Machine Learning II
Final Project
Individual Report
Jackson Crum

Identifying Digitally-Scanned Documents with Convolutional Neural Networks

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

The purpose of this project is developed a series of convolutional neural network and test various parameter settings to classify digitally scanned documents into 16 distinct classes. The data comes the RVL-CDIP Dataset (Ryerson Vision Lab Complex Document Information Processing). This dataset consists of 400,000 greyscale images of 16 classes of documents. The images are presplit into 320,000 training images, 40,000 validation images, and 40,000 testing images. All image dimensions are 1000 pixels or less, though this will be scaled down for training efficiency. The 16 document classes include letter, form, email, handwritten, advertisement, scientific report, scientific publication, specification, file folder, news article, budget, invoice, presentation, questionnaire, resume, and memo.[1]

There is significantly less previous research conducted on document-based image recognition than natural image recognition. Research conducted by the Department of Computer Science at Brigham Young University examined CNNs for document data analysis using a variety of network depths, data augmentation, non-linear components, and input sizes. Best results were achieved with less convolutional layers and using batch normalization and dropout. Overall accuracy sharply decreases with increasing network depth. This accuracy was decay especially pronounced when less than 10% of the data and smaller images were used. Performance also has a 1-2% decrease with 50% of the training data. Larger images (optimal dimensions at 384x384) and shallower networks appear to produce the best results.[2]

Convolutional neural networks are used for this project as they take matrices as inputs and use a relatively small number of trainable parameters to learn features. As images are simply matrices of a large number of pixel values, CNN's are ideal for this task. As a group, we developed a workable dataset, explored CNN architecture and tested different deep learning frameworks, including PyTorch and Keras.

[1] "The RVL-CDIP Dataset." *Carnegie Melon University,* www.cs.cmu.edu/~aharley/rvl-cdip/.
[2] Tensmeyer, Chris, and Tony Martinez. "Analysis of Convolutional Neural Networks for Document Image Classification." *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, August 10, 2017. doi:10.1109/icdar.2017.71.

In this project, we explored using CNN's in both PyTorch and Keras to examine if the architecture will produce an effective classifier for document images.

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

Data Cleaning

My initial individual work consisted of considerable data cleaning. The data consisted of 400,000 images split into training, testing, and validation datasets. Each image was in its own folder within a deep hierarchy of folders without any particular order. Image paths and numeric class labels were stored in separate text files. After downloading the images, I devised a method to create a new parent directory with training, testing, and validation folders and individual class folders within each of those, each named with the document type. Images were identified by path and moved to the appropriate new class folder using a reference dictionary between document type and numeric label. After sorting the data, I decided to subset the data to 2,000 images per training class and 750 per testing/validation class. I then explored the data to determine the overall distribution of image dimensions, white space proportion, and the tradeoff of image size and feature clarity.

Histogram Equalization

I next researched preprocessing methods and determined that histogram equalization would likely have a positive impact on image recognition. Histogram equalization attempts to linearize the histogram of the cumulative sum of pixel values as much as possible by applying a non-linear transformation to each pixel of the input image. This brings out areas of lower local contrast by equally distributing high intensity values. The equation for the

$$s_k = T(r_k) = \sum_{j=0}^{k} p_r(r_j) \quad \text{for} \quad 0 \le k \le L-1$$

where $p_r(r_j) = \frac{n_j}{n}, \quad j = 0,\ldots\ldots,L-1 \quad and \quad n = \sum_{j=0}^{L-1} n_j$

n_j : number of pixels with gray level r_j

n : total number of pixels

S_k = new pixel value, r_k = original pixel value, k = value range, L= highest pixel value in range k

transformed pixel value is the cumulative sum of the frequency of pixel values up to that level. A transformation is then applied to this new value to make the cumulative distribution linear.[3]

Convolutional Neural Networks

I then researched and developed the architecture of our convolutional neural network models. Convolutional neural networks are the primary neural architecture used for image analysis because they work with matrix inputs and have significantly fewer parameters that require training than other network designs. Where MLP would require each image to be flattened to a 1-dimensionl input array with a length of the square of the image dimension, CNNs work with the image in its original matrix form. MLP puts weight and bias parameters on each node (pixel) and connects that node to every other node in the next layer all the way down the network. For a 100 x 100 image, this means that the MLP would have 10,000 input neurons. A network with one hidden layer of the same size as the input and 16 output nodes would require over 1.6 billion weight parameters. The CNN drastically shrinks the number of parameters by running small sets of weight values over the input values in the form of kernels.

