Introduction and Background

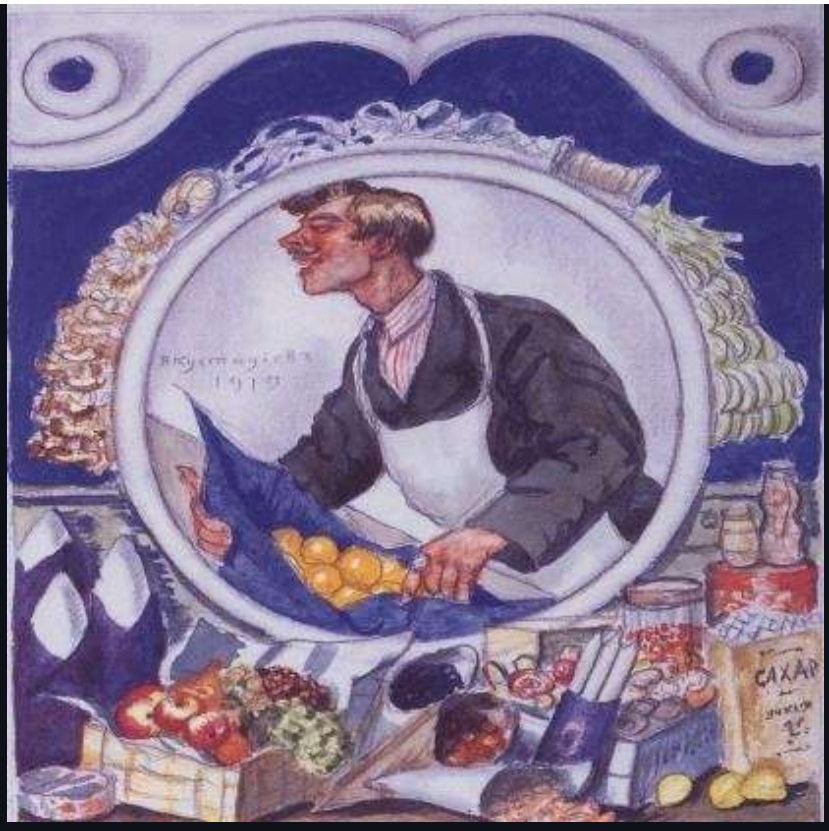
Classifying art by style is important for determining information about the artwork. Over centuries, changes in specific painting techniques have altered the visual style and genre of art. [1] Identifying the style of art through an automatic system is challenging as there are not single features that classify a specific style. Literature review has shown that style recognition through machine learning approaches have been performed with combining popular image features, including histograms of gradients, spatial envelopes, and discriminative color names. Yet, identifying the style has been difficult due to the curse of dimensionality, where adding more features does not improve accuracy [2]. The dataset will be Wikiart and Painting Dataset which have styles of impressionism, realism, romanticism, symbolism, and more.

Classified Styles


```
[
  0 : "Realism"
  1 : "Art_Nouveau_Modern"
  2 : "Analytical_Cubism"
  3 : "Cubism"
  4 : "Expressionism"
  5 : "Action_painting"
  6 : "Synthetic_Cubism"
  7 : "Symbolism"
  8 : "Ukiyo_e"
  9 : "Naive_Art_Primitivism"
  10 : "Post_Impressionism"
  11 : "Impressionism"
  12 : "Fauvism"
  13 : "Rococo"
  14 : "Minimalism"
  15 : "Mannerism_Late_Renaissance"
  16 : "Color_Field_Painting"
  17 : "High_Renaissance"
```

```
    18 : "Romanticism"
    19 : "Pop_Art"
    20 : "Contemporary_Realism"
    21 : "Baroque"
    22 : "New_Realism"
    23 : "Pointillism"
    24 : "Northern_Renaissance"
    25 : "Early_Renaissance"
    26 : "Abstract_Expressionism"
]
```

Abstract_Expressionism



Art_Nouveau_Modern

Baroque

Problem Definition

We will be solving the problem of art style indentification and classification. To approach this problem, we will gather data, preprocess, and use several ML algorithms to classify our data. To make our approach unique, we will combine these algorithms into an ensemble model for better performance. Our motivation behind this project is to gain insight into the difference of art styles through the use of ML algorithms and facilitate timely art classification.

Preprocessing Methods

Final Update

Since our midterm update, we have implemented two other models, SVM and CNN. Since neither of those require new preprocessing other than what we have already implemented and used, we did not add any new preprocessing for this section of our project.

