Machine Learning II
Final Report
Group 5

Identifying Digitally-Scanned Documents with Convolutional Neural Networks

Jackson Crum
Geetha Satya Mounika Ganji
Renu Gopal Reddy Durgampudi

Introduction

Convolutional neural networks were 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.

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.[1]

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.

Dataset

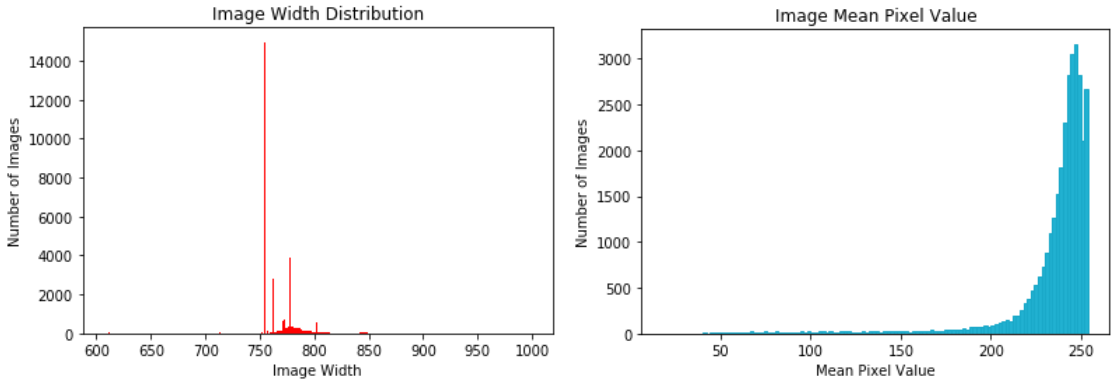
The data comes the RVL-CDIP Dataset (Ryerson Vision Lab Complex Document Information Processing). This dataset consists of 400,000 greyscale images of 16 classes of documents. The images are presplit into 320,000 training images, 40,000 validation images, and 40,000 testing images. The data consists of 16 document classes: letter, form, email, handwritten, advertisement, scientific report, scientific publication, specification, file folder, news article, budget, invoice, presentation, questionnaire, resume, and memo.[2]

[1] 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.

[2] "The RVL-CDIP Dataset." *Carnegie Melon University,* www.cs.cmu.edu/~aharley/rvl-cdip/.

With 320,000 training images and 16 classes, the dataset provides roughly 20,000 training images per class, far more than need to effectively train a neural network efficiently. For the sake of efficiency, random class subsets were taken from the training, validation, and testing datasets to scale down the overall data size while maintaining class proportions.

While the majority of images are 90% white space or greater, a few images consisted of majority black space. Almost all images were of between the pixel dimensions of *1000x750* and *1000x800*, with the majority of these images being of size *1000x752*.



Convolutional Neural Networks

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.

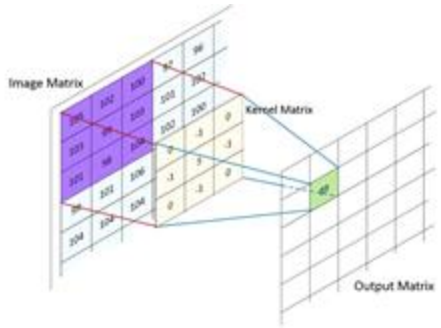
Input Layer

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.

Convolutional Layers

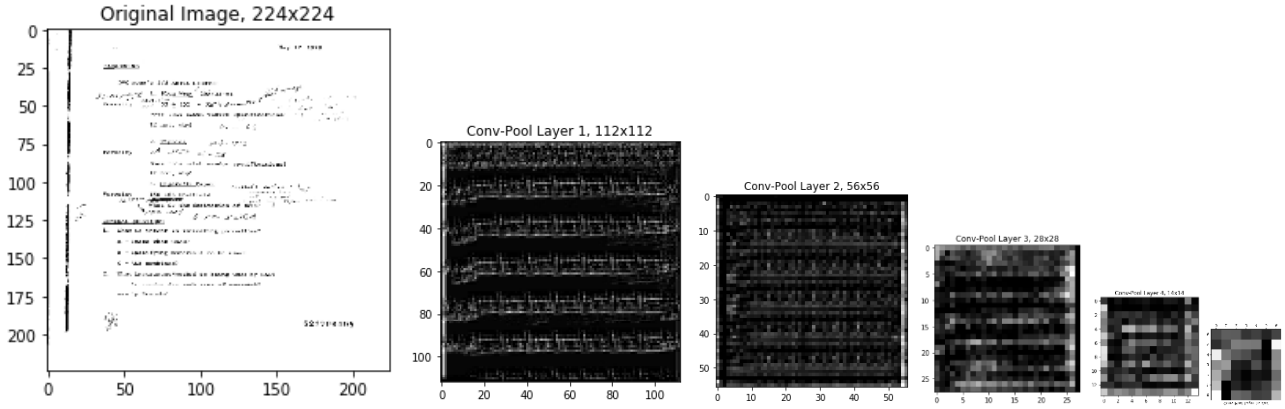
The convolution layer consists of a set of learnable kernels, which are smaller matrices of randomly initialized weight values in a 3-dimensional matrix (number of color channels, height, width). In the feedforward process, the kernels slide (convolve) over each element of the input matrix and compute 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 of Kernels

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

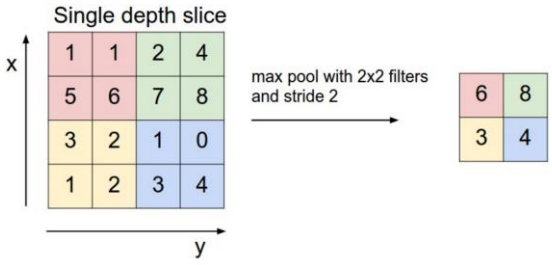
Following the batch normalization, a ReLU (Rectified Linear Units) activation layer is applied. The ReLU layer applies the function *f(x) = max(0, x)* to all of the values in the input, where *x* represents the input value. Essentially, the function sets all negative inputs to 0 and sets the output as 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%. [3]



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.

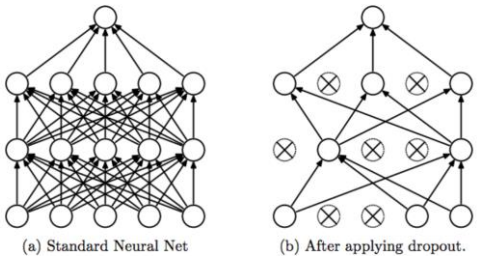
Fully Connected Layers

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.

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

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 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.[4]



Example of Dropout with Deactivated Neurons

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:

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

[4] 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.

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.[5]

Experimental Setup

Preprocessing

Histogram equalization was performed to increase the global contrast of each image and extract more faded text from each document. Histogram equalization is an image processing method that attempts to flatten the histogram of pixel values by linearizing the histogram of the cumulative sum of pixel values. This brings out areas of lower local contrast by equally distributing high intensity values. The equation for the 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.[6]

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

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

PyTorch

To load and prepare custom image data for analysis, PyTorch has an abstract Dataset class that representing a dataset. The custom dataset inherits Dataset functionality and overrides the

---

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

[6] 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.