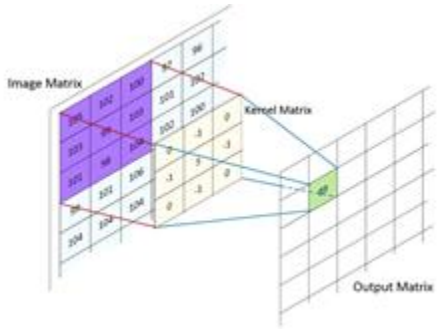
Input Layer

The input layer is a three-dimensional matrix with number of rows equal to the image height and number of columns equal to image width. For image analysis, the dimensions represent image height, width, and number of color channels, and the values represent pixel values within the range [0, 255]. Grayscale images, as used in this project, have one color channel representing grayscale value.

[3] COSTE, Arthur. "Project 1 : Histograms." *Summation of Twitches and Tetanization*, 5 Sept. 2012, www.sci.utah.edu/~acoste/uou/Image/project1/Arthur_COSTE_Project_1_report.html.

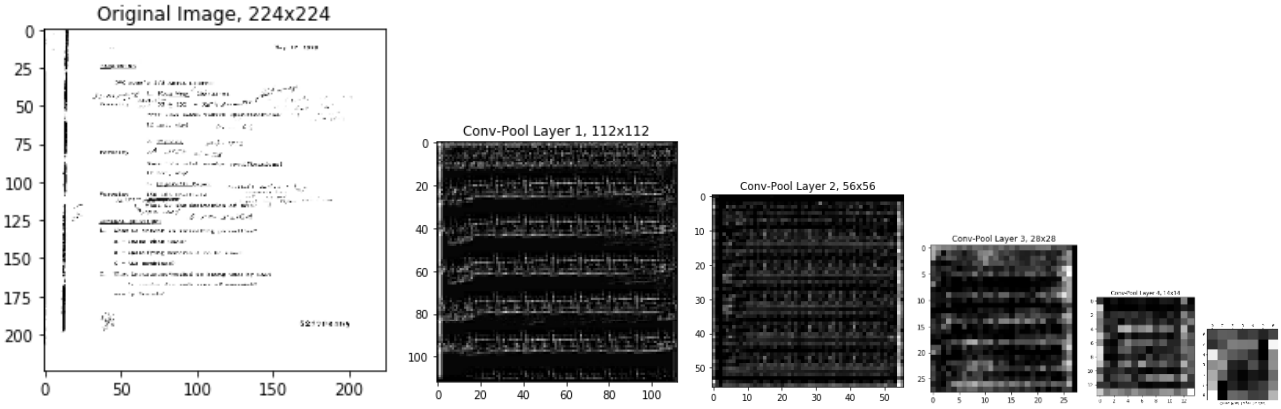
Convolutional Layers

The convolution layer consists of a set of learnable kernels, which are smaller matrices of randomly initialized weight values in a 3-dimensional matrix (number of color channels, height, width). In the feedforward process, the kernels slide (convolve) over each element of the input matrix and compute the sum of the element-wise multiplication between the kernel and the local image pixel values. This matrix multiplication results in a single value that maps to the feature map in the center position of the section of image matrix under the kernel.



Convolution Multiplication with Kernel

Each feature map is the resulting matrix from the computed values of a single kernel convolving over the whole of the input matrix. Aside from the number of kernels, convolutional layers have several adjustable parameters that control output size. Padding is a parameter that is used to maintain input size. As seen in the above figure, convolution results in a single value at the center of the kernel, which means that the edges of the input are cut off and the output is *n-1* (n representing the height and width of the kernel) dimensions smaller than the input. To prevent this, layers of zeros or other values are added around outside of the input so that the first position of the kernel is centered on the upper-left pixel value. The number of padding layers is calculated by *(n-1)/2* with *n* representing kernel size. Stride is number of pixels that kernel jumps with each step of convolution. A stride of 1 means that the kernel moves one pixel row or column at a time and results in an output image that is the same size as the input image. Increasing stride to *n* decreases the output size *n*-fold (stride of 2 results in an output ½ the input size).