Midterm Update

We have implemented several different preprocessing methods, including padding, pixel-value normalization, one-hot encoding, and finally PCA. Each method has a purpose behind us using it, which gives various benefits. We used padding to ensure that the spatial size of our input was preserved so our output will remain the same size after applying other methods. We used pixel-value normalization because it is important to have normalized values so that ML models that expect Gaussian distributions will run properly. One-hot encoding was chosen so that our categorical data is transformed into a format that our models will be able to use. Specifically, this allows each category to be treated independently. PCA was chosen to reduce our model complexity by reducing the number of features while preserving the most important ones. This also will help us to avoid overfitting.

Example of one-hot encoding: `Label 3 -> [0,0,1,0,0...]`

Image Resizing

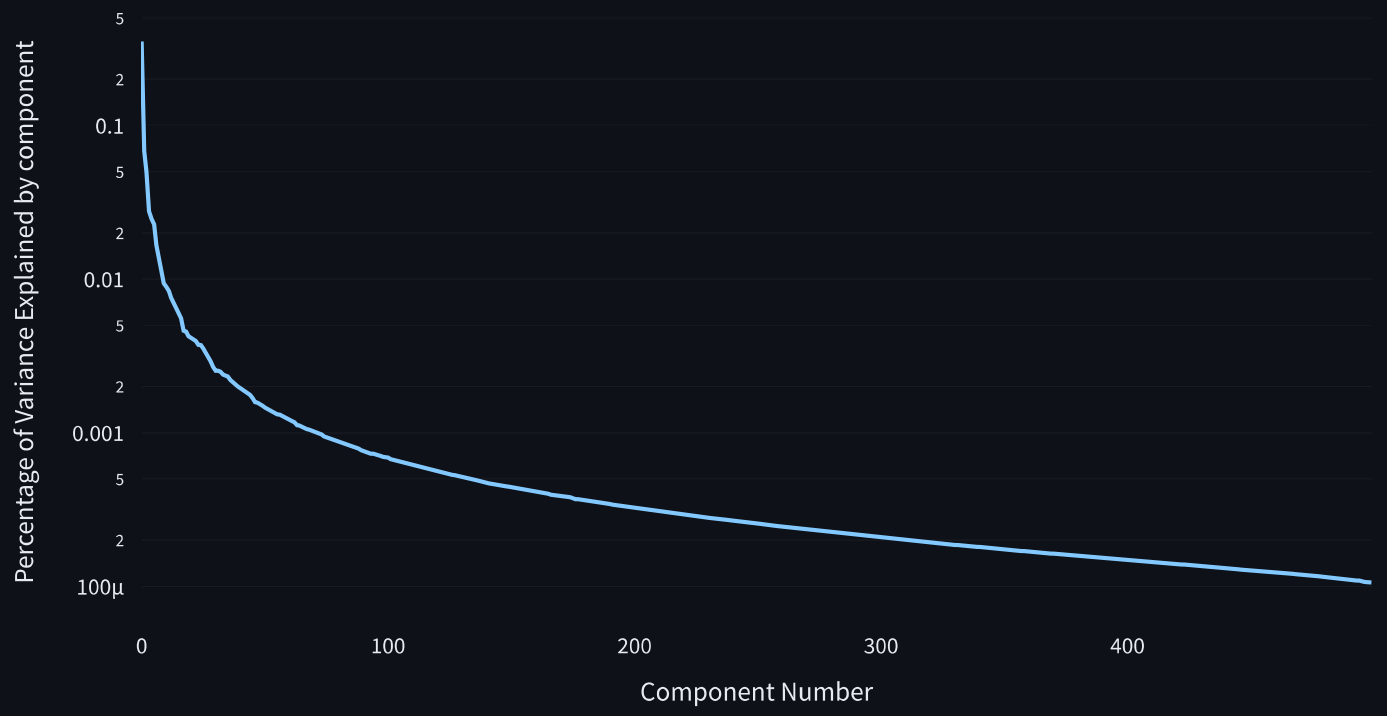
Raw image

Resized image

Normalized Pixel values on red channel of above image

	0	1	2	3	4	5	6	7	8	9	10	11
0	0.2863	0.2706	0.2039	0.2	0.1843	0.1882	0.1882	0.2588	0.349	0.4353	0.4627	0.462
1	0.3608	0.3137	0.2392	0.1725	0.1647	0.1608	0.1765	0.1686	0.251	0.4431	0.5137	0.470
2	0.3843	0.2941	0.2157	0.1529	0.1882	0.2275	0.2078	0.2784	0.4118	0.4941	0.4627	0.372
3	0.2196	0.1961	0.2196	0.2314	0.2196	0.2196	0.2392	0.2941	0.3882	0.4275	0.3843	0.30
4	0.2039	0.2078	0.2667	0.3569	0.3255	0.2588	0.2471	0.2471	0.3294	0.3922	0.4118	0.356
5	0.2941	0.3686	0.3961	0.4078	0.4118	0.3529	0.3059	0.2902	0.3216	0.3569	0.3843	0.34
6	0.2784	0.3569	0.3843	0.4	0.4118	0.4157	0.4157	0.3961	0.3059	0.2471	0.2627	0.345
7	0.2471	0.3451	0.4078	0.4314	0.4314	0.451	0.4157	0.3647	0.2353	0.2157	0.3137	0.396
8	0.3098	0.3216	0.3843	0.4745	0.4824	0.4667	0.4353	0.3961	0.2275	0.2314	0.3294	0.356
9	0.3569	0.3137	0.3451	0.4196	0.4196	0.4157	0.4196	0.4235	0.3059	0.3216	0.3765	0.407

