A Novel Supervised Learning Based Approach for block page Detection

Arian Akhavan Niaki
College of Information and Computer Sciences
University of Massachusetts
140 Governors Dr., Amherst, MA 01003
arian@cs.umass.edu

Abstract

Current supervised learning approaches in machine learning literature have mostly concentrated on a single aspect data such as images or text and develop models for text and image classification. However, combining these two types of data could improve performance in the case of missing data in either images or text. In this paper, we crawl the Alexa top 500 web pages and create a dataset of web page screenshots and HTML documents. We then design and implement a deep convolutional neural network (CNN) which takes both the image and text data in order to perform block page detection. We compare our results to an image classification CNN, a text classification CNN and heuristic based block page detection. We show that our model achieves a macro-averaged F1 score of 0.981 which is an improvement to the text's 0.9456 but slightly worse than the image CNN's score of 0.985.

1 Introduction

Extensive work has been done in supervised and unsupervised learning on both image and text classification. However, to the best of our knowledge, there does not exist an approach that applies supervised learning on a dataset containing both images and textual data. This can be advantageous in the case where we either have missing data in the images or text. We see this case occurring in the field of Internet censorship and Tor discrimination where crawlers fail to fetch both the screenshot and the HTML document of web pages.

Internet censorship often manifests in the form of *block pages* returned to users accessing content. These block pages vary in appearance and content depending on the country. Moreover, previous work has shown that the users of anonymity systems such as Tor, receive discrimination in multiple forms such as CAPTCHAs, interaction based discrimination, and block pages. [14]

Researchers have made attempts to characterize Internet censorship by detecting block pages automatically [6]. Nevertheless, their efforts have largely been heuristic based and have shortcomings in distinguishing server errors from block pages as we will reveal in this paper.

In this paper, In order to detect and classify block pages and accurately measure discrimination against users, we use a Selenium-based crawler [13] to fetch the Alexa top 500 websites from 50 different Tor exit relays. We then design and implement a deep neural network which leverages both images and the HTML text of the web pages in order to perform multiclass classification by detecting block pages, server errors, connection errors, and legitimate web pages. In order to measure the performance and accuracy of our proposed model, we perform three experiments comparing the precision, recall, F1 score and macro-averaged F1 score of our proposed model with a state-of-the-art image classification deep neural net, a text classification deep neural net separately and heuristic based block page detection. We show that our proposed model can achieve better performance compared to the text CNN but is slightly outperformed by the image CNN.

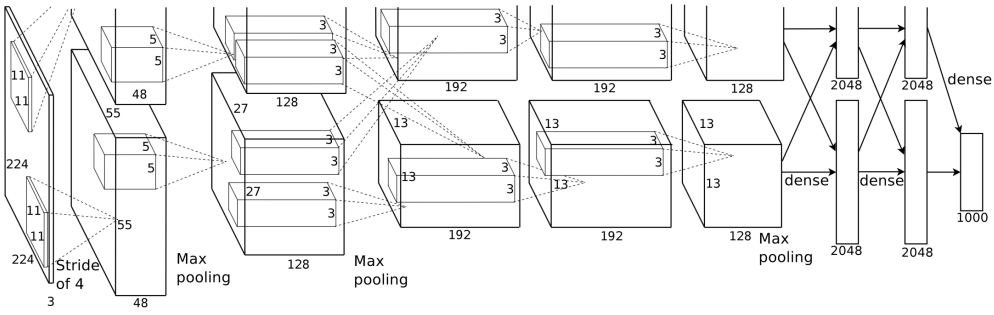


Figure 1: Imagenet architecture [9]

2 Related Work

As there is no related work on block page detection using machine learning techniques, in this section, we describe related work in image and text classification and prior work done in block page detection using heuristic techniques.