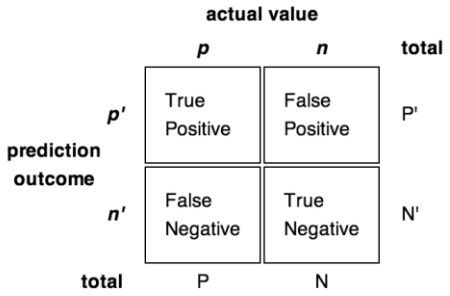
"__len__" (size of the dataset) and "__getitem__" (extract and index data) methods. The provide the custom data loader with a simple method for importing the images, a reference csv is first created for both the training and testing datasets with the full image path and the numeric label. This csv is passed to the custom data loader. Within the __getitem__ method, each image path is read and the image imported as a numpy array of pixel values. Any image operations, including preprocessing, resizing, and sectioning are conducted here. Finally, the image and label are exported to PyTorch's DataLoader to create the data in the format required.[7]

The model is built as a custom CNN( ) class in a series of sequential layer blocks using the premade layers from PyTorch's nn package. The convolution blocks consist of a convolution layer, batch normalization, ReLU activation, and a pooling layer. Each convolution layer takes as arguments the number of inputs, number of outputs (number of feature maps), number of padding layers, and size of the stride. After defining each layer block, fully connected layers (called Linear layers) are defined with the number input and output neurons. The input size of first linear layer is the size of the final pooling output (a *7x7* output with 32 feature maps results in a linear layer with 1568 neurons. Dropout layers are added between each fully connected layer as this is the location of the majority of the parameters and where overfitting is most likely to occur. The feedforward process is defined by passing the inputs through each defined layer in a sequential order.

PyTorch provides simple functions for conducting feed forward, loss calculation, optimizing, and gradient updating through backpropagation. The images are passed through the model through iteration of a user defined batch size, creating a 4-dimensional input of size *(batch size, # of color channels, image height, image width)*. The images and labels are converted to tensors and then PyTorch variables using the Variable class and run of the GPU. The images are fed to the CNN( ) class, loss is calculated, and gradients are updated with backpropagation with minimal code. The trained CNN is then tested on the testing dataset and the results converted back to numpy arrays for analysis.[8]

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

Batch size is used to updating weight and bias parameters using the average output of n-number of inputs to limit the amount of parameter updating required by the network. Picking a suitable batch size is vital for producing an efficient model. Using all the training data (1 batch) to update parameters is highly computational expensive due to the large input size while updating after each input creates noise if the sample is not a good representation of the whole data. Mini-batches are smaller batches of a portion of the data used to compromise between efficiency and noise. In this project, mini-batches are used to run inputs through the model and mini-batch size is tested to ascertain the ideal size.



Model performance is 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:

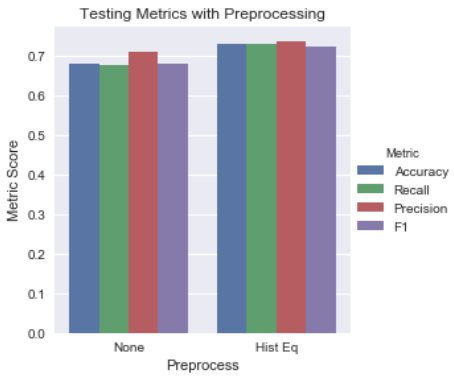
- Accuracy represents how often the model correctly predicts class and is calculated: *(TP + FN) / (TP + FP + TN + FN)*
- Recall represents what percentage of the inputs of a class are identified as that class and is calculated: *TP / (TP + FN)*
- Precision represents what percentage of the inputs predicted 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. Loss will also be monitored through the input iterations and convergence on the minimum of the loss function will be analyzed for model comparison.[9]

Training parameters are be examined in the PyTorch model by adjusting each one in turn while keeping other parameters constant and compared using the above metrics. Parameters tested included image size, mini-batch size, learning rate, kernel size, type of pooling, region testing, and preprocessing techniques.

Results

Histogram Equalization



Models were trained over 10 epochs with the original image pixel values and with histogram equalized pixel values.[10] 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

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

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

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

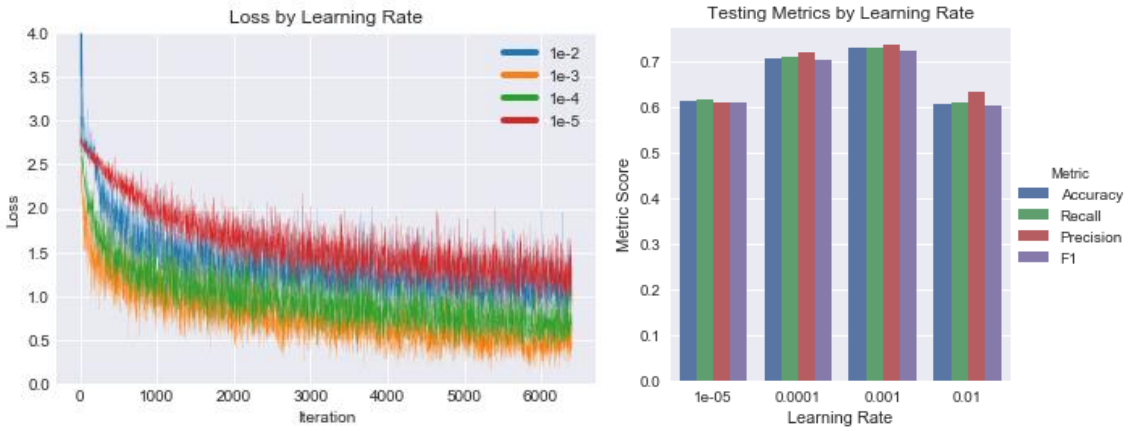
Models with varying batch sizes were trained and compared. Batch sizes include 1, 10, 50, and 100 images per batch. As can be seen in the metrics plot, varying the batch size produced unreliable and nonsensical results in PyTorch. Metrics should be best with the smallest batch size as this model would 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,

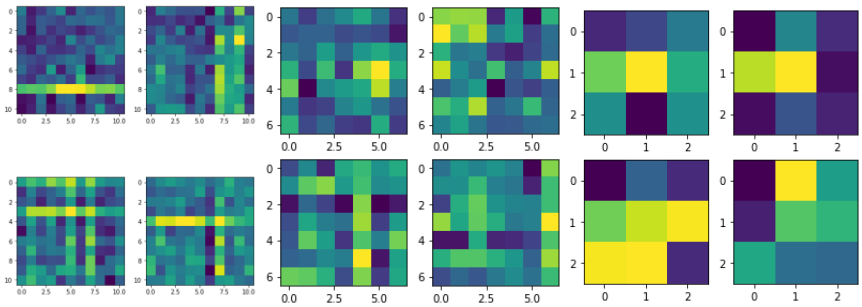
while a smaller learning rate will learn model parameters slower but provide a smoother convergence on the minimum of the loss function.

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.

