Project Report: Grumpy Numpy Pandas

Image Classification - Broadside Ballad Archive

Group Members: Hannan Waliullah, Niranjana Jayakrishnan, Mirthala Lopez, Grace Willhoite, and Neha Deshmukh

Date: December 17, 2020

Abstract

The English Broadside Ballad Archive has a library of approximately 30,000 images of early woodblock prints from the 16th and 17th centuries. Through these ballads, researchers can uncover years worth of societal history, which may provide grounds for the study of 16th and 17th century civilizations, cultures, and lifestyles. To provide access to researchers around the world, these images need to be categorized, which can be done through classification. Since manual analysis is tedious, we created softwares to identify efficient methods of image classification. Our classifiers are used to predict genres of images based on its features. Through this report, we transcribe our process of data storage, development of two classifiers, results and accuracy, and obstacles we encountered.

Introduction

Project Data Overview

The English Broadside Ballad Archive (EBBA) has amassed a library of approximately 30,000 images of early woodblock prints from the 16th and 17th centuries. These broadside ballads provided news updates, entertainment, stories, jokes, etc. applicable to the common people of each time period. Each of these ballads contained images, used to reach the majority of the population which was illiterate. Each image is unique, with distinct blocks for printing and little technical similarities.

Motivations

Through these ballads, researchers can uncover years worth of history about 16th and 17th-century civilizations and cultures, as well as how their functioning and decisions affected our present-day societies. Additionally, classifying and categorizing these ballad images ensures their accessibility to researchers around the world.

Our Contributions

Manual classification of these ballads would be tedious and time-consuming. To make this process efficient, researchers developed systems to analyze the ballad images and categorize them based on common, generic characteristics such as the content of the image, subject, type of image, etc.

Our goal is to aid Arthur Koehl, Tyler Shoemaker, and Carl Stammer from UC Davis DataLab in the cataloging of historical works by creating efficient methods of image classification and analysis, making ballads accessible through search engines. Through our study and exploration of the applications of machine learning, we implemented Python Libraries like OpenCV and TensorFlow for image classification. Through this process, we learned about image classification, its implications and difficulties, and contributed towards the analysis of these historical images.

Research goals and questions

Research Questions:

1. How accurate of an image classification system can be implemented to categorize using descriptive tags and/or feature extraction?
2. Can we use the features of tagged images in a certain genre, to accurately classify other untagged images by genre through the identification of similar features?
3. Which method is more accurate for image classification between visual Bag of Words and Convolutional Neural Networks (CNN)?

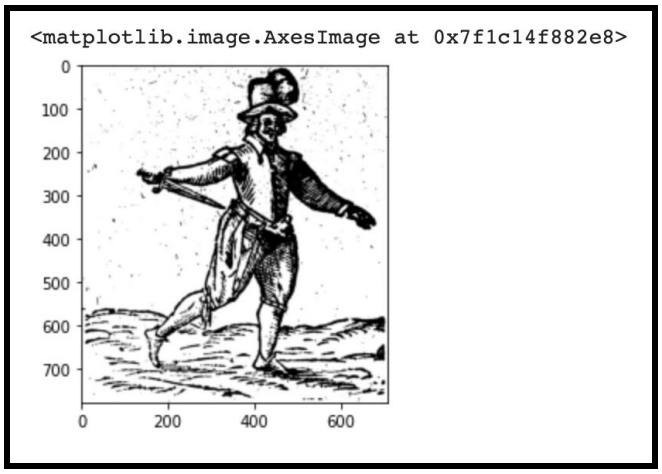
Research Goals

Our research goals include implementing and running our image classification algorithms on a training dataset of manually tagged images and then on the full dataset to predict an image's genre based on its extracted features. We analyzed our two classification algorithms and aimed to maximize their accuracy using statistics and visualization tools.

Data

Some of the given 17,000 images are tagged, most of them containing multiple tags per image. The two types of labelling tags are: **Descriptive tags,** which describe the content of an image with specific characteristics like animals, objects, people, etc. and **Genre tags**, which label genres and type of woodcut impressions; if an image is a portrait, landscape, etc. The remaining images are

untagged. Woodblock impressions from the same woodcut can be different due to details like unlinked parts, blotchy ink, holes, etc. and need to be ignored when features are extracted.