2.1 Image Classification

One of the foundational papers in image classification is ImageNet [9]. The structure of our image classification deep neural network is inspired by this work. However, our proposed network also receives a text document as input while this work only concentrates on images. The authors trained a large, deep convolutional neural network to classify 1.2 million high-resolution images into 10 different classes. In contrast, our network is designed to classify into 4 different classes. They implement their model using parallel GPUs and show that the best number of layers for their task is five convolutional layers followed by max-pooling layers and three fully-connected layers. Another advancements in this paper is applying Rectified Linear Units (ReLUs) as the linear activation unit in their neural network which makes training six times faster than the alternative $f(x) = tanh(x)$. Their model's architecture is shown in Figure 1. They solve overfitting by using (1) data augmentation which artificially enlarges the dataset using transformations on images and (2) applying dropout where they set the output of each hidden neuron to zero with 0.5 probability. These dropped neurons do not contribute to either forward pass or backpropagation allowing the model to learn more robust features. One weakness of this paper is not declaring the decision behind the parameters of the neural network architecture. They also do not provide numerical results for the effects of different dropout probabilities. Altogether, this deep neural network would not work with our dataset. Our work complements this by adding a neural network for text classification as well and we hypothesize that it will perform better in the case where image data is missing but we have the HTML of the web page.

The authors of [11] have introduced an extension to convolutional neural networks (CNN) called Tiled CNNs. Rather than all weights being equal, these networks only require hidden units positioned k steps of each other to have equal weights which will lead to learning more complex range of invariances. They also use an unsupervised learning algorithm that learns features from unlabeled image patches called Topographic ICA (TICA). They show that TICA pre-training for Tiled CNNs performs well on object recognition. We explore how this work can be integrated into our proposed model in section 7.

Authors in [10] show that one can build high-level, class-specific feature detectors from unlabeled data in an inexpensive way. This is an extension to autoencoders which are used for learning low-level features. They use a large dataset of 200×200 images and feed it into a deep autoencoder with pooling and local contrast normalization. They employ local receptive fields, local contrast normalization, and local l_2 pooling to scale for larger images and learn invariant features. Although our dataset contains only labeled data, we can use unlabeled data to learn high-level features and then train our network using labeled data. We will discuss how we can integrate this work with ours in section 7.

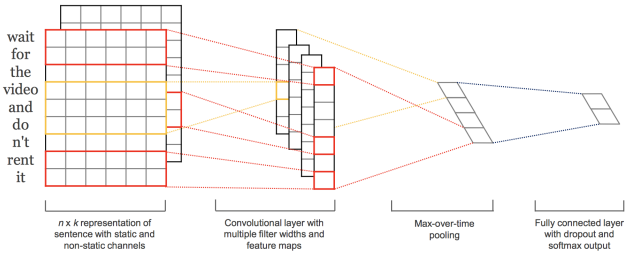


Figure 2: Architecture of text classification CNN [7]

2.2 Text Classification

There exists a myriad of publications on text classification and they all aim to assign free-text documents to predefined categories. In [7], authors demonstrate that applying a CNN on top of a pre-trained word vector performs well on sentence-level classification tasks. They train a CNN with a single convolutional layer on top of word vectors that are pre-trained on 100 billion words of Google News[1]. The results suggest that pre-trained vectors are universal feature extractors. Their model's architecture is shown in Figure 2. The structure of our text CNN is inspired by this work. They also apply regularization by using dropout on the output of the max pooling layer. One of the strong points in this work is performing experiments using several variants of the model. For example, comparing the result of a CNN with static pre-trained vectors, and CNN with randomly initialized word vectors on multiple datasets. However, this model also fails to work on our dataset because we have images in addition to HTML documents. Additionally, this model will not perform well on text documents which contain words not available in the embedding matrix. We complement this model by adding a neural network for image classification which will help in the cases where the text is missing in our data.