PCA visualization



Converting from X-space to Z-space for first 100 features and components

	Unnamed: 0	Weight of Raw Feature 0	Weight of Raw Feature 1	Weight of Raw Feature 2	Weight of R
0	Component 0	0.0023	0.0023	0.0022	
1	Component 1	0.0013	0.0016	0.0016	
2	Component 2	0.0038	0.0038	0.0036	
3	Component 3	0.0018	0.0017	0.0014	
4	Component 4	0.0015	0.0002	-0.0015	
5	Component 5	-0.0005	0.0002	0.0013	
6	Component 6	0.0042	0.004	0.0036	
7	Component 7	0.0028	0.0032	0.003	
8	Component 8	-0.0016	-0.0016	-0.0015	
9	Component 9	0.0003	0.0002	0	

Proposal

We will use preprocessing methods such as PCA dimensionality reduction, pixel value normalization, one-hot encoding, and image adjustment. Image adjustment like padding will making sure the image dimensions match each other, so they have the same features size. Normalizing the pixel values is important as ML models expect data to follow a gaussian distribution. PCA is a technique for dimensionality reduction, which helps mitigate the curse of dimensionality.

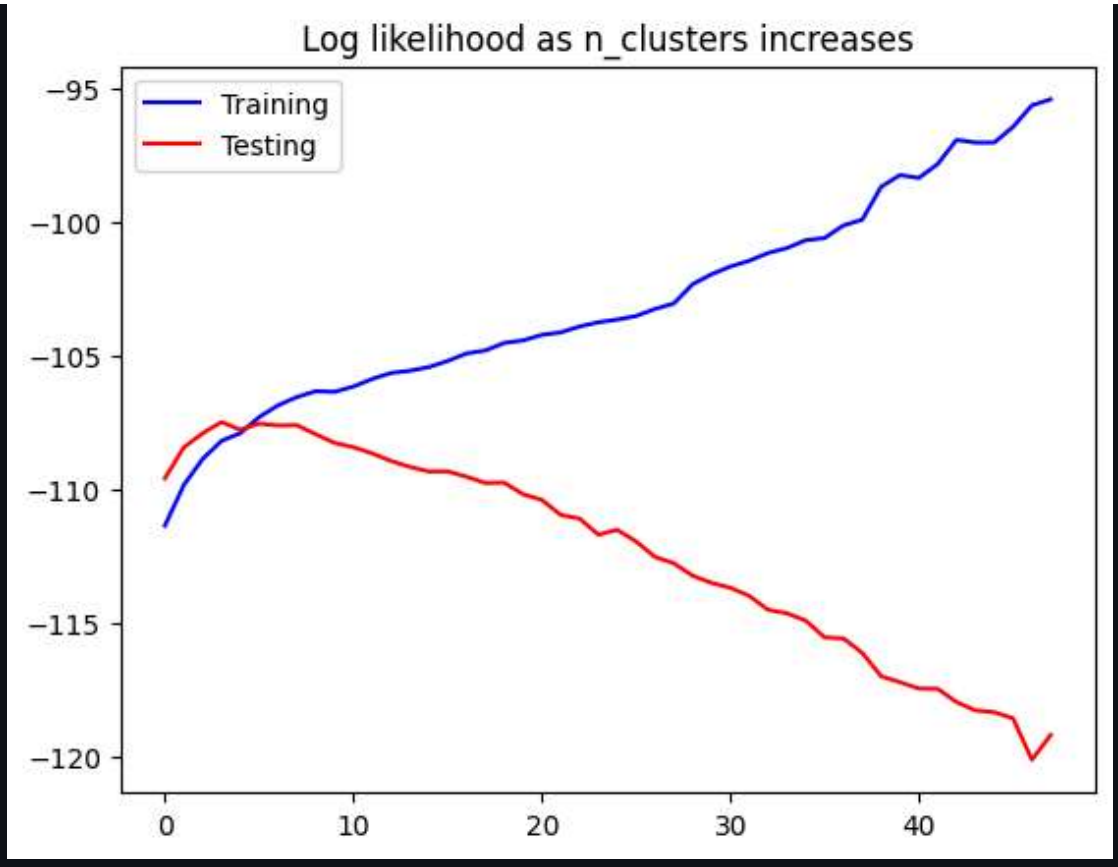
Machine Learning Methods

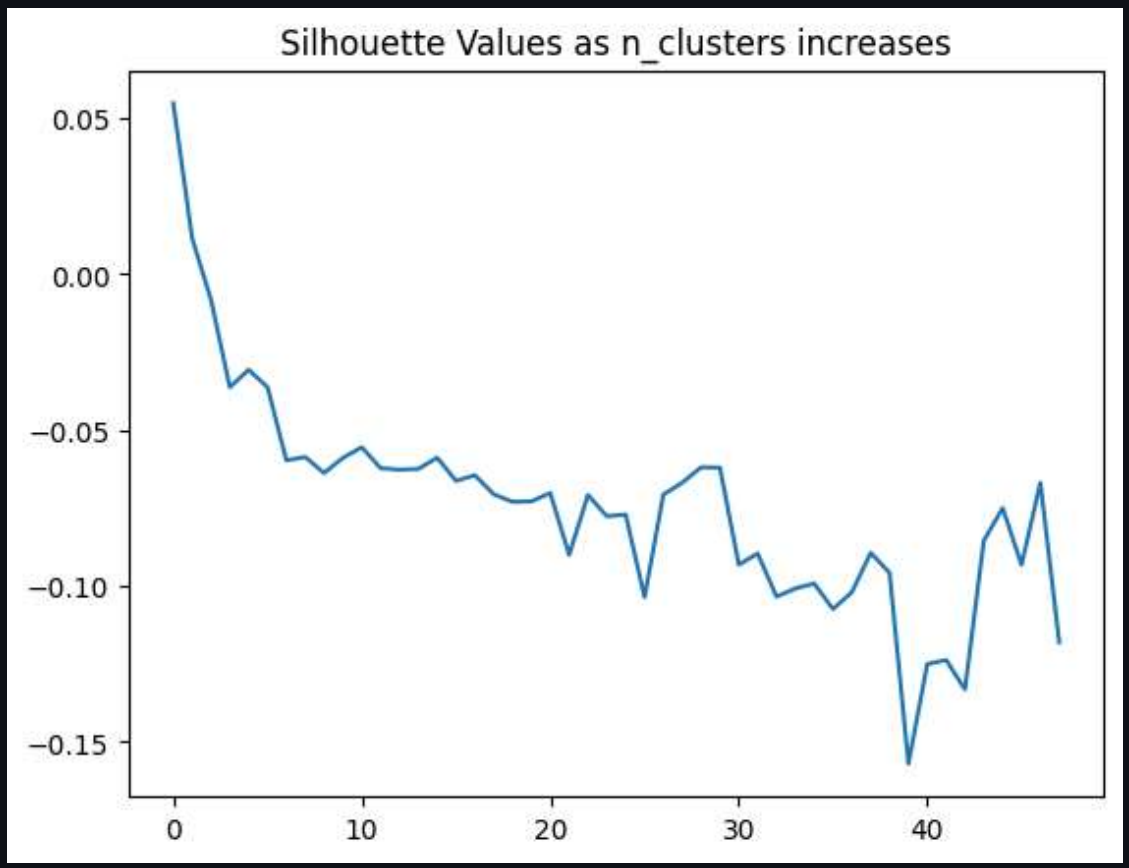
Final Update

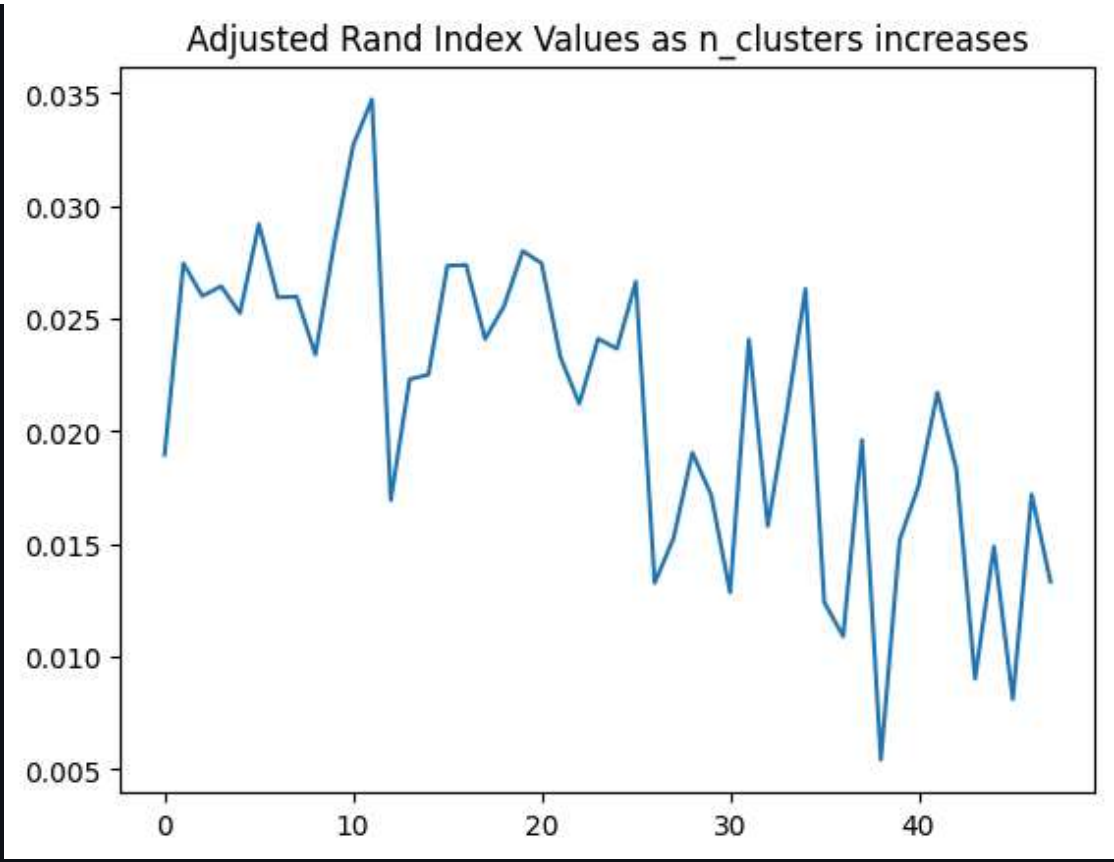