Data Received and Analysis:

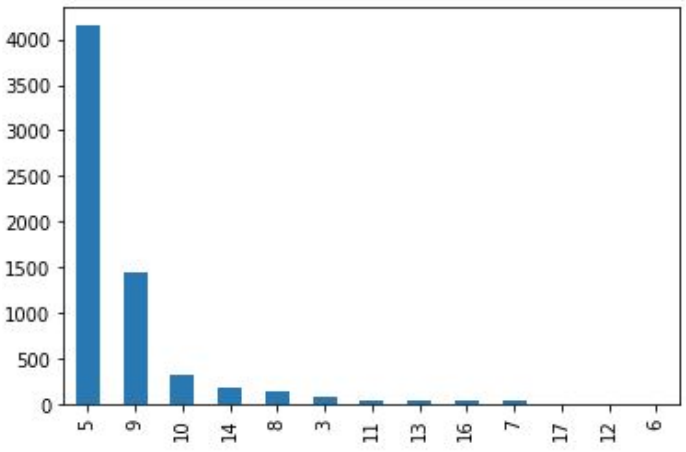
We first found how many images we were working with and how many of these images were genre-tagged, descriptive-tagged, and untagged using the CSV files previously mentioned. Below are our findings:

Total number of images in the dataset	17838
Number of images with id numbers	16103
Total number of images with tags	6296
Total number of images without tags	11542
Number of images with genre tags	6171
Number of images with descriptive tags	6295

One of the issues we ran into was that a lot of image IDs were missing from our combined dataframe. Only around 40% were actually tagged. We found the total number of tagged images by combining our files and counting the number of unique image IDs. Then, we used the CSV files to find out how many unique tags there were. Below are our findings:

Number of descriptive tags	419
Number of genre tags	13

We also calculated the number of images that appeared for each genre tag. Since there are only 13 genre terms, we were able to visualize the data in a histogram shown below.

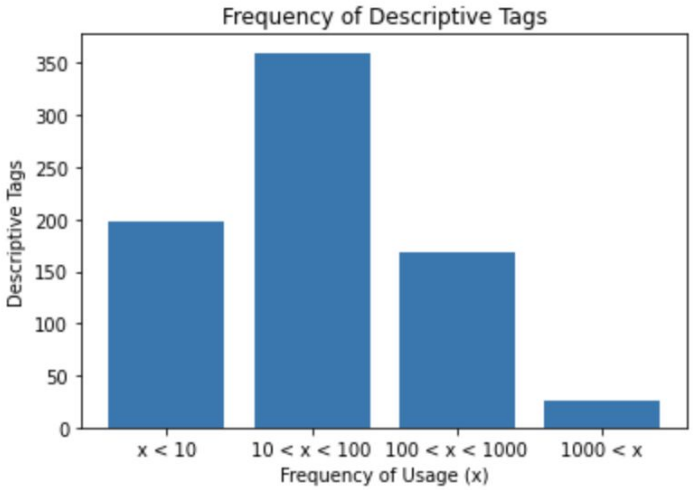


Histogram of the number of images per genre tag ID

From this histogram, we realized that we might need to add weights to our classifier because the data is skewed. There are over 4000 images in the portrait genre (5) but very few images for the other genres like landscape (17) , architecture (12) , animal art (6), etc.

Number of Images using a Descriptive Tag:

Although it may be effective to include descriptive tags to further classify images, we realized that the number of images using specific descriptive tags was inconsistent and we did not have enough data per tag to effectively incorporate Descriptive Tags as one of our filtering elements into our classifiers. For example, some tags were only used a few times by the given images whereas others were used more than a thousand times. We made a visual aid to distinguish the number of tags used per number of images.



Specifically, 200 tags were used less than 10 times throughout our dataset. This inconsistency caused accuracy issues within our classifier, so we ultimately refrained from including descriptive tag data.

Methods

Step-by-Step Processes:

Tagging Data Storage:

We imported the data from the five given CSV files into multiple Pandas Dataframes using the "pd.read_csv" function, then combined these into a single dataframe, so we could easily find tags associated with a given image ID. The "pd.merge" function was helpful to combine data frames without losing values. The following is the format of our dataframe, with columns for image ID, descriptive tag IDs, and genre tag IDs respectively.

```
[13] df
```

IDT_IMP_ID	IDT_DT_ID	IGT_BGT_ID
2	674,103,191,342,737,350,210,651,634,216,660,32...	5
4	172,334,737,325,670,634	5
6	644,634,325	5
8	669,634,325	5
9	264,320,670,256,737,634	5
...	...	...

Image Data Storage & Localization of Code:

We were given a 16 GB zip file of photos to work with. The file was too large to unzip directly on Google Colab as it required a large amount of RAM, so we resorted to downloading the file and unzipping it locally. For testing and collaboration purposes, we uploaded a random subset of these images to a shared Google Drive to enable multiple team members to test feature extraction methods, classification algorithms and sorting scripts. Once we implemented methods that worked, we ran them locally on the entire dataset.