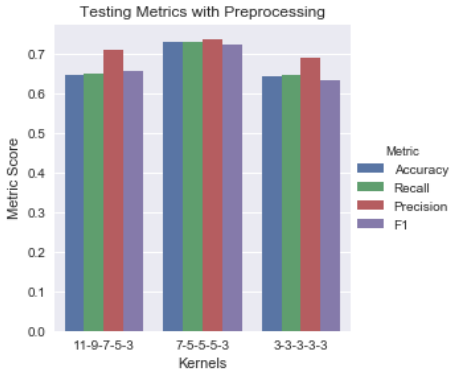


11 x 11 Kernels *7 x 7 Kernels* *3 x 3 Kernels*

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

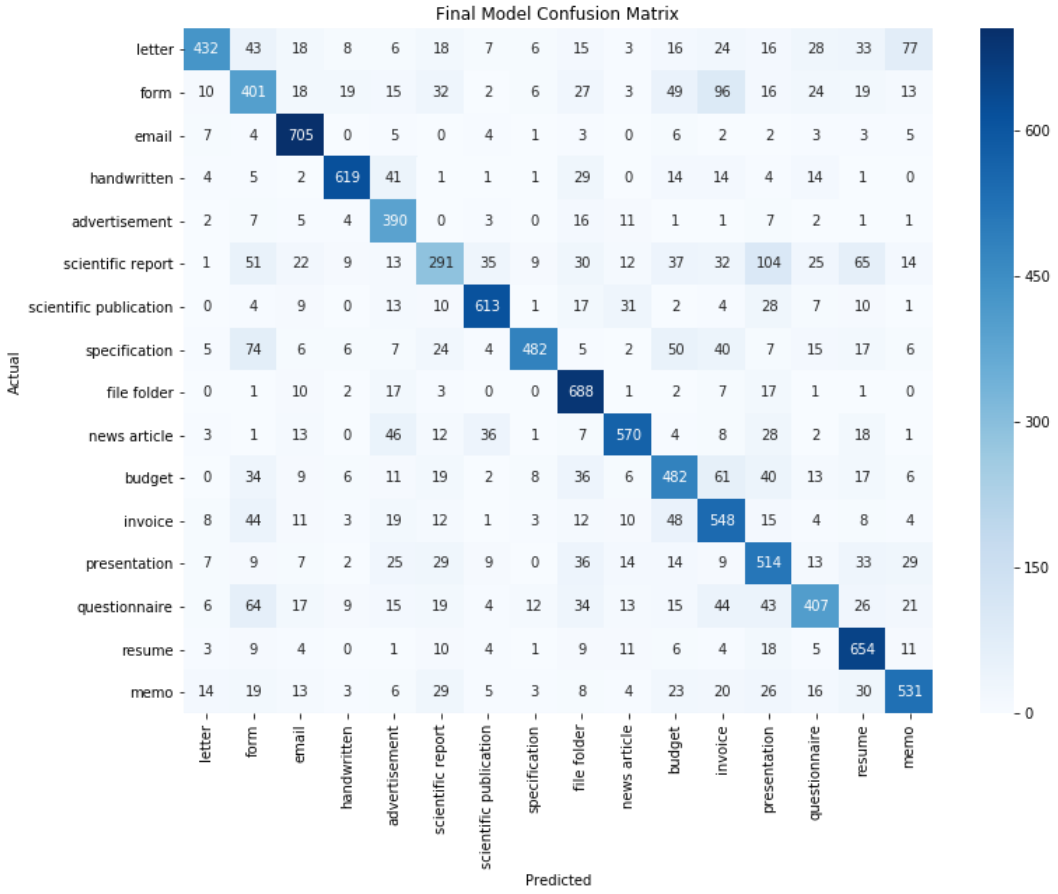
Manual examination of the documents led to the observation that most of the features of the document exist within the header or the center of the document. To test if running specific regions of the document through the network would produce equal classification metrics, two regions of the document were 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*).

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

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

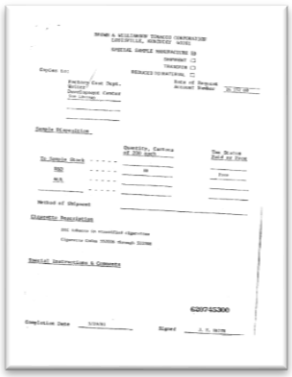
Summary and Conclusions

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.

Keras:

Data Preprocessing:

When using Tensorflow as backend, keras CNN requires a 4D array as input with shape.

1. Data is loaded as files and labels separately

```
train_files, train_targets = load_dataset('/home/ubuntu/Deep-Learning/Project/Regional/train/right_body')
valid_files, valid_targets = load_dataset('/home/ubuntu/Deep-Learning/Project/Regional/valid/right_body')
test_files, test_targets = load_dataset('/home/ubuntu/Deep-Learning/Project/Regional/test/right_body')
```

2. When using Keras CNN, 4D array is required as input with shape.
3. The convert_4darray function takes image as input returns a required 4D array suitable for CNN. The function first loads image and resizes it to an image that 224*224.
4. The convert_4darrays function takes numpy array of image paths as input and returns a 4D array with shape.
5. Images are rescaled by dividing every pixel in every image by 255.

Model Building:

1. **CNN:**

Stride:

This controls how the filter convolves around the input image. Simply we can say that, stride is the amount by which filter shifts. Implicit value of stride is 1. This is normally set in a way that the output is an integer not a fraction.

Padding:

When analyzing an image, the pixel in the corner will be covered only once whereas the pixel in the center of the image will be more than once which has 2 diadvanatages:

a. Shrinking the output and
b. May loose information on the corners of image which is solved by using padding

Padding is like an additional layer that can be added to the corner/border of the image to prevent loss of information.

If you have a stride of 1 and if you set the size of zero padding to

$$Zero\ Padding = \frac{(K-1)}{2}$$

K= filter size, then the input and output will always have the same dimensions.

The formula for calculating output size of any convolution layer is:

$$O = \frac{(W-K+2P)}{S} + 1$$

O is the output height/width, W is the input height/width, K is kernel size, P is padding and S is stride.

Architecture of CNN built:

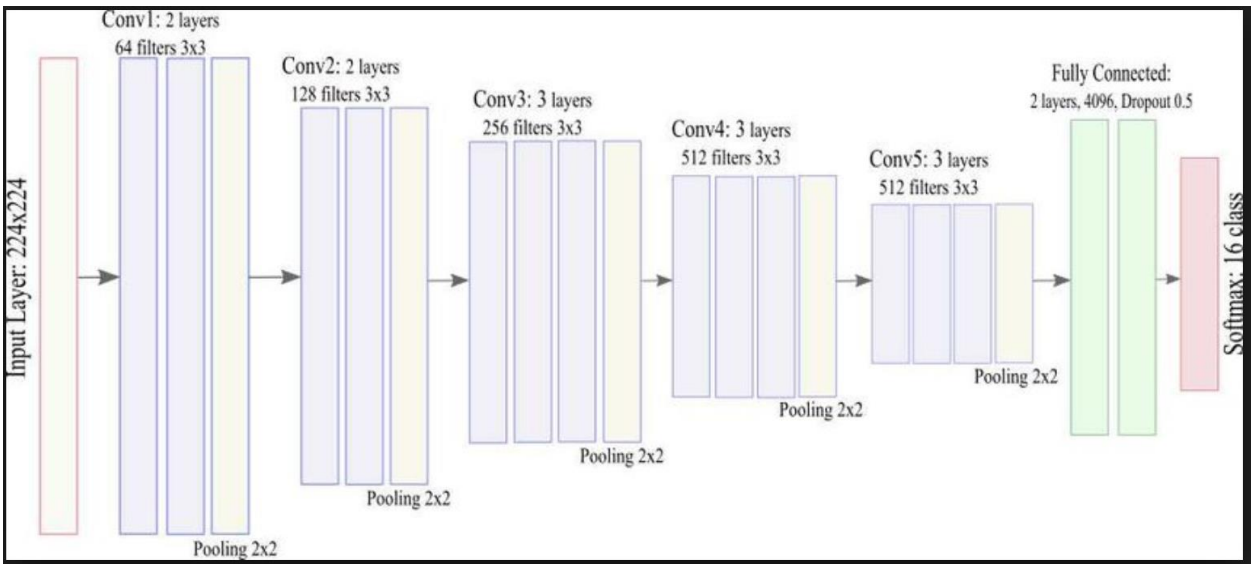
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 32)      4736
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 74, 74, 32)        25632
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 24, 24, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 24, 24, 32)        9248
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 8, 8, 32)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 32)          9248
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 32)          9248
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 2, 2, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                4128
_________________________________________________________________
dense_2 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_3 (Dense)              (None, 16)                528
=================================================================
Total params: 63,824
Trainable params: 63,824
Non-trainable params: 0
_________________________________________________________________
```

Result: Low accuracy – 8.125% with number of epochs =10

2. **VGG16:**

   VGG-16 layer is also called as Oxfordnet. Also called as Visual Geometry Group of Oxford named after who developed it. It is a simplified version of AlexNet and yields better results than AlexNet. This is built using convolution layers of 3*3 kernel size, stride = 1 and padding as ‘same’. All max pooling performed will be of size 2*2, stride = 2. Fully connected layers at end and total of 16 layers. The transfer function used in convolution layers and fully converted layers is ReLU.

   Architecture of VGG-16:



In the source code there is a code called applications from where VGG-16 is imported. This is pre-implemented. VGG-16 is trained on imagenet. They have taken imagenet dataset onto VGG network and the model is trained. So we actually get a model with all weights filled in which means we can reuse this model which is readily available in Keras. So we get VGG-16 trained and built in Keras. No need to choose the kernel sizes and strides.

Format:

VGG16(include_top = True, weights = 'None', input_tensor = None, input_shape = None, pooling = None, classes=1000)

Arguments:

1. **include_top:** decided whether to include 3-fully connected layers at the top of the network
2. **weights:**
    a. **None:** the weights are randomly initialized.
    b. **imagenet:** uses weights form pre-trained Imagenet.
3. **input _tensor:** optional. This is output of layers.Input()
4. **input_shape:** optional. Only specified if include_top is specified as False. Else the input_shape has to be 224*224 which channels 3.
5. **pooling:** optional. Used for feature extraction when include_top is specified as False.
    a. **None:** output of model will be a 4Darray output of the last convolution layer
    b. **avg:** global average pooling will be applied to the output of last convolution layer.
    c. **max:** global max pooling will be applied to the output of last convolution layer.
6. **classes:** optional. Only to be specified if include_top is False, and weights is specified as 'None'. Specifies number of classes to classify images into.

In constructing VGG-16 n RVL-CDIP dataset, weights are specified as 'imagenet' and input_shape as (224,224,3).

Result: Accuracy of 67% with number epochs =10

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/.

Code

Extracting and Sorting

