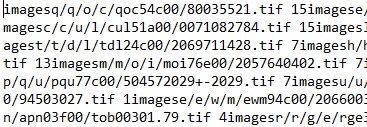
Introduction. An overview of the project and an outline of the shared work.

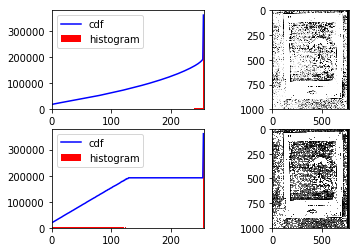
2. Description of your individual work. Provide some background information on the development of the algorithm and include necessary equations and figures.

3. Describe the portion of the work that you did on the project in detail. It can be figures, codes, explanation, pre-processing, training, etc.

**Individual Work**

My initial individual task consisted of considerable data cleaning. Images were stored in a format of each image residing in its own folder with multiple layers of parent folders with nonsensical names, all stored in the images directory. Image labels came in the form of a text file of all image file paths from the image directory and the numeric label with a space between the path and label with no space between the label and the next image path. The label text file was split on the image directory name and again on the space between path and label to create a nested list with each sub-list representing the image path and label. Next, a new image directory was created as well as a reference dictionary mapping numeric label to document type. New subdirectories were made for training, testing, and validation, with document type folders for each dataset folder. The image sub-path was then appended to the end of the new parent directory and moved to the individual document type folder via mapping the numeric label to the reference dictionary. Once sorted, the training set was randomly subsetted to create validation and testing datasets while preserving equal class distribution.

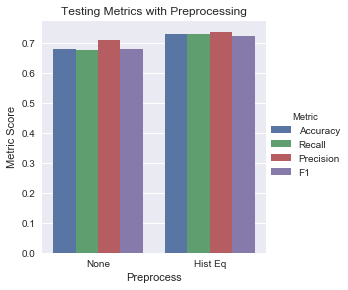
I then researched image preprocessing techniques that would be appropriate for the images and performed exploratory data analysis. I determined that histogram equalization would be ideal for the dataset. Histogram equalization is an image processing method that attempts to linearize the histogram of the cumulative sum of pixel values. This brings out areas of lower local contrast by equally distributing high intensity values. As can be seen in Figure \_\_\_\_, histogram equalization casts more pixel values towards the lower end of the grayscale spectrum (black) and increases overall images contrast. I then explored images for the distribution of dimensions and determined that all images had a height of 1000 pixels and almost all images had a width between 750 and 800 pixels, meaning that relative aspect ratio between images would be mostly preserved in resizing. Finally, I looked into how features were affected by resizing by checking images at half, quarter, eighth, and sixteenth sizes. As can be seen below, images lose distinguishable features at eighth size, therefore keeping images at *224x224* is the minimum size for preserving enough clear features for classification.



I was also the primary convolutional neural network model builder and trainer. I conducted all my model training in PyTorch. I first devised a custom wrapper around PyTorch’s Dataset class to load the data. This method first involved creating a reference csv for each training and testing datasets that contained the full image path and the numeric label. This csv was then passed through the custom data loader and each image was read using the path with OpenCV. The image and label were passed through PyTorch’s DataLoader class and iterated over to create batches for training.

4. Results. Describe the results of your experiments, using figures and tables wherever possible. Include all results (including all figures and tables) in the main body of the report, not in appendices. Provide an explanation of each figure and table that you include. Your discussions in this section will be the most important part of the report.

**Results**



**Histogram Equalization**

Models were trained over 10 epochs with the original image pixel values and with histogram equalized pixel values. The model trained on the original images tested with an accuracy of 67% and similar recall, precision and F1. The model trained on the histogram equalized image, which are shown to have greater contrast than the original images, tested with an accuracy of 72% with similar recall, precision, and F1. As stated, this is most likely a result of the increase in contrast and the highlighting of lower local contrast through equal distribution of high intensity values.