Convolutional Output Down 5 Layers

Batch Normalization

Batch normalization normalizes the output of the convolution by subtracting the batch mean and dividing by the batch standard deviation. Batch normalization lessens the likelihood of the gradients decaying (moving to 0) or exploding (moving toward infinity) as well as the chances of the model becoming dependent of a small number of features with high weight values due to exploding gradients. Higher learning rates can also be used because batch normalization ensures stabilization in activation output.

ReLU Activation

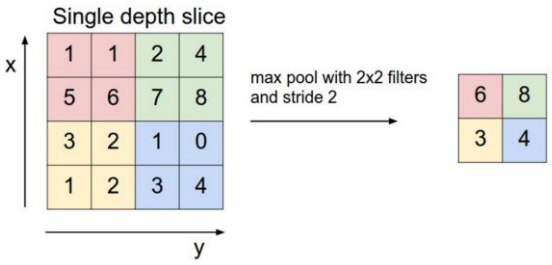
Following the batch normalization, a ReLU (Rectified Linear Units) activation layer is applied. The ReLU layer applies the function *f(x) = max(0, x)* to all of the values in the input, where *x* represents the input value. Essentially, the function sets all negative inputs to 0 and sets the output as the input for positive inputs. This function is used to calculate gradients for parameter updating and. ReLU results in a network that is computationally efficient and fast training due to the simplicity of the function derivation and the avoidance of vanishing gradient (gradient values decreasing exponentially) by maintaining a constant derivative.

$$RELU(x) = \begin{cases} 0 \; if \; x < 0 \\ x \; if \; x >= 0 \end{cases}$$

ReLU equation

Max Pooling

A pooling layer is applied after the ReLU activation function to shrink the input to reduce the number of trainable parameters. Maximum pooling applies a filter of size *n x n* and stride *n* and calculates the maximum of that filter as the output. The output of this layer is n^2-times smaller than the output. For example, as seen in Figure, a maximum pooling of 2x2 reduces the output a quarter the size of the input and the number of parameters by 75%.[4]



Max Pooling Calculation

Flattening Layer

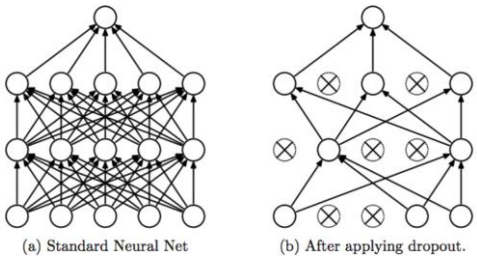
For the purposes of classification, the final layer of the CNN is a 1-dimensional array of the class probabilities. This requires that the output of the convolutional layers also be converted to a 1-dimensional array. Following the final pooling layer, the output is flattened into a 1d array of a length equal to the number of feature maps multiplied by the output size of the final layer. For example, a final layer with a 6 x 6 output and 32 feature maps results in a 1 x 1152 array.

Fully Connected Layers

The fully connected layers act as a multilayer perceptron in which every neuron in the input layer is connected with weights and bias to every neuron in the output layer. These layers serve as the classification layers of the network by extracting important features related to each class. The final fully connected layer in the model is a *1 x n* array where *n* is the number of classes.

Dropout

Dropout is a technique used to prevent overfitting and extrapolation. During each feedforward pass in the training stage, all nodes in a certain layer are nodes are either dropped with probability *1-p* or kept with probability *p*. Fully connected layer consist of the majority of parameters in the network



Dropout Example with Deactivated Neurons

[4] Britz, Denny. "Understanding Convolutional Neural Networks for NLP." *WildML*, 10 Jan. 2016, www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/.

and this cause neurons to develop co-dependency with each other during training which leads to the weakening of individual neuron power and leads to over-fitting of training data. Randomly turning off neurons allows for the neurons of true importance to be determined and weighted appropriately.[5]

Softmax Activation

The softmax function normalizes the final output vector to values between 0 and 1 and divides each output value by the sum of the vector so that the sum of the vector equals 1. The result is a categorical probability distribution for the input class, with the index of the maximum value being the label of the most probable class.