```
import os
import random
os.chdir('C:/Users/sjcrum/Documents/Machine Learning II/Final Project')
labels = open('labels/val.txt', 'r')
labels = labels.read()


classes = {'0': 'letter', '1': 'form', '2': 'email', '3': 'handwritten', '4': 'advertisement', '5': 'scientific report',
          '6': 'scientific publication', '7': 'specification', '8': 'file folder', '9': 'news article', '10': 'budget',
          '11': 'invoice', '12': 'presentation', '13': 'questionnaire', '14': 'resume', '15': 'memo'}

root_path = 'C:/Users/sjcrum/Documents/Machine Learning II/Final Project/images/images'

def make_doc_dirs(path):
    for label in classes.values():
        if not os.path.exists(path + str(label)):
            os.mkdir(path + str(label))
#         else:
#             print("Directory " , str(label) ,  " already exists")

make_doc_dirs('C:/Users/sjcrum/Documents/Machine Learning II/Final Project/DocImages/training/')
make_doc_dirs('C:/Users/sjcrum/Documents/Machine Learning II/Final Project/DocImages/testing/')
make_doc_dirs('C:/Users/sjcrum/Documents/Machine Learning II/Final Project/DocImages/validation/')
make_doc_dirs('C:/Users/sjcrum/Documents/Machine Learning II/Final Project/DocImages/experiment/')


labels_split = labels.split('images')
full_paths = []
for image_label in labels_split:
    label = root_path + image_label
    full_paths.append(label)


print(len(full_paths))
print(full_paths[0:10])


full_paths = full_paths[1:]
#print(full_paths[:10])
paths = []
for path in full_paths:
    path = path.split('\n')
    for p in path:
        p = p.rsplit(' ',1)
        paths.append(p)
```

```
paths = [x for x in paths if x != ["]]
print(len(paths))
print(paths[:5])


# In[6]:


i = 0
for p in paths[:20000]:
    old_path = p[0]
    full_path = p[0].split('images/')
    doctype = p[1]
    #print(doctype)
    for key, value in classes.items():
        if doctype == key:
            doctype = value
    root_path = full_path[0]
    image_filename = full_path[1].rsplit('/', 1)[1]
    #new_path = root_path +'DocImages/training/'+str(doctype) + '/' + image_filename
    #new_path = root_path +'DocImages/training/'+str(doctype) + '/' + image_filename
    new_path = root_path +'DocImages/validation/'+str(doctype) + '/' + image_filename
    if not os.path.exists(new_path):
        os.rename(old_path, new_path)
        i += 1
        if i % 1000 == 0:
            print(str(i) + "/20000")
            print(old_path)
            print(new_path)


print(paths[1])

make_doc_dirs('C:/Users/sjcrum/Documents/Machine Learning II/Final Project/DocumentImages/train/')
make_doc_dirs('C:/Users/sjcrum/Documents/Machine Learning II/Final Project/DocumentImages/test/')
make_doc_dirs('C:/Users/sjcrum/Documents/Machine Learning II/Final Project/DocumentImages/valid/')


#path = 'C:/Users/sjcrum/Documents/Machine Learning II/Final Project/DocImages/training/'
#path = 'C:/Users/sjcrum/Documents/Machine Learning II/Final Project/DocImages/testing/'
path = 'C:/Users/sjcrum/Documents/Machine Learning II/Final Project/DocImages/validation/'
roots = []
for root, dirs, files in os.walk(path):
    roots.append(root)

print(roots[1:])


random.seed(29)

i = 1
```

```
for root in roots[1:]:
    print("Subsetting images from class {}/16".format(i))
    print(root)
    images = []
    for root, dirs, files in os.walk(root):
        for file in files:
            images.append(str(root)+os.sep+str(file))
    random.shuffle(images)
    #images_subset = images[:2000]   #for training images
    images_subset = images[:750]     #for testing and validation images
    print(len(images_subset))
    for image in images_subset:
        split_im = image.split("DocImages/validation")
        os.rename(image, split_im[0]+'DocumentImages/valid'+split_im[1])
    i += 1
```

EDA

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
import seaborn as sns
import random

np.random.seed(29)  # For numpy numbers
random.seed(29)   # For Python

# Extract all image paths
def extract_file_paths(path):
    image_filenames = []
    for root, dirs, files in os.walk(path):
        if len(files) > 0:
            for file in files:
                if(file[-3:] == "tif" or file[-3:] == "Tif"):
                    image_filenames.append(str(root)+os.sep+str(file))
    print(len(image_filenames))
    return image_filenames

image_files = extract_file_paths("C:/Users/sjcrum/Documents/Machine Learning II/Final
Project/DocumentImages/train")

#Get image dimensions
height = []
width = []
docs = []

for image in image_files:
    path = os.path.dirname(image)
    base = os.path.basename(path)
```

```
    docs.append(base)
    img = cv2.imread(image)
    height.append(img.shape[0])
    width.append(img.shape[1])
    #Print images wider than taller
    if img.shape[1] > 1000:
        print(base)
        print(image)
        plt.imshow(img)
        plt.show()


image_dim = {"Document": docs, "Height": height, "Width": width}
image_dim_df = pd.DataFrame(image_dim, columns = ["Document", "Height", "Width"])


image_dim_df.groupby('Document').quantile([0, 0.25, 0.5, 0.75, 1])  #Quantiles by class


print(len(width))
width1 = [w for w in width if w <= 1000]
plt.hist(width, bins = range(min(width1), max(width1) + 1, 1), color = 'red')
#sns.distplot(width, hist=True, kde=True, bins=5, color = 'darkblue', hist_kws={'edgecolor':'black'},
kde_kws={'linewidth': 1})
plt.title('Image Width Distribution')
plt.xlabel('Image Width')
plt.ylabel('Number of Images')
plt.show()


#Getting mean pixel value of each image

pixel_means = []

for image in image_files:
    img = cv2.imread(image)
    pixel_mean = np.mean(np.array(img))
    pixel_means.append(pixel_mean)



pixel_means_int = [int(x) for x in pixel_means]
plt.hist(pixel_means, bins = range(min(pixel_means_int), max(pixel_means_int) + 2, 2), color =
'#21B1D1', edgecolor = '#1AA9C9')
plt.title('Image Mean Pixel Value')
plt.xlabel('Mean Pixel Value')
plt.ylabel('Number of Images')
plt.show()
```

```
#Identifying the images with highest (whitest) and lowest (blackest) mean pixel values

max_index = pixel_means.index(max(pixel_means))
min_index = pixel_means.index(min(pixel_means))
max_image = image_files[max_index]
min_image = image_files[min_index]
max_img = cv2.imread(max_image, 0)
min_img = cv2.imread(min_image, 0)
max_path = os.path.dirname(max_image)
min_path = os.path.dirname(min_image)
max_doctype = max_path.split('\\')[1]
min_doctype = min_path.split('\\')[1]
plt.imshow(min_img, cmap = 'gray')
plt.title(min_doctype.title() + "\nmean pixel value = {}".format(min(pixel_means).round(2)))
plt.show()
plt.imshow(max_img, cmap = 'gray')
plt.title(max_doctype.title() + "\nmean pixel value = {}".format(max(pixel_means).round(2)))
plt.show()


#Creating images at half size of the previous for visual examination

full_image = cv2.imread('C:/Users/sjcrum/Documents/Machine Learning II/Final
Project/DocumentImages/train/letter/0000116633.tif', 0)
height, width = full_image.shape
print(full_image.shape)
cv2.imwrite('C:/Users/sjcrum/Documents/Machine Learning II/Final Project/Report/Images/fullimage.tif',
full_image)
half_image = cv2.resize(full_image, (int(width/2), int(height/2)))
print(half_image.shape)
cv2.imwrite('C:/Users/sjcrum/Documents/Machine Learning II/Final
Project/Report/Images/halfimage.tif', half_image)
quarter_image = cv2.resize(full_image, (int(width/4), int(height/4)))
print(quarter_image.shape)
cv2.imwrite('C:/Users/sjcrum/Documents/Machine Learning II/Final
Project/Report/Images/quarterimage.tif', quarter_image)
eighth_image = cv2.resize(full_image, (int(width/8), int(height/8)))
print(eighth_image.shape)
cv2.imwrite('C:/Users/sjcrum/Documents/Machine Learning II/Final
Project/Report/Images/eighthimage.tif', eighth_image)
sixteenth_image = cv2.resize(full_image, (int(width/16), int(height/16)))
print(sixteenth_image.shape)
cv2.imwrite('C:/Users/sjcrum/Documents/Machine Learning II/Final
Project/Report/Images/sixteenthimage.tif', sixteenth_image)
```