Character-level CNNs for text classification is studied in [17]. They consider text as a raw signal at the character level and apply one-dimensional CNNs to it. An alphabet of size m is constructed and each character is encoded to a one-hot vector with a fixed length l_0. Characters not present in the alphabet are set to all-zero vectors. One strong aspect of this paper is applying data augmentation using a thesaurus and replacing words by their synonyms to control generalization error. They compare their results to word-based CNN approach described in [7] as well as traditional text classification models such as bag-of-means on word embeddings. They conclude that the model's performance depends on various factors such as dataset size and choice of the alphabet and show that character-level CNNs have the potential of performing text classification without needing words.

Finally, a work done by [3] performs a survey on neural network models from the perspective of natural language processing. Specifically related to our work is their discussion on convolutional neural networks and the use of word embeddings for representing each feature as a vector in a low dimensional space. According to [3], convolutional networks with pooling layers perform well in classification tasks. The strong point of this work that it does a comprehensive study on different aspects of neural network models for natural language processing. However, it does not provide sufficient details about each approach. We have designed our text classification neural network while having in mind that CNNs are to be integrated into larger networks in order to be effective [4].

2.3 Block page detection

The Authors in [6] study several metrics for block page detection such length difference, cosine and Document Object Model(DOM) similarity. They conclude that a length difference of more than 30.19% between a censored page and its uncensored counterpart is the most accurate heuristic, relying on the idea that block pages are usually shorter than the original page. However, this approach fails to distinguish between server errors, block pages, and connection errors. Whereas, our model is designed to distinguish them accurately. We did not find research on block page detection in the field of machine learning, however, machine learning techniques have been used for detecting website defacements by [2]. The structure of their deep neural network was inspired by work from [9, 11, 10].

[1] https://code.google.com/p/word2vec/

3 Methodology

In this section, we describe the architecture of the components in our proposed deep convolutional neural network for classifying web pages into 4 classes. Our model consists of 2 CNNs, one for images and one for text. The novelty of our approach is that we consider both data cases of image and text at the same time. This will prove advantageous when having missing data, either not having the HTML document or not having the screenshot. We have such data cases in our dataset where an error occurred and the HTML document is blank but we have a screenshot of the error page. Our proposed neural network is implemented in Python using the tensorflow library [1]. Another case where our proposed model will be superior to previous text classification models is that when web pages contain words that are not in the pre-trained word vector matrix. In these scenarios, our model uses the screenshots to make the prediction whereas a simple text classification CNN would fail.

3.1 Image Neural Network

The design of our image classification convolutional neural network is inspired by [9] and consists of 5 layers. More specifically, the first three layers are convolutional layers and the final two are fully-connected layers. Since we have 4 categories, the output of the last fully-connected layer is given as input to a 4-way softmax function shown in equation 2 which predicts the class labels by producing a class probability $f^c(x, \theta) \in [0, 1]$. Our proposed model uses softmax cross-entropy described in equation 1 as the cost function.

$$\begin{aligned} \theta_* &= \underset{\theta}{\operatorname{argmin}} \sum_{n=1}^{N} L_{ce}(y_n^c, f^c(x_n, \theta)) \\ L_{ce}(y, p) &= -y \times log(p) - (1 - y) \times log(1 - p) \end{aligned} \tag{1}$$