Loss

Loss is a measure the quality of the parameters based how well the network classified the input, the difference between the output and the target labels. The entire purpose of training is to find parameters that minimize loss in the model. As this is a classification problem, cross entropy is utilized to find the minimum of the loss function. Cross entropy, or log loss, compares the probability of a correct prediction to the actual prediction and punishes both errors, meaning highly confident and wrong answers are scored worse than less confident and wrong answers. Cross entropy is calculated:

$$H(p,q) = -\sum_{x} p(x)\ \log q(x)$$

Optimization

The purpose of optimization is to move towards weight and bias parameters that minimizes the loss function. Optimization calculates gradient descent and discovers the direction of steepest descent towards the minimum of the loss function along which to update the weight vector. The Adam optimizer was used for all networks.[6]

[5] Budhiraja, Amar. "Learning Less to Learn Better - Dropout in (Deep) Machine Learning." *Medium.com*, Medium, 15 Dec. 2016, medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5.

[6] *CS231n Convolutional Neural Networks for Visual Recognition*, cs231n.github.io/convolutional-networks/.

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.

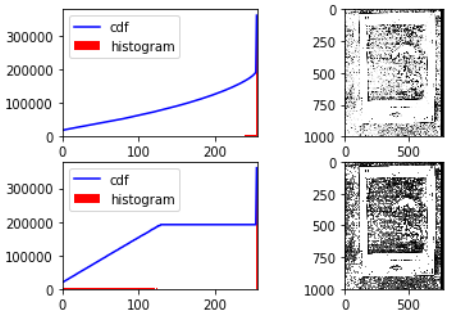
Individual Work

My initial individual task consisted of considerable data cleaning. Images were stored in a format of each image residing

```
imagesq/q/o/c/qoc54c00/80035521.tif 15in
magesc/c/u/l/cul51a00/0071082784.tif 15i
agest/t/d/l/tdl24c00/2069711428.tif 7ima
tif 13imagesm/m/o/i/moi76e00/2057640402.
```

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, testing, and validation datasets set was randomly subsetted to create workable, class uniform datasets.

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.[7]



Histogram Equalization with CDF distribution

[7] *Cascade Classification - OpenCV 2.4.13.7 Documentation*, docs.opencv.org/3.1.0/d5/daf/tutorial_py_histogram_equalization.html.

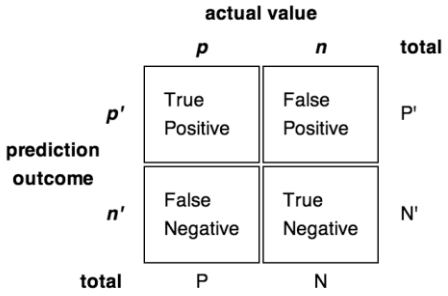
As can be seen in the above figure, histogram equalization casts more pixel values towards the lower end of the grayscale spectrum (black) and increases overall images contrast. I then explored images for the distribution of dimensions and determined that all images had a height of 1000 pixels and almost 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.[8]

After loading the image data, I built the CNN model with varying parameters. Each layer sequence consisted of a convolution layer, batch normalization, ReLU activation, and a max pooling layer. The convolution layer required the specification of input channels (first layer is 1 for grayscale input, rest are number of feature maps of previous layer), output channels (number of feature maps), kernel size, padding size (*((kernel_size-1) / 2* to maintain image size through convolution), and stride. After the convolutional layers, a fully connected layer of input size *image size * number of feature maps* (*7x7x32 = 1568* for most implementations of the model) and output size *selected number of neurons* was added. Dropout was implemented after this layer, followed by a fully connected layer of output size of 16 (number of classes) and a softmax activation for class probability. Models were tested with varying image sizes, preprocessing techniques, batch sizes, learning rates, and kernel sizes.[9]

[8] "PyTorch Documentation¶." *PyTorch*, pytorch.org/docs/stable/index.html.
[9] amir-jafari. "Amir-Jafari/Deep-Learning." *GitHub*, 18 Aug. 2018, github.com/amir-jafari/Deep-Learning/tree/master/Pytorch_/6-Conv_Mnist.

