CS460G Machine Learning Final Report - AI Art Critic

Dylan Dawson and Ryan Whitt

December 2023

Abstract This report gives an overview of the methods used to develop a Machine Learning model that uses various models to 'critique' an image. Users are able to submit an image and receive a text response based upon the categories into which the models classified the image. The results across most of the models were better than expected. However, there were some performance degradation issues when loading those models after saving them. Possible remedies for these issues are discussed in the conclusion.

1 Introduction

The purpose of the machine learning model created in this project is to have the capability to generate a textual critical analysis of artwork and other images supplied by an end-user. The analysis will not necessarily be based upon the contents of the images but instead upon several elements commonly used to evaluate artwork and photographs. These elements include the following: clarity, level of detail, color, perspective, and style. The final output of the model will be a paragraph explaining to the user the results of the analysis of the artwork.

The primary interesting characteristic of the model is that, as of the creation of this proposal, there are not any easily accessible and freely available models that are designed to accomplish the same task. There are a wide variety of models that can analyze images, and even critique artwork, but they focus more upon the content, rather than the elements of those images.

One challenge in this process was the collection of adequate training data. For the model to be successful, there must be a large quantity of labeled images falling into a wide range of categories that can be used for training and evaluating the model. While there are expansive image-based datasets available for download, many of these are not properly labeled such that they can be used in the model without requiring adjustments to the data. To remedy this issue, most of the data was gathered and labeled manually.

2 Related Work

As mentioned above, models with a similar purpose to ours are common, but we are unaware of any models that explicitly have the same goal, so there are no prior established results against which to compare our model on this specific task. Fortunately, we were still able to build off of previous work by transfer learning from a DenseNet121 model. Although originally designed for classification on ImageNet, we were able to fine tune the model to successfully accomplish our task as well. We did so by freezing the lower levels of the model and replacing the last layer without our own layers for classification.

3 Approach/Method

The first step in the project was to gather and process the data. This required a total of three short programs to be written. The first was able to pull search results from Google, show them to the user, and allow the user to select whether or not the image should be saved. This program was used to quickly gather much of the data. Second, another program was written to download files linked to in the ArtBench dataset. These images were used to train a model to detect the style of an image. Finally, a third program was written to clean and normalize the data. This program ensured all images maintained their same aspect ratio while also being resized to fit on a 224 x 224 white background. This was done to ensure compatibility with DenseNet121 and ensure that the model would learn accurately without size playing a factor in the training process.

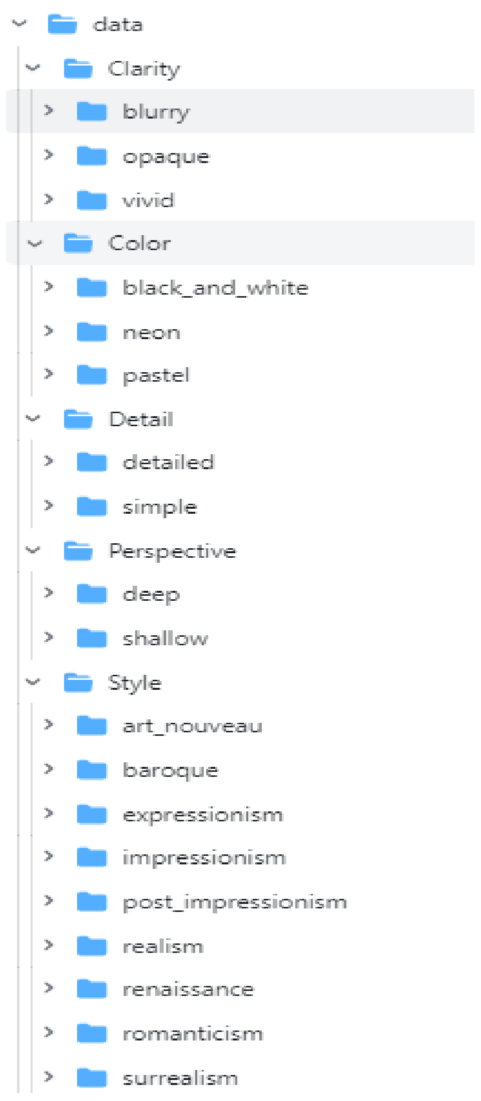
After the data had been gathered, the next step was to generate models to be able to accurately classify images into categories based on the elements of art described in the introduction. To do this, a program was created that would create a model layered on top of DenseNet121 with imagenet weights, train the model, and tune the hyperparameters. The model can have the path to the data on which it is trained specified by the variable "className". The same program was then reused to tune a model for each of the different categories for which data was gathered.