**Image Size**

This initial training parameter tested is image size. Image size variations create a tradeoff between the number of parameters that require training and the number of features that are available for classification. Models with image sizes of *224x224, 100x100, and 50x50* are tested, with image sizes larger than *300x300* exceeding the memory capabilities of the GPU.

As expected, the larger image produced the best results, with 72.8% accuracy. Smaller images drastically worsen results, but greatly improve training time. This is due to the exponentially larger number of parameters that require training will the increased size of the input, with larger images having significantly more features to classify on but more parameters to update.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Image Size | Accuracy | Recall | Precision | F1 | Train Time |
| 50 | 0.398 | 0.407 | 0.52 | 0.376 | 1254 |
| 100 | 0.578 | 0.582 | 0.631 | 0.576 | 1842 |
| 224 | 0.728 | 0.729 | 0.736 | 0.7234 | 3039 |



**Batch Size**

Models with varying batch sizes were trained and compared. Batch sizes include 1, 10, 50, and 100 images per batch.

**Learning Rate**

Models were trained with several different learning rates, including 0.01, 0.001, 0.0001, and 0.00001. A higher learning rate will learn model parameters faster but result in a jumpy output, while a smaller learning rate will learn model parameters slower but provide a smoother convergence on the minimum of the loss function.

The below figures show the outputs of the different learning rates. A learning rate of 0.001 produces the best results, with an accuracy of 72.8% with similar recall, precision, and F1 scores. As the loss chart shows, this learning rate creates the fastest convergence on the minimum loss, with 0.0001 producing the next best results. A learning rate of 0.00001 causes a slower convergence but a more compact variance while a learning rate of 0.01 causes an initial spike of high loss and has a high loss variance. The learning rate of 0.001 produces the best compromise of convergence speed and variance.



**Kernel Size**

Models were trained with several difference kernel sizes in each convolution layer.

**Pooling Type**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Pooling Type | Accuracy | Recall | Precision | F1 |
| Average | 0.304 | 0.308 | 0.469 | 0.279 |
| Max | 0.728 | 0.729 | 0.736 | 0.7234 |

Two types of pooling were considered: Max Pooling and Average Pooling. Average pooling acts similar to max pooling with the exception that the calculated value is the average of the kernel values rather than the maximum value. While max pooling is better at extract importance features, average pooling produces a smooth representation of the image. The results show that max pooling performs significantly between for these inputs. This is due to the limited number of features and majority white space of the input images. Max pooling extracts and enhances the limited number of features whereas average pooling reduces individual feature importance.

**Header and Center Cropping**

Manual examination of the documents led to the observation that most of the features of the document exist within the header or the center of the document. To test if running specific regions of the document through the network would produce equal classification metrics, two regions of the document were cropped, the header (top 1/3rd of the document) and the center (central 1/3rd vertically and central ½ horizontally) and resized to the same dimensions of the full image input (*224 x 224*).

5. Summary and conclusions. Summarize the results you obtained, explain what you have learned, and suggest improvements that could be made in the future.

6. Calculate the percentage of the code that you found or copied from the internet. For example, if you used 50 lines of code from the internet and then you modified 10 of lines and added another 15 lines of your own code, the percentage will be (50−10 / 50+15) ×100.

For this project, I wrote approximately 850 lines of codes, including:

* 80 lines for data extraction and sorting
* 100 lines for EDA
* 175 lines for preprocessing
* 40 lines for making reference csv’s
* 250 lines for loading data into PyTorch, building CNN’s, calculating metrics, and extracting kernel values
* 200 lines for plotting metrics, loss, kernels, confusion matrices, and convolution outputs

Of these lines, approximately 350 lines were influenced by internet sources. Of these, 200 lines were modified and 150 were copied directly, mostly from OpenCV documentation for preprocessing, GitHub for developing the CNN in PyTorch, and Stack Overflow for simple inquiries or error handling. This results in a percentage of 17.6% of code copied directly from the internet and 41.7% of code influenced from internet sources.

7. References