We implemented SVM and CNN for our final two models. We chose SVM for a number of reasons, the first of which was it is often used for image classification and gave us a baseline for supervised learning. Specifically, SVM can be used for handling large feature spaces, which is what we were dealing with for our images. We used RBF kernels to utilize this ability and bring our data into higher dimensionality to find a separating hyperplane. This was how we ensured our data was linearly separable. The decision boundaries for SVM are also very easy to interpret which is an added bonus. We chose CNN for several different benefits, as well. Firstly, it can extract relevant features automatically from our image data without us having to undergo manual feature engineering. It also has the ability to learn complex features by building up from simpler features through its use of several layers, and specifically for our purposes, it could capture detailed patterns in images. CNN also performs well against noise better than other methods. All in all, these two models were helpful to us in many ways and we had the above reasons to choose them.

Midterm Update

We chose to implement GMM for our first model. This model should help us to model data distributions, so we can capture variability and patterns in the art. We are using GMM with soft clustering which means our model assigns each pixel a probability of belonging to different clusters. This will give us a more nuanced understanding of data grouping as well as help us to understand how data is spread. GMM is an unsupervised model; later on we plan to use SVM, which is a supervised model. The following are some visualizations of GMM quantitative metrics including silhouette score, log likelihood, and adjusted rand index for different hyperparameter values of the GMM model (the number of clusters)

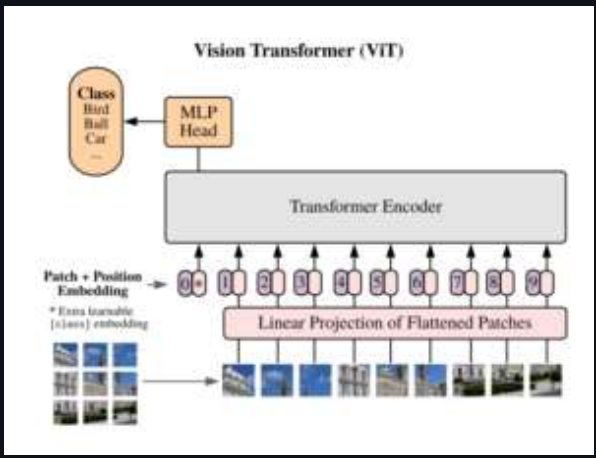

Log likelihood as n_clusters increases
Training
Testing
−95
−100
−105
−110
−115
−120
0
10
20
30
40


Silhouette Values as n_clusters increases
0.05
0.00
−0.05
−0.10
−0.15
0
10
20
30
40

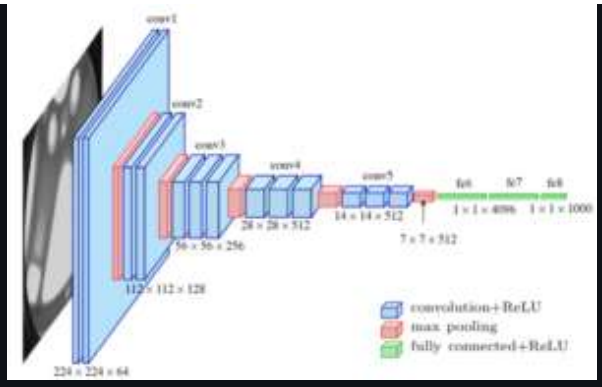


Proposal

GMMs will be useful for understanding how data is organized. SVMs is often used for image classification, so it is a starting point to work with supervised learning. Scikit-learn contains SVC. CNN models are promising state of the art approaches for this. [3] Pytorch has a class for CNN. Visual transformers perform well and are also state of the art for image classification. Also contained in pytorch.