When developing the models, it became apparent that a user interface was necessary for users to be able to interact with the functionality of the models easily. The decision was made to use Tkinter for this interface, an interface to the Tcl and Tk UI that is packaged with many Python releases. The art critic interface was configured to allow users to select an input image from their file system and have it displayed in a window while it is analyzed by the model. Finally, a text output is shown to the user based on the results of the models. The user can then continue to select more images to have analyzed by the models.

*All images used during both training and classification must be ".PNG" files. In addition, version 2.14.0 of the Keras and Tensorflow libraries should be used to ensure compatibility.

4 Experimental Setup

The primary dataset used to train and test the model was a collection of images split into five different main categories: clarity, detail, perspective, style, and color. These categories are further split into descriptors, which are outlined in the following file structure diagram:



```
data
  Clarity
    blurry
    opaque
    vivid
  Color
    black_and_white
    neon
    pastel
  Detail
    detailed
    simple
  Perspective
    deep
    shallow
  Style
    art_nouveau
    baroque
    expressionism
    impressionism
    post_impressionism
    realism
    renaissance
    romanticism
    surrealism
```

Each descriptor folder contains a collection of about one hundred relevant images, and each main folder was used to train its own separate model.

The models were tuned using the Hyperopt library with accuracy as the loss function. 5-fold cross validation was used during hyperparameter tuning to allow for better comparisons amongst models during hyperparameter tuning rounds. The loss was the average accuracy across all five folds. The following are the best hyperparameters for each model after tuning.

	Detail	Clarity	Perspective	Style	Color
Batch Size	64	64	128	64	32
Epochs	15	10	15	10	10
Dropout	0.4254	0.4254	0.4013	0.6336	0.1898

Learning Rate	2.629069e-06	1.522985e-05	1.085440e-06	1.251159e-05	2.990878e-06

5 Results

The results of testing the models were as follows:

	Detail	Clarity	Perspective	Style	Color
Accuracy	0.9275	0.7830	0.9179	0.4070	0.9203

6 Discussion

Seeing as each model had a unique training set and most models differed on their number of classes, each model must be discussed individually. The perspective, color, and detail models all had accuracies around 92%. These results exceeded our initial expectations. However, the clarity and style models did slightly underperform. Although the style model only had an accuracy around 40%, it was the most difficult category both because of the subjectivity of defining the "style" of an art piece and the number of classes, 9, in which an image could be classified. The clarity model achieved results around 78%, which we believed could be improved upon significantly based on the performance of the other models with 2 or 3 classes.

7 Conclusion

Overall, we were successful in developing a model that would classify an image based on the given characteristics of detail, clarity, perspective, style, and color. However, the project could be improved in a few ways.

Firstly, although the results on the test set during training were good, the color model in particular seemed to struggle a little during by-hand testing when using the AI_Art_Critic.py program itself. There are two likely explanations for this issue. The first is that there was a mistake made in the process used to load the model. This explanation is likely because the models had to be loaded layer by layer as a work-around to a strange issue preventing Keras from loading the saved model normally. Another explanation of the issue could be that many of the files used during training were originally JPEGs that were converted to PNG files. It is possible that one class was made up of more converted JPEGs than the others and the model learned to utilize that in its classification, introducing inaccuracies when the original file type of a new image does not align with the way in which the model learned to utilize that information.

Secondly, the models seem to struggle sometimes when classifying images that are dissimilar to those on which they were trained. The primary reason for this issue is likely that the training sets for the majority of classes were small. This was necessary because the data had to be collected by hand and trained on Google Colab. If we had access to a large corpus of labeled data and a more powerful GPU, we could likely produce a more generalized model.

Thirdly, in future iterations, the interfaces and process of collecting images could be improved. Tkinter is a fine interface to begin with, but there are certainly better options with greater customization options and potential for providing a more robust user experience. The collection of images was initially done through a Tkinter interface as well. In future attempts, the collection process could definitely be improved upon and the interfaces could be more powerful, but for our purposes, our implementations worked well enough.