Preprocessing

```
import cv2
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
import os, random
random.seed(29)

def random_photo_per_class(path):
    random_images = []
    for root, dirs, files in os.walk(path):
        if root[-5:] != "train":
            image = random.choice(os.listdir("{}".format(root)))
            random_images.append(str(root)+os.sep+str(image))
    return random_images

random_photo_per_class("C:/Users/sjcrum/Documents/Machine Learning II/Final
Project/DocumentImages/train")



examples = random_photo_per_class("C:/Users/sjcrum/Documents/Machine Learning II/Final
Project/DocumentImages/train")



for image in examples:
    directory = os.path.dirname(image)
    doctype = directory.rsplit('\\', 1)[-1]
    img = cv2.imread(image, 0)

    equ = cv2.equalizeHist(img)
    res = np.hstack((img,equu))

    plt.subplot(111)
    plt.imshow(res, cmap='Greys_r')
    plt.title('{}'.format(doctype))
    plt.xticks([])
    plt.yticks([])

    plt.show()


    #np.histogram(1d array of img values, 256 bins between range 0 and 255, inclusive), returns histogram
values and bin edges



#### Hard Coding Histogram Equalization with Code from OpenCV ####

img = cv2.imread('C:/Users/sjcrum/Documents/Machine Learning II/Final
Project/DocumentImages/train/advertisement/502599496.tif',0)

#get histogram (count of each value) values and bin edges for all pixel values
hist,bins = np.histogram(img.flatten(),256,[0,256])
```

```
#cumulative sum to get cumulative distribution function
cdf = hist.cumsum()
#Normalize cdf by multiplying by most common pixel value divided by sum of pixel values
cdf_normalized = cdf * hist.max()/ cdf.max()


cdf_m = np.ma.masked_equal(cdf,0)
#min-max normalizing to [0,255] range
cdf_m = (cdf_m - cdf_m.min())*255/(cdf_m.max()-cdf_m.min())
cdf = np.ma.filled(cdf_m,0).astype('uint8')

img2 = cdf[img]

hist2,bins2 = np.histogram(img2.flatten(),256,[0,256])

cdf2 = hist2.cumsum()
cdf_normalized2 = cdf2 * hist2.max()/ cdf2.max()

plt.subplot(221)
plt.plot(cdf_normalized, color = 'b')
plt.hist(img.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.legend(('cdf','histogram'), loc = 'upper left')

plt.subplot(222)
plt.imshow(img, cmap='Greys_r')

plt.subplot(223)
plt.plot(cdf_normalized2, color = 'b')
plt.hist(img2.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.legend(('cdf','histogram'), loc = 'upper left')

plt.subplot(224)
plt.imshow(img2, cmap='Greys_r')

plt.show()
print(len(cdf))
print(len(cdf_m))



image = cv2.imread("C:/Users/sjcrum/Documents/Machine Learning II/Final
Project/DocumentImages/test/advertisement/502607274+-7274.tif", 0)

plt.imshow(image, cmap='gray')
plt.show()


height, width = img.shape
```

```
header = img[0:(int(height*0.33)), 0:width]
footer = img[int(height*0.67):height, 0:width]
lbody = img[int(height*0.33):int(height*0.67), 0:int(width*0.5)]
rbody = img[int(height*0.33):int(height*0.67), int(width*0.5):width]
center = img[int(height*0.33):int(height*0.67), int(width*0.25):int(width*0.75)]


plt.imshow(img, cmap = "gray")
plt.show()
plt.imshow(header, cmap = "gray")
plt.show()
plt.imshow(lbody, cmap = "gray")
plt.show()
plt.imshow(rbody, cmap = "gray")
plt.show()
plt.imshow(footer, cmap = "gray")
plt.show()
plt.imshow(center, cmap = "gray")
plt.show()


### Create train/test/valid datasets for each region####
### Later not used ####

def create_regions(folder, dataset):
    image_paths = []
    for root, dirs, files in os.walk(folder):
        for file in files:
            image_paths.append(str(root)+os.sep+str(file))


    path = 'C:/Users/sjcrum/Documents/Machine Learning II/Final
Project/RegionalImages/{}/'.format(dataset)
    print(path)
    regions = ['whole', 'header', 'footer', 'left_body', 'right_body']
    classes = {'letter': '0', 'form': '1', 'email': '2', 'handwritten': '3', 'advertisement': '4', 'scientific report': '5',
      'scientific publication': '6', 'specification': '7', 'file folder': '8', 'news article': '9', 'budget': '10',
      'invoice': '11', 'presentation': '12', 'questionnaire': '13', 'resume': '14', 'memo': '15'}

    for region in regions:
        new_path = path + region
        if not os.path.exists(new_path):
            os.makedirs(new_path)
        for cl in classes.keys():
            class_path = new_path + '/' + cl
            if not os.path.exists(class_path):
                os.makedirs(class_path)
```

```
    for image in image_paths:
        img = cv2.imread(image,0)
        height = img.shape[0]
        width = img.shape[1]
        split_im = image.split('\\', 1)
        print(split_im)
        cv2.imwrite('{}/whole/{}'.format(path, split_im[1]), img)
        header = img[0:(int(height*0.33)), 0:width]
        cv2.imwrite('{}/header/{}'.format(path, split_im[1]), header)
        footer = img[int(height*0.67):height, 0:width]
        cv2.imwrite('{}/footer/{}'.format(path, split_im[1]), footer)
        left_body = img[int(height*0.33):int(height*0.67), 0:int(width*0.5)]
        cv2.imwrite('{}/left_body/{}'.format(path, split_im[1]), left_body)
        right_body = img[int(height*0.33):int(height*0.67), int(width*0.5):width]
        cv2.imwrite('{}/right_body/{}'.format(path, split_im[1]), right_body)

create_regions("C:/Users/sjcrum/Documents/Machine Learning II/Final Project/DocumentImages/train",
"train")
create_regions("C:/Users/sjcrum/Documents/Machine Learning II/Final Project/DocumentImages/valid",
"valid")


path = 'C:/Users/sjcrum/Documents/Machine Learning II/Final Project/RegionalImages/'
regions = ['whole', 'header', 'footer', 'left_body', 'right_body']
for region in regions:
    new_path = path + region
    if not os.path.exists(new_path):
        os.makedirs(new_path)



path = 'C:/Users/sjcrum/Documents/Machine Learning II/Final Project/RegionalImages/valid'
def make_regional_folders(path):

    regions = ['whole', 'header', 'footer', 'left_body', 'right_body']

    classes = {'letter': '0', 'form': '1', 'email': '2', 'handwritten': '3', 'advertisement': '4', 'scientific report': '5',
      'scientific publication': '6', 'specification': '7', 'file folder': '8', 'news article': '9', 'budget': '10',
      'invoice': '11', 'presentation': '12', 'questionnaire': '13', 'resume': '14', 'memo': '15'}

    for region in regions:
        path = 'C:/Users/sjcrum/Documents/Machine Learning II/Final
Project/RegionSubset/valid/{}'.format(region)
        if not os.path.exists(path):
            os.makedirs(path)
        for i in list(classes.keys()):
            path = 'C:/Users/sjcrum/Documents/Machine Learning II/Final
Project/RegionSubset/valid/{}/{}'.format(region, i)
            if not os.path.exists(path):
                os.makedirs(path)
```

```
make_regional_folders(path)


def subset_regional(path):
    images = []
    for root, dirs, files in os.walk(path):
        for file in files:
            images.append(str(root)+os.sep+str(file))
    random.shuffle(images)
    print(len(images))
    sub_images = images[:int(0.25 * len(images))]
    print(len(sub_images))
    for im in images:
        img = im.split("RegionalImages")
        print(img)
        #os.rename('')


subset_regional('C:/Users/sjcrum/Documents/Machine Learning II/Final
Project/RegionalImages/valid/header/budget')
```