Visual transformer diagram

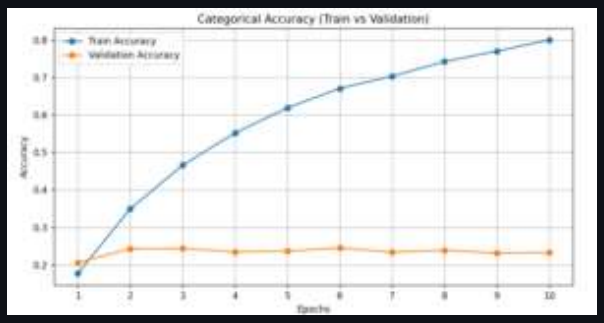


Convolutional Neural Net diagram

Results and Discussion

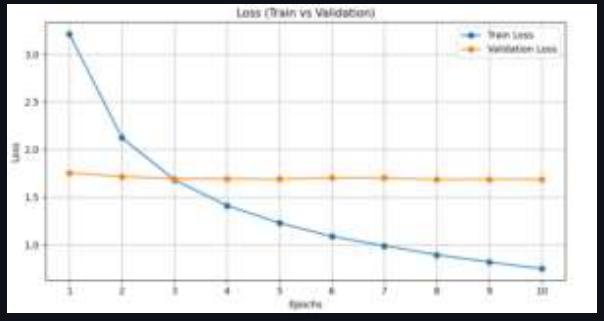
Final Update

From the last update, we have implemented SVM and CNN as our final two models. Below are our results after training them on our different datasets as well as our discussion. We have a small, medium, and large dataset, which are referred to as 1, 2, and 3 below. We only included results from sets 2 and 3 for CNN because 1 is obsolete after completing 2. For SVM, we were only able to complete dataset 2, due to time restraints on GPUS on ICE. We tried running dataset 3 on it, but after running for 14 hours on ICE it did not finish and timed out.



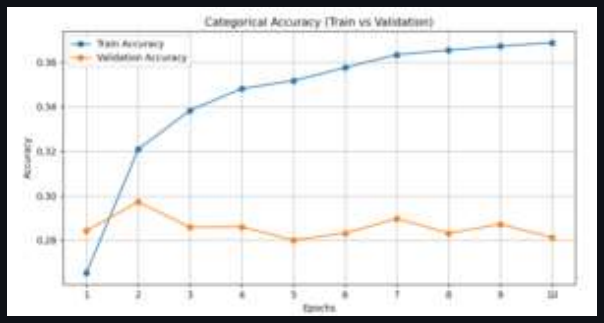
CNN Dataset 2

The categorical accuracy measures how often the predicted classification matches the true classification. The training categorical accuracy increases from about 17.6% to almost 80% across the 10 epochs. This indicates that the model is improving in predicting the correct class as the training progresses. The validation categorical accuracy is around 20-25% and stay consistent over the 10 epochs. This suggests the model may be overfitting the training data as it is working significantly well in training, yet not as well in validation on unseen data.



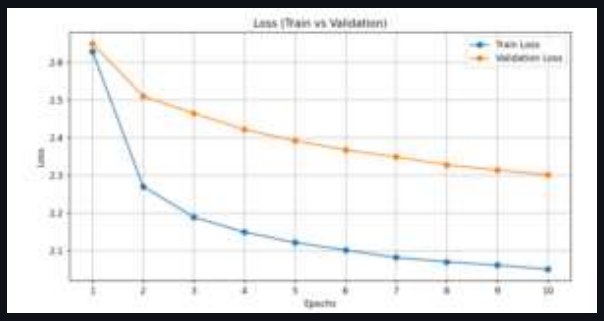
CNN Loss Dataset 2

The loss function measures how far off the model's predictions are from the actual classifications. The training loss decreases over the 10 epochs from 3.21 to 0.751, showing that the model is improving over time. Loss decreases over the epochs as the model updates its weights during training. The validation loss does not alter as much and stays consistent at around 1.75. This suggests the model is not generalizing as well to unseen data and could be due to overfitting.