I then calculated model metrics and plotted the various metrics, losses, confusion matrices, kernel values, and feature maps at each convolution layer. Model performance was tested on several criteria. Classification metrics used include accuracy, recall, precision, and F1 are used to analysis performance. Classification can be defined by four possible results: true positive (predicted as positive class, actual label is positive class), false positive (predicted positive class, actually negative class), false negative (predicted negative class, actually positive class) and true negative (predicted negative class, actually negative class). The classification metrics calculate the following:



- Accuracy represents how often the model correctly predicts class and is calculated: *(TP + FN) / (TP + FP + TN + FN)*
- Recall represents what percentage of the inputs of a class are identified as that class and is calculated: *TP / (TP + FN)*
- Precision represents what percentage of the inputs predicted as a class are actually of that class and is calculated: *TP / (TP + FP)*
- F1 is the harmonic mean of recall and precision and is calculated: *2 * ((Precision * Recall) / (Precision + Recall))*

The metrics are calculated as an average of individual class metrics using sci-kit learn.[10] Loss will also be monitored through the input iterations and convergence on the minimum of the loss function will be analyzed for model comparison.

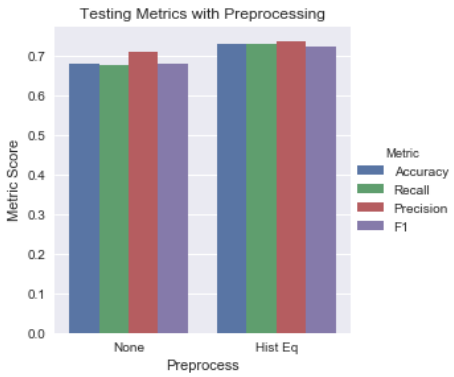
These results can be seen in the following section. Finally, I wrote the almost all of the report, put together most of the PowerPoint, and organized the entire GitHub repository.

[10] "API Reference." *1.4. Support Vector Machines - Scikit-Learn 0.19.2 Documentation*, scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics.

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

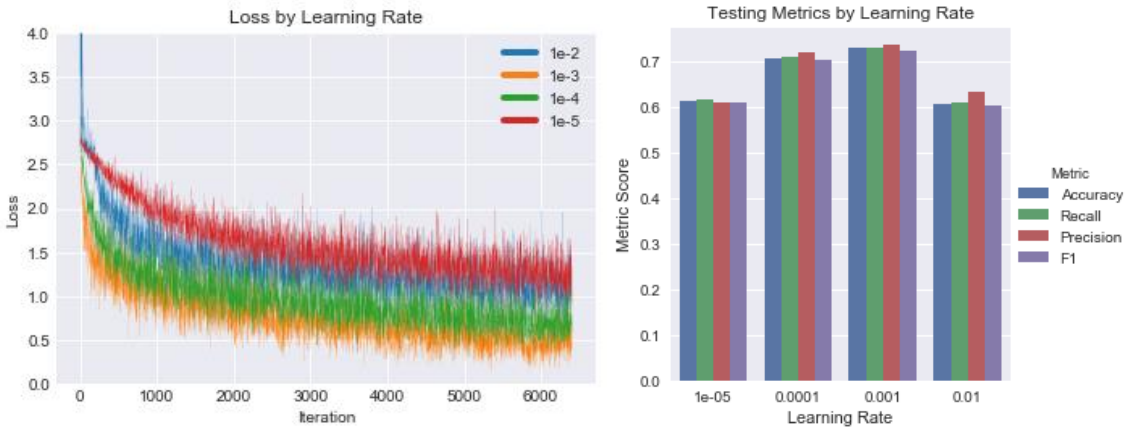
Batch Size

Models with varying batch sizes were trained and compared. Batch sizes include 1, 10, 50, and 100 images per batch. As can be seen in the metrics plot, varying the batch size produced unreliable and nonsensical results in PyTorch. Metrics should be best with the smallest batch size as this model would have the most parameter updating. Models with larger batch sizes should have worse metrics but better training time as parameter updating in completed through average batch loss and weights are updated fewer times. Batch size models will need to be rebuilt and examined again.