Making CSV

```
import os
import random
import pandas as pd
import numpy as np
import cv2

classes = {'letter': '0', 'form': '1', 'email': '2', 'handwritten': '3',
'advertisement': '4', 'scientific report': '5',
           'scientific publication': '6', 'specification': '7', 'file folder': '8',
'news article': '9', 'budget': '10',
           'invoice': '11', 'presentation': '12', 'questionnaire': '13', 'resume':
'14', 'memo': '15'}

def extract_paths(path):
    image_filenames = []
    for root, dirs, files in os.walk(path):
        if len(files) > 0:
            for file in files:
                image_filenames.append(str(root)+os.sep+str(file))
    return image_filenames

filenames_test = extract_paths('/home/ubuntu/MachineLearningII/test/')
filenames_train = extract_paths('/home/ubuntu/MachineLearningII/train/')

def make_csv(files, dataset):
    images_and_labels = []
    for image in files:
        print(image)

        base = os.path.dirname(image).rsplit("/", 1)

        class_lab = base[1]
```

```
            label = classes[class_lab]
            image_and_label = [image, label]
            images_and_labels.append(image_and_label)

    df = pd.DataFrame(images_and_labels,columns=['image_paths','labels'])

#df.to_csv('/home/ubuntu/MachineLearningII/{}_images_and_labels_1_image.csv'.format(da
taset), index = False)

make_csv(filenames_test, 'test')
make_csv(filenames_train, 'train')
```

CNN PyTorch

```
import pandas as pd
import numpy as np
import cv2
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torchvision.transforms as transforms
from torch.autograd import Variable
from torch.utils.data import DataLoader
import torchvision.transforms as transforms
from torch.utils.data.dataset import Dataset
import time
# ----------------------------------------------------------------------------------------

image_size = 224

class CustomDatasetFromImages(Dataset):
    def __init__(self, csv_path, transforms):
        # Read the csv file
        self.data_info = pd.read_csv(csv_path)
        # First column contains the image paths
        self.image_array = np.asarray(self.data_info.iloc[:, 0])
        # Second column is the labels
        self.label_array = np.asarray(self.data_info.iloc[:, 1])
        # Calculate len
        self.data_len = len(self.data_info.index)
        self.transforms = transforms

    def __getitem__(self, index):
        # Get image name from the pandas df
        single_image_name = self.image_array[index]
        img_as_img = cv2.imread(single_image_name, 0)

        #####This area for testing of various image preprocessing techniques#####
        # Always end with img_resized for easy modularity#

        ###Histogram Equalization###
        equ = cv2.equalizeHist(img_as_img)
        img_resized = cv2.resize(equ, (image_size, image_size))


        ###Image Header###
        #height, width = img_as_img.shape
        #img_header = img_as_img[0:(int(height*0.33)), 0:width]
        #img_resized = cv2.resize(img_header, (image_size, image_size))
```

```
        ###Image Center###
        #img_center =
img_as_img[int(height*0.33):int(height*0.67),int(width*0.25):int(width*0.75)]
        #img_resized = cv2.resize(img_center, (image_size, image_size))

        img_final = np.expand_dims(img_resized, 0)
        single_image_label = self.label_array[index]
        # Return image and the label
        return (img_final, single_image_label)


    def __len__(self):
        return self.data_len

if __name__ == '__main__':
    train_loader =
CustomDatasetFromImages('/home/ubuntu/MachineLearningII/train_images_and_labels.csv',
transforms = transforms)
    test_loader =
CustomDatasetFromImages('/home/ubuntu/MachineLearningII/test_images_and_labels.csv',
transforms = transforms)

#transform = transforms.Compose([transforms.ToPILImage(), transforms.ToTensor()])

num_epochs = 10
batch_size = 50
learning_rate = 0.001

train = DataLoader(train_loader, batch_size = batch_size, shuffle=True)
test = DataLoader(test_loader, batch_size = batch_size, shuffle=True)

train_iter = iter(train)
test_iter = iter(test)

images, labels = train_iter.next()

print('images shape on batch size = {}'.format(images.size()))
print('labels shape on batch size = {}'.format(labels.size()))

# ------------------------------------------------------------------------------------
# CNN Model (2 conv layer)
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.layer1 = nn.Sequential(
            #1 input for grayscale, # of feature maps,
            nn.Conv2d(1, 32, kernel_size=9, padding=4, stride=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(32, 32, kernel_size=7, padding=3),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2))
        self.layer3 = nn.Sequential(
            nn.Conv2d(32, 32, kernel_size=5, padding=2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2))
        self.layer4 = nn.Sequential(
            nn.Conv2d(32, 32, kernel_size=5, padding=2),
            nn.BatchNorm2d(32),
```

```
            nn.ReLU(),
            nn.MaxPool2d(2))
        self.layer5 = nn.Sequential(
            nn.Conv2d(32, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2))
        self.fc1 = nn.Linear(7 * 7 * 32, 128)
        self.drop1 = nn.Dropout(0.2)
        self.fc2 = nn.Linear(128, 16)

    def forward(self, x):
        out = self.layer1(x.float())
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = self.layer5(out)
        out = out.view(out.size(0), -1)
        out = self.fc1(out)
        out = self.drop1(out)
        out = self.fc2(out)
        return out

# ----------------------------------------------------------------------------------------
cnn = CNN()
cnn.cuda()
# ----------------------------------------------------------------------------------------
# Loss and Optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(cnn.parameters(), lr=learning_rate)
# ----------------------------------------------------------------------------------------
# Train the Model

losses = []

start_time = time.time()
for epoch in range(num_epochs):
    print ("Starting Epoch {}".format(epoch + 1))
    train_iter = iter(train)
    i = 0
    for images, labels in train_iter:

        images = Variable(images).cuda()
        labels = Variable(labels).cuda()

        # Forward + Backward + Optimize
        optimizer.zero_grad()
        outputs = cnn(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        i += 1

        if (i) % 50 == 0:
            print('Epoch {}/{}, Iter {}/{}, Loss: {}'.format(epoch + 1, num_epochs, i,
(train_loader.data_len / batch_size), loss.item()))

        losses.append(loss.item())

    print("Epoch Done")

print("--- %s seconds ---" % (time.time() - start_time))
# ----------------------------------------------------------------------------------------
```

```
# Test the Model
cnn.eval()  # Change model to 'eval' mode (BN uses moving mean/var).
correct = 0
total = 0

im_labels = []
im_preds = []

for i, (images, labels) in enumerate(test_iter):
    images = Variable(images).cuda()
    outputs = cnn(images)
    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)
    correct += (predicted.cpu() == labels).sum()

    if (i) % 10 == 0:
        print('Epoch {}/{}, Iter {}/{}, Loss: {}'.format(epoch + 1, num_epochs, i,
(test_loader.data_len / batch_size),
                                                         loss.item()))

    im_labels.append(labels.cpu().numpy())
    im_preds.append(predicted.cpu().numpy())
# ------------------------------------------------------------------------------------
print('Test Accuracy of the model on the test images: {}'.format(100 * correct /
total))
# ------------------------------------------------------------------------------------
# Save the Trained Model
#torch.save(cnn.state_dict(), 'cnn-wholeimage-50-5ConvBlocks-5Kernel-3fc2DropsHistEq-
5Epochs-50batchsize.pkl')
#torch.save(cnn.state_dict(), 'cnn-wholeimage-224-4ConvBlocks-00001lr-11kernel-
Adam.pkl".pkl')

preds = [x for pred in im_preds for x in pred]
labels = [x for label in im_labels for x in label]

from sklearn.metrics import confusion_matrix, accuracy_score, f1_score,
precision_score, recall_score

cm = confusion_matrix(labels, preds)
print(cm)

acc = accuracy_score(labels, preds, sample_weight=None)
print("Accuracy: " + str(acc))

f1 = f1_score(labels, preds, average='macro')
print("F1: " + str(f1))
recall = recall_score(labels, preds, average='macro')
print("Recall: " + str(recall))
precision = precision_score(labels, preds, average='macro')
print("Precision: " + str(precision))

weights = []
for i in range(32):
    kernel = cnn._modules['layer1']._modules['0'].weight.data[i][0].cpu().numpy()
    weights.append(kernel)

flat_weights = []
for weight in weights:
    flat_weight = [x for row in weight for x in row]
    flat_weights.append(flat_weight)

weights_df = pd.DataFrame(flat_weights)
weights_df.to_csv('/home/ubuntu/MachineLearningII/Weights/50imsize.csv')
```

```
print('Done')
```