Feature extraction through Google Colab was difficult with file sizes exceeding a computable threshold. This process helped us realize that data storage is a big part of the research process.

Image Sorting Scripts:

We coded multiple Python scripts to sort the given images into different folders based on their category. The first script separates the tagged and untagged images. It does this by iterating over all the image files in the directory, converting the filenames to image IDs using a dictionary, and then cross-checking these IDs against a list of tagged IDs. This script is necessary to determine the set of images to be used for training our classifier as we can't use untagged images for training. We also coded a script that further extracts genre-tagged images from the tagged images and sorts them into folders.

Classifier 1: Visual Bag Of Words using Image Features

Feature Extraction - OpenCV and SURF:

The library that we will be using is OpenCV in Python. OpenCV has a keypoint extractor named **SURF**, which helps identify and extract features of images. An **image feature** is a numerical characteristic describing the content of an image based on pixel density contrast, usually indicating specific properties. In SURF, image features have 64 dimensions. Assigning these extracted features to an image's associated tags can identify the image genre.

While SURF is an extractor used with 3-D images, the Broadside Ballad archive is a collection of 2-D drawings. Using SURF, we will have to account for different image artifacts than 3-D images, skewed contrast, and the degradation of images due to age.

Classification Algorithm - OpenCV:

For our classification algorithm, we are in the process of creating a visual 'bag-of-words' vocabulary using the SURF features, k-means classification, and an SVC.

We used a **bag-of-words (BoW)** to represent the features of images. Our visual words are the SURF features. Since the number of image features found on each image varies, it's necessary for us to create a representation that acknowledges the inconsistent properties of each image.

Steps:

1. **Feature extraction**

   Using our script for extracting image features, we created a list of features F from a training set with known genre tags. $F = \{ f_1, f_2, f_3, ... f_M\}$

2. **Clustering to create a bag-of-words**

   After creating our list of features, we divide them into k clusters using the k-means algorithm. Each cluster has a centroid $C = \{c_1, c_2, ... c_k\}$, where C is a set of centroids and $c_i \in \mathbb{R}^{64}$ (surf features have 64 dimensions). These centroids, or **visual words**, somewhat represent the "main features" in the training dataset.

3. **Feature vector generation**

   Given a new image j ∉ D (where D is the training set), we will extract the features of j. Then, for each feature f ∈ j, we will find the closest centroid C, and create a feature vector of the frequencies, *I (which represents the frequency of the codewords in image j)*.

4. **SVM Classifier**

   We can use this feature vector for a SVM classification model. That way, given a new image, we can extract its features and predict its genre based on its features.

Classifier 2: Convolutional Neural Networks

Classification Algorithm - Tensorflow and Keras:

The library we used is Google's Tensorflow and its high-level API, Keras, in Python. **Keras** can only be used with **Tensorflow 2.0**, and allows for data scientists to focus on deep learning by supplying abstractions for high-velocity iterations (**epochs**) over datasets. For image classification and other forms of data processing, TensorFlow creates structures known as **dataflow graphs**, which describe the way data moves around throughout a graph**.** These graphs consist of nodes that each represent some mathematical function or operation. The connections between these nodes are called **tensors**.

Steps:

1. **Sort images into a training and test set for each genre**
2. **Regularize size of pictures to 150x150 and extract filter**
3. **Activate our function with either sigmoid for binary classification or softmax activation for categorical classification**
4. **Preprocess images for training and validation sets**
5. **Train and evaluate model**

Results

A) Classifier - OpenCV:

Feature Extraction Results:

After running the feature extraction script, we stored the information in a CSV file with image name, features, and number of features per image. We then converted this file into a dataframe, which we later combined with a dataframe containing tagging information.

One of the main considerations with the feature extractor is determining an adequate Hessian Threshold. A **Hessian Threshold** correlates with the number of features found for an image: higher the threshold, lower the number of features found.

We decided to make a robust adaptive Hessian Threshold. Our default threshold was 50,000, however if no features were extracted, then we decreased the threshold to 10,000. If there were more than 1,000 features extracted, we increased the threshold to 75,000. Thus, we would bottleneck the images so that they all had 100-1000 features. k0o0

Model Summary

Through our Visual Bag of words model, we were able to create a classifier which had ~55% accuracy. We created an 80/20 train-test split for cross validation.