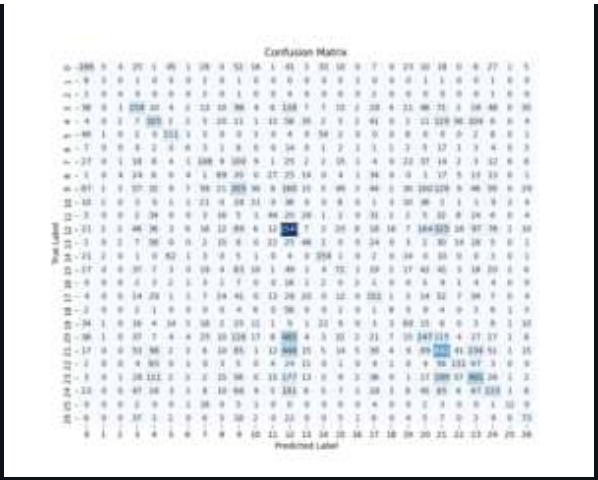
CNN Dataset 3

The training accuracy increases from 26.53% to 36.87% over the 10 epochs. This suggests that the model is gradually improving its ability to correctly classify the images. The validation accuracy measures how well the model is generalizing unseen data. In our model the validation accuracy stays consistent around 28-29% over the 10 epochs. This suggests that the model may not be generalizing the data as well as expected.



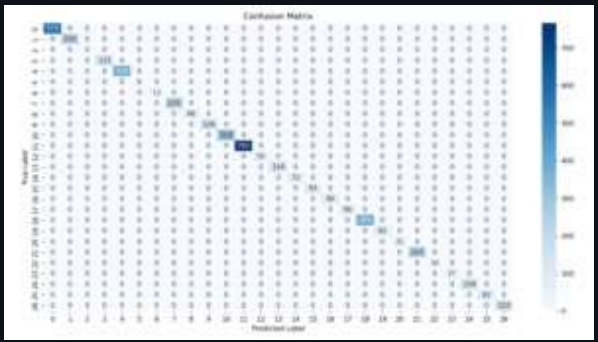
CNN Loss Dataset 3

In the training data, loss steadily decreases from 2.63 to 2.05 over the 10 epochs. This indicates the model is making progress in reducing error over the epochs. The validation loss decreases from about 2.6 to about 2.3, suggesting that error is being reduced over the epochs. This is all positive results as it shows that our CNN model is learning how to categorize the data and improving over time.



CNN Confusion Matrix

The confusion matrix of a multi-class classification problem is a square matrix where the number of rows and columns correspond to the number of classes. Along the main diagonal is where the model predicts a data points class correctly. The confusion matrix heat map generated for the CNN model reveals that most of the images were correctly classified as there is greater classifications along the main diagonal. The misclassifications show that there are some errors in this model, which is important to note when obtaining results.



SVM Confusion Maxtrix Dataset 2

Our Accuracy = 1.0, and our F1 Score = 1.0. This SVM model has perfect accuracy, as it correctly predicted 100% of the instances for the training dataset split. The F1 score is also perfect, highlighting the model performs well across all classes. From the accuracy and F1 score measurements, we see that this SVM model achieves perfect performance, classifying each image correctly. The confusion matrix was the same for both the testing and training datasets, revealing the accuracy and f1 score are consistent. However, further investigation revealed that the testing split wasn't properly generated, and it was actually the same as the training split. The perfect accuracy and F1 score could indicate overfitting as there is no error. For next steps, we could debug the issue with the training/testing split and look at the testing results. We could also try different kernels and C values, and for the rbf kernel, different values for gamma. Gamma is essentially the curvature in our decision boundary, and C is the amount of error we allow the SVM to have. Different kernels can allow our SVM to create different shapes in the original x-space of the dataset. For example, a linear kernel will always create some kind of hyperplane.

Midterm Update

The results of the GMM model was determined by looking at the visualizations, log likelihood, akaike information criterion, Bayesian information criterion, silhouette score, adjusted rand index, and a confusion matrix. The vizualizations plot the data points into their relative clusters and provide insight into the data points predicted clusters. The GMM model performed well when clusters were easily identifiable, yet struggled when there were many similarities and overlap between components. From the GMM visualizations, it is evident when clusters are clearly distinguishable, but too much overlap in the clusters can cause misclassification. Log likelihood looks at the probability of the observed data using the estimated parameters. Maximizing the log likelihood is essential to reduce error. A high log likelihood value suggests that the model fits the data well. The Akaike Information Criterion (AIC) aims to reduce overfitting by penalizing models with too many parameters. A low AIC value indicates a strong fit for the model and will reveal the optimal number of components the model should use. The Bayesian Information Criterion (BIC) is similar to AIC as it also reduces the risk of overfitting by confirming the optimal number of components for the model. BIC and AIC work together to confirm the optimal number of components to use for the model to perform best. The Silhouette Score looks at each data point and compares how similar it is to it's own cluster and to the other clusters. A higher silhouette coefficient reveals that the clusters are well defined and have a strong separation. The Adjusted Rand Index (ARI) compares predicted clusters to the true labels. An ARI close to 1 means there is perfect alignment between the predicted and actual values, where an ARI of 0 means there is no overlap between the predicted and actual values. The Confusion Matrix analyzes the values of the true positives, true negatives, false positives and false negatives. Having this information is important for understanding where the model is falling short and can help lead to adjusting misclassifications. For next steps, it would be helpful to utilize the AIC and BIC metrics to determine the optimal number of components to use for the selected data set. Ensuring the number of components will aid in reducing overfitting or underfitting and lead to a more effective GMM model.