For optimizing the cost function we employ the Adam optimizer. According to empirical results, it achieves better results in comparison to other stochastic optimization methods [8]. Our first convolutional layer filters the $160 \times 160 \times 3$ input image with 32 kernels of size $16 \times 16 \times 3$ with a stride of 4 pixels which means we slide the convolutional filter over the input image by 4 pixels at a time, this will reduce the size of the CNN's output. Further, we want the output of the CNN to have the same size as the input, therefore we apply zero padding where 0 values are added to the edge of the input to preserve the input size. ReLU is chosen as the activation function $f(x) = max(0, x)$ for the outputs of the CNN neurons because it is shown to be faster than the alternative $tanh$ activation function [9]. We then apply a 2×2 max-pooling where we select the largest value from each 2×2 window. The second and third layers, filter the output of their previous layer using 64 kernels of size $8 \times 8 \times 3$ and 128 kernels of size $4 \times 4 \times 3$ with a stride of 4 pixels, 2×2 max pooling, and ReLU activation functions. In order to avoid overfitting, we apply dropout in our network as depicted in Figure 3. It has been shown in [16] that using dropout in convolutional neural networks improves performance. Prior work has also shown that using dropout on the last hidden layer of a fully-connected layer reduces the error rate [5]. To be more concise, we set the output of each hidden neuron to zero with probability 0.5. By doing this, we assure that the network is not always using the same neurons and is not getting fitted to the training data. Finally, we flatten the output of the third convolutional layer in order to make it compatible as input to the fully-connected layers with 128 neurons. If we want to run the network separately as we do in section 5, the output of the last fully-connected layer will go through a 4-way softmax function. The network's architecture is shown in the top part of Figure 3. The weights in the CNN and fully-connected layer are generated from a truncated normal distribution with $stddev = 0.05$ and the biases are set to 0.05. The full network structure is available in the appendix.

$$P(y = c|x; \theta) = \frac{\exp(x^T \theta_c)}{\sum_{n=1}^{N} \exp(x^T \theta_n))}, \tag{2}$$

3.2 Text Neural Network

The design of our text classification CNN is inspired by [7]. Similar to our image classification CNN, it consists of 5 layers, with the first three being convolutional layers and the rest being fully-connected layers. We need to apply some pre-processing on our HTML data in order to extract features before we can use them as input. First, we separate the section in which most of the content resides, which is the *body* section, then we only consider the first 200 words of the *body*. Thus we will have the following vector for each HTML web page: $doc = [w_1, w_2, ...w_{200}]$. In order to

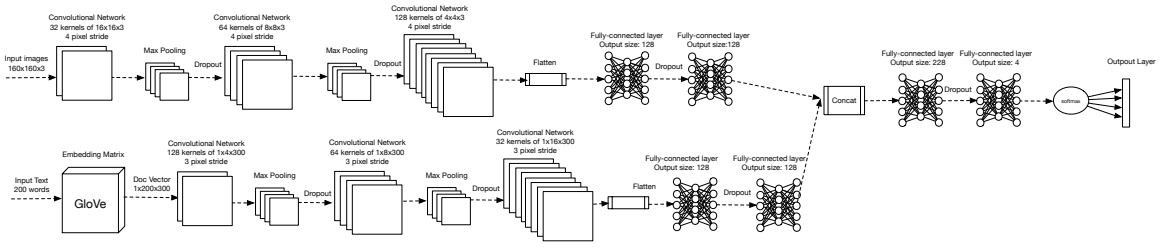


Figure 3: The architecture of our proposed deep convolutional neural network

convert these words into features, we use pre-trained word vectors from GloVe's embedding matrix which contains 300-dimensional vectors for 1.9M words. GloVe is a log-bilinear regression model of word representations [12]. We fetch the vector v_i for the first 200 words in a HTML document from GloVe's embedding matrix and build $doc = [[v_1], [v_2], ..[v_{200}]]$. In the case that the HTML document contains less than 200 words, we pad the final doc with 200-dimensional zero vectors. Therefore, each HTML document will have the shape $1 \times 200 \times 300$ with 300 being the number of channels. The structure of the text CNN is similar to the image classification one with the same cost function and optimization algorithm. The first convolutional layer applies a filter to the input using 128 kernels of size $1 \times 4 \times 300$ with a stride of 3 pixels. After applying the ReLU function we select the max value within a 1×2 window. The mathematical description is as follows:
Every document is represented by the concatenation of the word vectors $doc = [v_1 \oplus v_2 \oplus ...v_{200}]$ and by applying the convolution in windows of size 2 and the ReLU activation function, we will have $c_i = ReLU(w \cdot v_{i:i+2} + b)$ so max pooling will compute the maximum between $c = [c_1, c_2, ...c_{199}]$ which is shown as $\hat{c} = max(c)$. The second convolutional layer applies a filter using 64 kernels of $1 \times 8 \times 300$ with the same max pooling scheme as the previous layer and the third convolutional layer applies a filter with 32 kernels of $1 \times 16 \times 300$. The intuition behind increasing the layer sizes is that we first want look at the phrases and then move our way up to the whole sentence. ReLU is used as the activation function of all layers and we apply a dropout probability of 0.5. The same flattening process occurs on the output of the third convolutional layer and we go through two fully-connected layers with dropout probability of 0.5. As we will show in section 5 the output of the last fully-connected layer will go through a 4-way softmax function if we want to run the network separately. The network's architecture is shown in the bottom part of Figure 3. The complete network structure is available in the appendix.