A summary of our model:

K-Value for Kmeans	k = 200
Regularization Constant	C = 1
Decision Function	One Vs. Rest

Confusion matrix:

```
array([[  0,   0,   0,   0,   1,   0,   0,   0,   0,   0,   0],
       [  0, 147,   0,   0,  30,   0,   0,   0,   0,   0,   0],
       [  0,   1,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   8,   0,   6,   1,   7,   1,   1,   1,   4,   0],
       [  0,  52,   0,   1,  26,   3,   0,   0,   0,   3,   0],
       [  0,  13,   1,   9,   5,  12,   2,   0,   0,   8,   0],
       [  0,   3,   0,   0,   0,   0,   0,   0,   0,   1,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   1,   0,   1,   0,   0,   0,   0,   2,   1,   1],
       [  0,   5,   0,   0,   1,   1,   0,   0,   1,  22,   0],
       [  0,   3,   2,   0,   5,   0,   0,   0,   0,   4,   0]])
```

Accuracy: `0.5415617128463476 (54%)`

Comparison of only two genres:

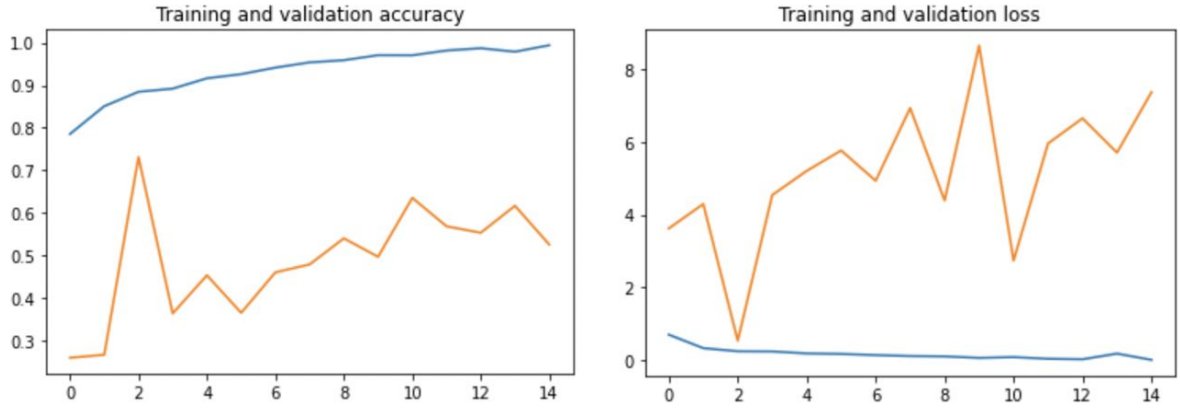
To see if the classifier would be better on genres of similar frequencies, we decided to test our classifier genres 9 & 10, which have a similar number of images.

Through this, we got an **80% accuracy** rate, which is higher than the multi-class classifier.

B) Classifier - Neural Networks:

Binary CNN classification with two genres

We initially built a binary CNN classifier that could classify an image as a portrait (genre 5) or narrative (genre 9) as each of these genres contained more than 1000 images. First we split the images in each genre into training and validation directories. Then we ran these images through a 3-layer convolutional neural network for 15 epochs. For our neural network, we used a sigmoid activation function for binary classification. Below are the graphs for evaluating the accuracy and loss. In blue we have our training data and in yellow we have our validation data.



Accuracy:

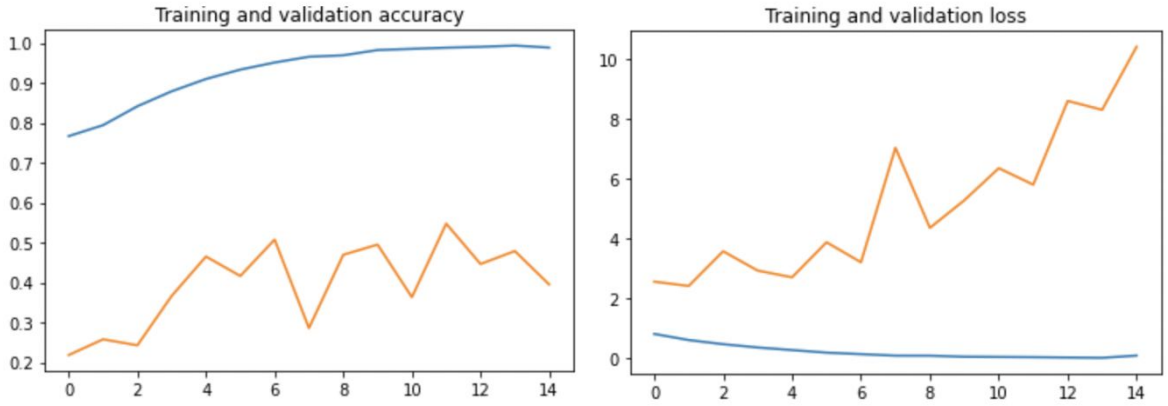
The validation accuracy oscillates with each epoch, peaking at 73% accuracy after epoch 3. It starts to even out around epoch 7 reaching an accuracy of 54%. We can see that overfitting begins to happen here as the training accuracy reaches 90% and continues to get closer to 100%.