Proposal

Metrics

Some metrics we can use include accuracy, precision, recall, and the F1-score. We'll try out different metrics with our model and choose the one that gives us the most relevant information. Accuracy is the number of correct predictions divided by the total dataset size.Precision measures how many "true" predictions were correct. Recall measures how many of the "true" samples were correctly guessed. In a multiclass scenario like ours these metrics can be recorded for all classes, or we can average them in various ways. F1-Score combines precision and recall. This can be helpful if we don't want to keep track of

two different metrics, but as a tradeoff, it hides which way our model is biased since the two metrics are averaged together. All of these metrics are in the scikit.metrics library, as well as the different ways to implement them with averaging and multiclass situatio

Goals and Results

We expect our model to have a final accuracy that is better than a human who knows only the very basics about art and its history. This is around the emerging/competent levels of AI. Another one of our goals is to save training time. Ideally, we reach our target accuracy in a timely manner. This can be achieved by optimizing our algorithms and only rerunning a training job after a change in either the dataset or the model.

Members:

- Tanush Prathi
- Elijah Tarr
- Daniel Bataillon
- Lily Mauriello
- Daniel Huang

Name	Final Contributions
Tanush Prathi	Made all CNN model and trainer code. Saved results of CNN model to JSON files and created a few visualizations for CNN model
Daniel Bataillion	Wrote the final report, consisting of the preprocessing, method, and discussion final update. Discussed the reasons behind choosing models.
Daniel Huang	Wrote all the SVM code. Updated readme with new files and helped with scheduling using slurm on ICE
Lily Mauriello	Created visualizations for CNN model and SVM model. Analyzed results and wrote discussion for CNN and SVM models.
Eli Tarr	Created the slideshow and presentation. Helped with debugging CNN and streamlit page

Name	Midterm Contributions
Tanush Prathi	Converted streamlit page from Propsal page to Midterm skeleton, Completed PCA, functions to download dataset and split into training, validation, and testing, started on CNN model, PCA visualizations
Daniel Bataillion	Discussing our choices for preprocessing and model methods, as well as explaining our choices for each
Daniel Huang	Preprocessing functions, helped with connecting to ICE, readme directory and file explanations

Name	Midterm Contributions
Lily Mauriello	Analyzing and explaining the results and discussion of our models
Eli Tarr	Coded gaussian mixture model and integrated with data processing and pca code. Trained model and made metrics.

Name	Proposal Contributions
Tanush Prathi	brainstormed ideas for project, made preprocessing and ML algo sections, made meeting with TA, found 1 one of the datasets
Daniel Bataillion	Worked on the problem definition, taking a look at different papers and how we can approach our problem from a different angle.
Daniel Huang	Created streamlit page, researched metrics and wrote expected goals/results section, found one of the datasets
Lily Mauriello	I researched the literature review articles and wrote the introduction. I ideated for the dataset and collaborated with the team in meetings.
Eli Tarr	Made slideshow, recorded proposal video, and helped with ideas for ML algorithms/preprocessing/technologies

Gantt chart

References

- [1] J. Dan, “Design of automatic style classification system of visual art works based on image processing,” Applied Mathematics and Nonlinear Sciences, vol. 9, no. 1, p. 20241590, Jan. 2024, doi: 10.2478/amns-2024-1590
- [2] A. Lecoutre, B. Negrevergne, and F. Yger, “Recognizing Art Style Automatically in Painting with Deep Learning,” in Proceedings of the Ninth Asian Conference on Machine Learning, M.-L. Zhang and Y.-K. Noh, Eds., Yonsei University, Seoul, Republic of Korea: PMLR, Nov. 2017. [Online]. Available: https://proceedings.mlr.press/v77/lecoutre17a.html
- [3] B. L. Menai and M. C. Babahenini, “The Effect Of Optimizers On CNN Architectures For Art Style Classification,” International Journal of Computing and Digital Systems, vol. 13, no. 1, pp. 353–360, Jan. 2023, doi: 10.12785/ijcds/130128.