Learning Rate

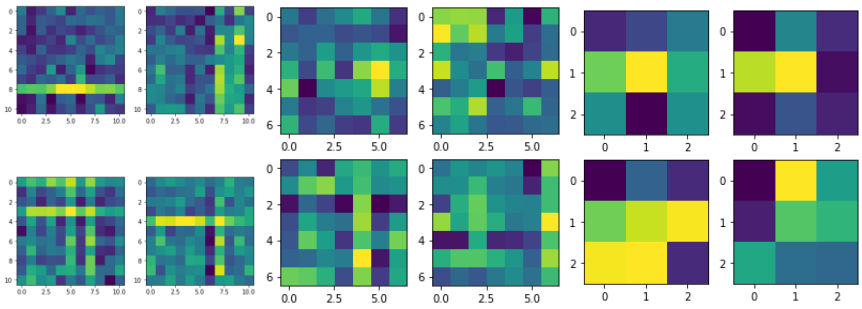
Models were trained with several different learning rates, including 0.01, 0.001, 0.0001, and 0.00001. A higher learning rate will learn model parameters faster but result in a jumpy output, while a smaller learning rate will learn model parameters slower but provide a smoother convergence on the minimum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

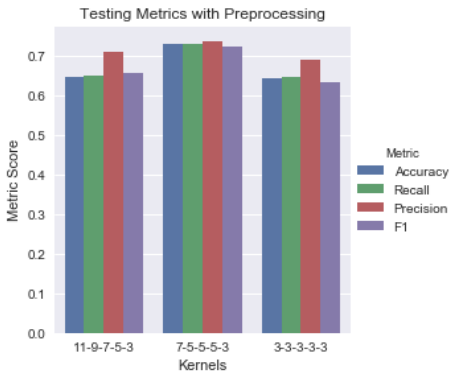
Models were trained with several difference kernel sizes in each convolution layer. Convolutional models were built with 5 convolution blocks consisting of kernel sizes 11-9-7-5-5, 7-5-5-3-3, and 3-3-3-3-3. The different sized kernels can be seen below.



11 x 11 Kernels *7 x 7 Kernels* *3 x 3 Kernels*

The larger kernels detect larger features, as can be seen by the length of horizontal lines of high kernel weights, whereas medium detect simpler linear and random features. The small kernels have minimal complex feature detection ability as can be seen by the randomness of the kernel values.

The 7-5-5-3-3 model (medium sized kernels) performed the best, with all classification metrics above 72% while the 11-9-7-5-3 model classified with a precision of 71% but an accuracy, recall, and precision all around 65%. The 3-3-3-3-3 model performed the worst, with precision at 68% and other metrics around 62%. This is most likely due to the nature of the features of the images and the inputted image size. The images generally have a combination of large features such as borders and small features such as text. The medium sized kernels provide a good tradeoff between the detection of features of various sizes. The large selected image size of *224x224* also benefits from the used of slightly bigger kernels to detect small features.



Pooling Type

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.