Loss:

The validation loss oscillates more than the accuracy, peaking with a minimum of 0.54 during epoch 3. It starts to slightly even out around epoch 12 with a loss of 6.4 when the model begins overfitting.

Multi-categorical CNN classifier

Next, we built a multi-categorical classifier that could classify an image into one of the 13 different genres available. We split the images in each genre into training and test directories. Then we ran these images through our neural network which used a softmax activation function for the categorical classification instead of the sigmoid activation function. Below are the graphs for evaluating accuracy and loss:



Accuracy:

The validation accuracy oscillates with each epoch, peaking at 50% accuracy after epoch 7. It starts to even out around epoch 12 reaching an accuracy of 46%. We can see that overfitting begins to happen here as the training accuracy reaches 90% and continues to get closer to 100%.

Loss:

The validation loss oscillates more than the accuracy, peaking with a minimum of 2.42 during epoch 2. It starts to slightly even out around epoch 12 with a loss of 6.4 when the model begins overfitting.

Discussion

We implemented our classifiers to work only on genres containing a large volume of images. Since some genres did not have substantial data, it was difficult to include such genres for classification of our training and testing sets. For genres that may only have, for example, 3 images, it becomes difficult to detect such a genre in the classifier at the validation stage.

Bag of Words Interpretation:

Based on the Confusion matrix and accuracy (54%), it is clear to see that the classifier is adept at identifying the more frequent genres , based on the true positive rate. However, these genres also

have a high false positive rate because the classifier is unable to identify any of the smaller genres. With fewer genres the classifier is fairly accurate (80%).

Neural Networks Interpretation:

The accuracy for CNN fluctuates pretty significantly epoch to epoch. While the information is valuable it is also inconsistent. Looking at the graphs, there is usually a peak for the validation accuracy that then dips and eventually oscillates around an average value. Looking at the initial peaks, CNN appears sufficiently effective (up to ~73 percent with minimal loss for 2 genres, ~65 percent accuracy for top 3 genres, up to ~55 percent accuracy for all genres). After the overfitting (> 95% training set accuracy) , the accuracy falls between 40-60 percent. Based on these results, our CNN classifier is better than a 1/13 guess for the genre, but still is only correct 40-60 percent of the time. It performs better for 2 genre classification than 3 or 13 genre classification.

Comparison:

The accuracy of both classifiers were similar: both had around 50% accuracy. However, Bag of Words was less time intensive compared to the neural networks. With the Bag of Words classifier, the features only had to be extracted one time for all of the images. Then the classifier could be modified without needing to re-extract the features each time. However, CNN takes ~45 minutes to run when there are alot of genres and for every edit must be rerun for the same amount of time before determining accuracy. With less genres, Bag of Words is more accurate, and less time-intensive.

Overall, both classifiers had a low accuracy rate, which could have been due to a few factors. There were a large number of genres. Some genres had more than 1000 images, while others had less than one hundred. As a result, the false negative rate for smaller genres was inflated. A better classifier would need a consistently tagged data set.

Conclusion

Throughout our project, we discovered that the percent error is significant when using our data set. This means any benefit of having a quick process to classify image genres for research is mitigated by the amount of time taken to double check the results for those accessing the image. Regardless, automation would be preferable and far more useful than in person cataloguing. Between the two classifiers, our CNN classifier performs better for classifying all 13 genres though it takes significantly longer to run than Bag of Words. CNN also has potential for further advancement in the future and more flexibility. However, Bag of Words is more effective for smaller data sets and is more time efficient.

There are few factors that could have contributed to improving our classifiers initially. The dataset as is does not have a consistent distribution of genres. In order to train more reliably, we would need a couple hundred tagged images per category. A larger dataset with an adequate number of tagged images for each smaller genre would have allowed us to obtain better samples when creating our training and testing sets which could be obtained by manually tagging certain genres more intentionally.

For future research, designing a two layer classifier could give better results. In the first pass, the classifier would check if the image is a portrait, narrative or neither. If the image gets classified as neither portrait nor narrative, then we could classify the genre further with one more tailored for smaller data sets. This idea was thought of outside of the scope of time left for the duration of our project, but we would love to see it developed further.