (2016). A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285.
- Nwankpa, C., Ijomah, W., Gachagan, A., & Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. arXiv preprint arXiv:1811.03378.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958.
- Ng, A. Y. (2004, July). Feature selection, L1 vs. L2 regularization, and rotational invariance. In Proceedings of the twenty-first international conference on Machine learning (p. 78).
- Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning (pp. 448-456). PMLR.
- Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.
- He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision (pp. 1026-1034).