POOLING TYPE	ACCURACY	RECALL	PRECISION	F1
AVERAGE	0.304	0.308	0.469	0.279
MAX	0.728	0.729	0.736	0.7234

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

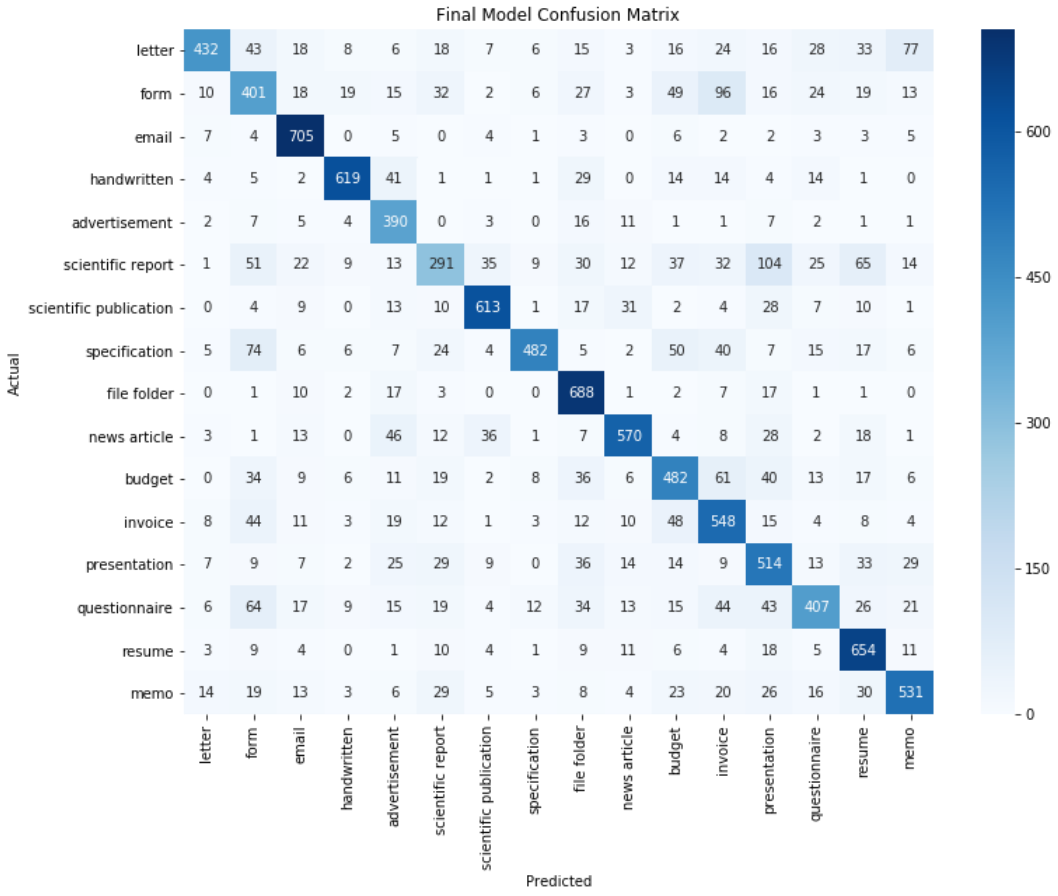
REGION	ACCURACY	RECALL	PRECISION	F1
CENTER	0.281	0.288	0.295	0.268
HEADER	0.167	0.179	0.426	0.426

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*).

This method performed poorly. The model training on only the center of the image performed with an accuracy of only 28.1% while the model training on only the header of the image performed with an accuracy of 16.7%. After examining the results, the models predicted most images as members of very few classes. This indicates that important predictive features are distributed across the entire image, meaning all regions of the image.

Final Model

The final model consisted of full, histogram equalized, *224x224* images, a batch size of 50, a learning rate of 0.001, max pooling, and a kernel architecture of 7-5-5-3-3. This model produced the following testing results:



As can be seen in the classification report below, emails, resumes, file folders, and scientific publications were the most correctly identified classes, with F1-scores of 0.91, 0.86, 0.85, and 0.81, respectively. Scientific reports, forms, questionnaires, and presentations were the most misclassified classes, with F1-scores of 0.47, 0.58, 0.60, and 0.62, respectively.

	Precision	recall	f1-score	support
letter	0.81	0.64	0.72	750
form	0.57	0.6	0.58	750
email	0.86	0.96	0.91	750
handwritten	0.97	0.71	0.82	750
advertisement	0.7	0.83	0.76	750
scientific report	0.51	0.43	0.47	750
scientific publication	0.76	0.86	0.81	750
specification	0.71	0.81	0.76	750
file folder	0.83	0.86	0.85	750
news article	0.76	0.78	0.77	750
budget	0.67	0.6	0.63	750
invoice	0.61	0.74	0.67	750
presentation	0.57	0.68	0.62	750
questionnaire	0.59	0.62	0.6	750
resume	0.88	0.84	0.86	750
memo	0.88	0.61	0.72	750
micro average	0.72	0.72	0.72	11700
macro average	0.73	0.72	0.72	11700
weighted average	0.73	0.72	0.72	11700

Classification Report for Final Model

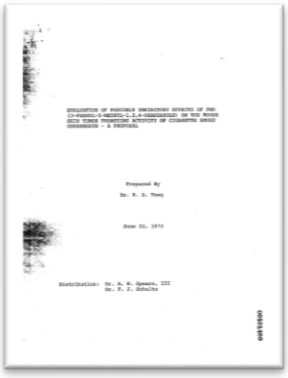
5. Summary and conclusions. Summarize the results you obtained, explain what you have learned, and suggest improvements that could be made in the future.