Plotting Metrics

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df1 = pd.read_csv("C:/Users/sjcrum/Documents/Machine Learning II/Final Project/Github/Document-Image-Recognition/Code/Metrics/BatchSize.csv")
df2 = pd.read_csv("C:/Users/sjcrum/Documents/Machine Learning II/Final Project/Github/Document-Image-Recognition/Code/Metrics/LearningRate.csv")
df3 = pd.read_csv("C:/Users/sjcrum/Documents/Machine Learning II/Final Project/Github/Document-Image-Recognition/Code/Metrics/ImageSize.csv")
df4 = pd.read_csv("C:/Users/sjcrum/Documents/Machine Learning II/Final Project/Github/Document-Image-Recognition/Code/Metrics/HistEq.csv")
df5 = pd.read_csv("C:/Users/sjcrum/Documents/Machine Learning II/Final Project/Github/Document-Image-Recognition/Code/Metrics/Kernel.csv")


training_time = df3.iloc[:, -1]

df1 = df1.iloc[:3,:-1]
df2 = df2.iloc[:3,:-1]
#Getting nan rows below data
df3 = df3.iloc[:3, :-1]


batch_size = pd.melt(df1, id_vars="Batch Size", var_name="Metric", value_name="Metric Score")
print(batch_size)

lr = pd.melt(df2, id_vars="Learning Rate", var_name="Metric", value_name="Metric Score")
print(lr)

im_size = pd.melt(df3, id_vars="Image Size", var_name="Metric", value_name="Metric Score")
print(im_size)

hist_eq = pd.melt(df4, id_vars="Preprocess", var_name="Metric", value_name="Metric Score")
print(hist_eq)

kernel = pd.melt(df5, id_vars="Kernels", var_name="Metric", value_name="Metric Score")
print(kernel)

sns.set(rc={'figure.facecolor':'white'})
sns.factorplot(x='Batch Size', y='Metric Score', hue='Metric', data=batch_size, kind='bar')
plt.title("Testing Metrics by Batch Size")
plt.show()
```

```
sns.set(rc={'figure.facecolor':'white'})
sns.factorplot(x='Learning Rate', y='Metric Score', hue='Metric', data=lr, kind='bar')
plt.title("Testing Metrics by Learning Rate")
plt.show()

sns.set(rc={'figure.facecolor':'white'})
sns.factorplot(x='Image Size', y='Metric Score', hue='Metric', data=im_size, kind='bar')
plt.title("Testing Metrics by Learning Rate")
plt.show()

sns.set(rc={'figure.facecolor':'white'})
sns.factorplot(x='Preprocess', y='Metric Score', hue='Metric', data=hist_eq, kind='bar')
plt.title("Testing Metrics with Preprocessing")
plt.show()

sns.set(rc={'figure.facecolor':'white'})
sns.factorplot(x='Kernels', y='Metric Score', hue='Metric', data=kernel, kind='bar')
plt.title("Testing Metrics with Preprocessing")
plt.show()
```

Plotting Loss

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import os
os.chdir('C:/Users/sjcrum/Documents/Machine Learning II/Final Project/Github/Document-Image-
Recognition/Code/Losses/')

loss00001 = pd.read_csv('ModelLosses-4LR.csv', header=None)
loss0001 = pd.read_csv('ModelLosses-3LR.csv', header=None)
loss001 = pd.read_csv('ModelLosses-50batch-2LR.csv', header=None)
loss01 = pd.read_csv('ModelLosses-1LR.csv', header=None)

def make_df(df, lr):
    x = range(len(df.columns))
    df_T = df.transpose()
    df_T['1'] = x
    df_T.columns = ['{}'.format(lr), 'Iteration']
    return df_T

lr01 = make_df(loss01, '1e-2')
lr001 = make_df(loss001, '1e-3')
lr0001 = make_df(loss0001, '1e-4')
lr00001 = make_df(loss00001, '1e-5')

fig, ax = plt.subplots()
```

```
sns.set_style("darkgrid")
ax.plot(lr01['Iteration'], lr01['1e-2'], linewidth = 0.1)
ax.plot(lr001['Iteration'], lr001['1e-3'], linewidth = 0.1)
ax.plot(lr0001['Iteration'], lr0001['1e-4'], linewidth = 0.1)
ax.plot(lr00001['Iteration'], lr00001['1e-5'], linewidth = 0.1)

leg = ax.legend()
for line in leg.get_lines():
    line.set_linewidth(4.0)

plt.ylim(0,4)
plt.title('Loss by Learning Rate')
plt.ylabel('Loss')
plt.xlabel('Iteration')
plt.show()

smallkernel = pd.read_csv('ModelLosses-All3Kernel.csv', header=None)
bigkernel = pd.read_csv('ModelLosses-BigKernels.csv', header=None)

smallk = make_df(smallkernel, 'Small Kernels')
bigk = make_df(bigkernel, 'Large Kernels')

fig, ax = plt.subplots()

sns.set_style("darkgrid")
ax.plot(smallk['Iteration'], smallk['Small Kernels'], linewidth = 0.1)
ax.plot(bigk['Iteration'], bigk['Large Kernels'], linewidth = 0.1)


leg = ax.legend()
for line in leg.get_lines():
    line.set_linewidth(4.0)

plt.ylim(0,4)
plt.title('Loss by Kernel Size')
plt.ylabel('Loss')
plt.xlabel('Iteration')
plt.show()

batch100 = pd.read_csv('ModelLosses-100batch.csv', header=None)
batch50 = pd.read_csv('ModelLosses-50batch-2LR.csv', header=None)
batch10 = pd.read_csv('ModelLosses-10batch.csv', header=None)

b100 = make_df(batch100, '100 Batch Size')
b50 = make_df(batch50, '50 Batch Size')
b10 = make_df(batch10, '10 Batch Size')

fig, ax = plt.subplots()

sns.set_style("darkgrid")
```

```
ax.plot(b100['Iteration'], b100['100 Batch Size'], linewidth = 0.1)
ax.plot(b50['Iteration'], b50['50 Batch Size'], linewidth = 0.1)
ax.plot(b10['Iteration'], b10['10 Batch Size'], linewidth = 0.1)

leg = ax.legend()
for line in leg.get_lines():
   line.set_linewidth(4.0)

plt.ylim(0,4)
plt.xlim([0,5000])
plt.title('Loss: 100 Batch Size')
plt.ylabel('Loss')
plt.xlabel('Iteration')
plt.show()
```