3.3 Proposed Neural Network

Our network as depicted in Figure 3 first receives the HTML document and the image of the web page. It then feeds the input into the two separate CNNs in parallel. If the HTML document is blank we feed a $1 \times 200 \times 300$ zero vector to the text classification CNN and the associating screenshot to the image classification CNN. There exist cases where we have the legitimate HTML document of the web page but the screenshot is a white picture, we proceed as before and feed each of the inputs to the appropriate CNN. Our model concatenates the output of each of the text and image CNNs and feeds them as an input to another fully-connected layer where a dropout of 0.5 is applied. Finally, we go through one last fully-connected layer. A 4-way softmax function is applied to the output of the last fully-connected layer to predict the class label.

4 Datasets

We use an automated Selenium-based web crawler [13] written in Python for collecting screenshots and the HTML documents of the Alexa[2] top 500 web pages from a control host and 50 Tor [15] exit relays located in 23 countries. Our control host does not connect to Tor exit relays and is located in an uncensored network in the US, the uncensored web pages will be used in the heuristic based block page detection. Our crawls took place in November 2017 and the dataset consists of 21246 screenshots and their associated HTML documents. Glancing at the screenshots we can clearly see 4 types of pages. (1) The Tor user can access the website like a non-Tor user which we see in Figure 4a, we label these pages as *ok*. (2) the Tor user reaches a server error page which we label with *servererror* an example of these pages is shown in Figure 4d. (3) the Tor user gets a connection failed or timeout page, we label these as *connectionerror*. A sample connection error

[2]https://www.alexa.com/topsites

(a) Ok web page (b) Block page (c) Connection error page (d) Server error page

Figure 4: Samples of our dataset

page is depicted in Figure 4c and finally (4) the Tor user gets a block page or a page containing some form of CAPTCHA which we label as *block* that can be seen in Figure 4b. Our images have variable-resolutions, therefore we rescale our images to 160×160. We do not crop the center of each image because many of the server error pages have content shown in the top part of the page. No pre-processing or data augmentation is executed on our data. The raw RGB values of the pixels are used as our image input. For the HTML documents, we take out the *body* section of the page and parse over the first 200 words and provide this as our text inputs. This is because of the fact that the text content in an HTML document is available in the body section. Our labeled dataset consists of 17783, 2175, 769 and 519, ok, block, connection error and server error pages.

5 Experiments

In order to evaluate the performance of our proposed model, we compare it with (1) a CNN for image classification as described in section 3.1 (2) a CNN for text classification as described in section 3.2 and, (3) prior work for block page detection (heuristic based length difference) using a set of performance metrics. The heuristic based block page detection is a threshold that marks any difference in size over 30.19% between a censored page and the uncensored version as blocked [6].