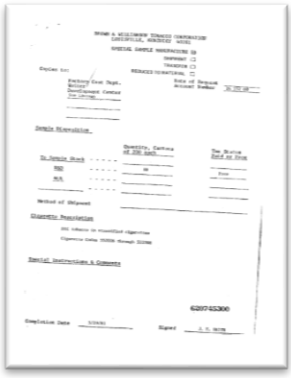
In general, the convolutional neural network proved to be an effective classifier of documents but had varied results on different classes. Some classes, including emails, resumes, file folders, and scientific publications predicted very well, with F1-scores of 0.91, 0.86, 0.85, and 0.81, respectively, while others, including scientific reports, forms, questionnaires, and presentations, predicted relatively poorly, with F1-scores of 0.47, 0.58, 0.60, and 0.62, respectively. The best parameters, including batch size, kernel size, and learning rate, were shown to be those in the middle of their respective spectrums.

Document images are more difficult to identify than natural images. Whereas most natural images attempt to locate a specific object within the overall image, documents require the analysis of multiple elements of each image for classification. Elements in document headers, footers, and bodies all factor into correct classification. In addition, documents contain small features that require input sizes to be kept relatively large, requiring more parameters to be trained and a longer training time.

High intraclass variance also exists in all of the class and makes identification even more difficult. As can be seen below in the two worst performing classes (scientific report and form), documents vary considerably within each class, though variance is greater in some classes over others. Classes with more strict document structures, such as emails and resumes, predicted better than classes with varying structure.

Scientific Report





Form

Several improvements can be made to this project. Classes could be merged to make fewer, more homogeneous classes to reduce the size of the dataset need for training and reduce intraclass variance. Regions of each image could be explored in separate models and combined using another classifier such as an SVM. Lastly, a top-3 or top-5 classifier could be developed rather than a top-1 classifier to provide more clarity into how the CNN model is misclassifying some of the worse predicting classes.

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 and7
- 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

amir-jafari. “Amir-Jafari/Deep-Learning.” *GitHub*, 18 Aug. 2018, github.com/amir-jafari/Deep-Learning/tree/master/Pytorch_/6-Conv_Mnist.

“API Reference¶.” *1.4. Support Vector Machines - Scikit-Learn 0.19.2 Documentation*, scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics.

Britz, Denny. “Understanding Convolutional Neural Networks for NLP.” *WildML*, 10 Jan. 2016, www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/.

Budhiraja, Amar. “Learning Less to Learn Better - Dropout in (Deep) Machine Learning.” *Medium.com*, Medium, 15 Dec. 2016, medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5.

Cascade Classification - OpenCV 2.4.13.7 Documentation, docs.opencv.org/3.1.0/d5/daf/tutorial_py_histogram_equalization.html.

COSTE, Arthur. “Project 1 : Histograms.” *Summation of Twitches and Tetanization*, 5 Sept. 2012, www.sci.utah.edu/~acoste/uou/Image/project1/Arthur_COSTE_Project_1_report.html.

CS231n Convolutional Neural Networks for Visual Recognition, cs231n.github.io/convolutional-networks/.

“PyTorch Documentation¶.” *PyTorch*, pytorch.org/docs/stable/index.html.

Tensmeyer, Chris, and Tony Martinez. "Analysis of Convolutional Neural Networks for Document Image Classification." *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, August 10, 2017. doi:10.1109/icdar.2017.71.

“The RVL-CDIP Dataset.” *Carnegie Melon University*, www.cs.cmu.edu/~aharley/rvl-cdip/.