Plotting Confusion Matrix

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

cm = [[432 , 43 , 18 ,  8 , 6 , 18 , 7 ,  6 , 15 ,  3 , 16 , 24 , 16 , 28 , 33 , 77],
 [ 10 ,  401 , 18 , 19 , 15 , 32  , 2  , 6 , 27  , 3 , 49 , 96 , 16 , 24 , 19 , 13],
 [  7  , 4 ,705 ,  0 ,  5  , 0  , 4  , 1 ,  3 ,  0 ,  6 ,  2 ,  2 ,  3 ,  3 ,  5],
 [  4 ,  5  , 2, 619,  41  , 1  , 1  , 1 , 29 ,  0,  14 , 14 ,  4 , 14  , 1  , 0],
 [  2  , 7 ,  5 ,  4, 390 ,  0 ,  3  , 0  ,16 , 11 ,  1 ,  1 ,  7 ,  2 ,  1 ,  1],
 [  1 , 51 , 22 ,  9 , 13 ,291 , 35 ,  9 , 30 , 12 , 37 , 32, 104 , 25  ,65 , 14],
 [  0 ,  4  , 9 ,  0 , 13 , 10 ,613  , 1 , 17 , 31 ,  2 ,  4  ,28 ,  7 , 10 ,  1],
 [  5 , 74 ,  6 ,  6 ,  7 , 24 ,  4 ,482 ,  5 ,  2 , 50 , 40 ,  7 , 15 , 17  , 6],
 [  0  , 1  ,10  , 2 , 17 ,  3  , 0 ,  0 ,688 ,  1  , 2 ,  7 , 17  , 1 ,  1,   0],
 [  3  , 1 , 13 ,  0 , 46 , 12,  36 ,  1 ,  7, 570 ,  4 ,  8 , 28 ,  2 , 18  , 1],
 [  0 , 34  , 9  , 6 , 11 , 19  , 2 ,  8 , 36 ,  6 ,482 , 61 , 40  ,13 , 17 ,  6],
 [  8 , 44 , 11,   3 , 19 , 12 ,  1 ,  3,  12 , 10 , 48, 548 , 15 ,  4  , 8  , 4],
 [  7  , 9  , 7 ,  2  ,25 , 29 ,  9  , 0 , 36 , 14 , 14  , 9 ,514 , 13 , 33 , 29],
 [  6 , 64 , 17 ,  9 , 15 , 19 ,  4,  12 , 34 , 13 , 15 , 44 , 43 ,407 , 26 , 21],
 [  3  , 9 ,  4 ,  0  , 1 , 10 ,  4 ,  1 ,  9 , 11 ,  6 ,  4 , 18 ,  5 ,654,  11],
 [ 14,  19 , 13 ,  3  , 6 , 29  , 5 ,  3  , 8 ,  4 , 23 , 20 , 26 , 16 , 30, 531]]

classes = {'letter': '0', 'form': '1', 'email': '2', 'handwritten': '3', 'advertisement': '4', 'scientific report': '5',
        'scientific publication': '6', 'specification': '7', 'file folder': '8', 'news article': '9', 'budget': '10',
        'invoice': '11', 'presentation': '12', 'questionnaire': '13', 'resume': '14', 'memo': '15'}


df_cm = pd.DataFrame(cm, index = classes.keys(), columns = classes.keys())
plt.figure(figsize = (12,9))
sns.heatmap(df_cm, annot=True, fmt='g')
plt.title("Final Model Confusion Matrix")
```

```
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

Plotting Kernels

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

#weights_df = pd.read_csv("C:/Users/sjcrum/Documents/Machine Learning II/Final
Project/Weights/model1kerneldf.csv")
#weights_df = pd.read_csv("C:/Users/sjcrum/Documents/Machine Learning II/Final
Project/Weights/3kernelmode1kerneldf.csv")
weights_df = pd.read_csv("C:/Users/sjcrum/Documents/Machine Learning II/Final Project/Weights/11-9-
7-5-3kernelmodel.csv")

print(weights_df.head())

weights = weights_df.iloc[:,1:]
kernels = []
for i in range(len(weights)):
    kernel = np.reshape(weights.iloc[i,:], (11,11))
    #kernel = np.reshape(weights.iloc[i,:], (7,7))
    #kernel = np.reshape(weights.iloc[i,:], (3,3))
    kernels.append(kernel)
    if i == 0:
        plt.imshow(kernel, cmap = 'gray')
        plt.show()


fig, ax = plt.subplots(4,8, figsize = (30,20))

index = 0
for i in range(4):
    for j in range(8):
        ax[i,j].imshow(kernels[index])
        index += 1

plt.show()
```

Plotting Feature Maps

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import cv2

import os
os.chdir('C:/Users/sjcrum/Documents/Machine Learning II/Final Project/Weights')
```

```
featmap_df1 = pd.read_csv("QuestFMLayer1.csv")
featmap_df2 = pd.read_csv("QuestFMLayer2.csv")
featmap_df3 = pd.read_csv("QuestFMLayer3.csv")
featmap_df4 = pd.read_csv("QuestFMLayer4.csv")
featmap_df5 = pd.read_csv("QuestFMLayer5.csv")

original_image = cv2.imread("0012210663.tif",0)
orig = cv2.resize(original_image, (224,224))

map1 = np.reshape(np.array(featmap_df1.iloc[:, 1]), (112,112))
map2 = np.reshape(np.array(featmap_df2.iloc[:, 1]), (56,56))
map3 = np.reshape(np.array(featmap_df3.iloc[:, 1]), (28,28))
map4 = np.reshape(np.array(featmap_df4.iloc[:, 1]), (14,14))
map5 = np.reshape(np.array(featmap_df5.iloc[:, 1]), (7,7))

plt.imshow(orig, cmap= 'gray')
plt.title('Original Image, 224x224')
plt.show()
plt.imshow(map1, cmap = 'gray')
plt.title('Conv-Pool Layer 1, 112x112')
plt.show()
plt.imshow(map2, cmap = 'gray')
plt.title('Conv-Pool Layer 2, 56x56')
plt.show()
plt.imshow(map3, cmap = 'gray')
plt.title('Conv-Pool Layer 3, 28x28')
plt.show()
plt.imshow(map4, cmap = 'gray')
plt.title('Conv-Pool Layer 4, 14x14')
plt.show()
plt.imshow(map5, cmap = 'gray')
plt.title('Conv-Pool Layer 5, 7x7')
plt.show()
```