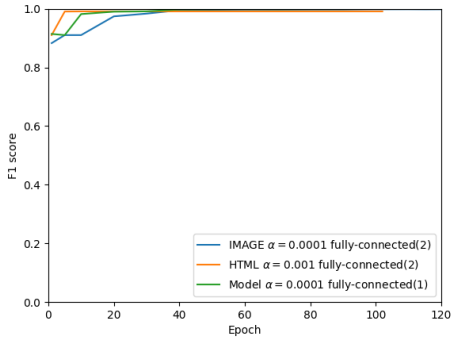
We conduct all the experiments on the previously mentioned dataset where we randomly take 80% of the data as training and set aside the remaining 20% for the test phase. We take out 25% of the training data for validation. Our hypothesis is that the our designed CNN should perform better than the CNNs for image and text classification. The reason being that there are cases where either the image or the text is missing for a data case and our proposed CNN would be more beneficial than a single CNN only focusing on text or image. Finally, previous techniques for block page detection were only heuristic based and had false positives, our method is designed to classify web pages into 4 distinct classes more accurately. In all experiments, the training batch size, validation batch size and test batch size are 607, 85 and 607 respectively. We also apply an early stopping technique to our experiments. If the validation loss value does not improve in 7 subsequent epochs we do not continue and report the performance metric values when running the test data on the model's best-learned parameters(weights).

5.1 Performance Metrics

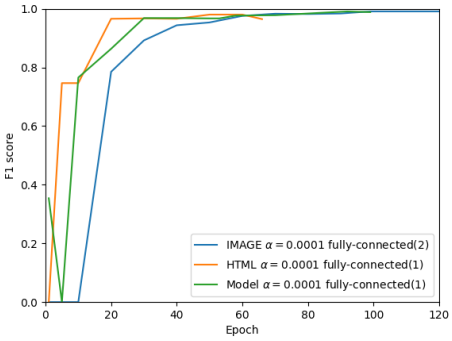
Accuracy is not a valid performance metric for our model, because our dataset is unbalanced. Meaning that the number of *ok* pages is much more than the other classes. Hence, we select precision, recall, F1 score and macro-average F1 score as our performance metrics. The formulas are also shown in equation 3 and 4. These performance metrics have been used to evaluate block page detection in prior work as well [6]. In this work, tp_c(True Positives) is the number of correctly predicted cases of class c, fp_c(False Positives) is the number of cases that are incorrectly predicted as class c and fn_c(False Negatives) is the number of cases belonging to class c that the classifier failed to classify. The macro-average F1 score is the average F1 score over all classes, this is important to us because our classes do not have the same number of instances and the macro-averaged F1 gives equal weight to all classes.

$$precision_c = \frac{tp_c}{tp_c + fp_c}, \quad recall_c = \frac{tp_c}{tp_c + fn_c}, \tag{3}$$

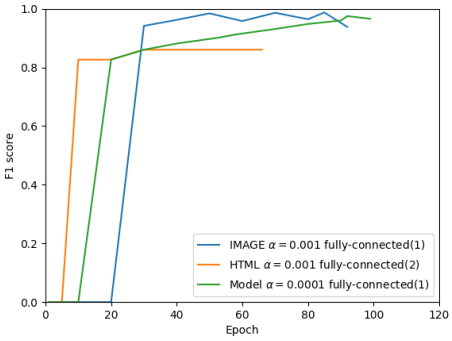
$$F1_c = 2 \cdot \frac{precision_c \cdot recall_c}{precision_c + recall_c} \quad F1_{macro-averaged} = \frac{\sum_{c=1}^{C} F1_c}{C} \tag{4}$$



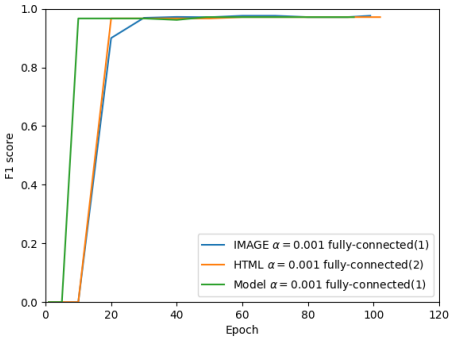
(a) Ok class



(b) Block page class



(c) Connection error class



(d) Server error class

Figure 5: F1 score vs number of epoch for each category

5.2 Hyper-parameters and Training

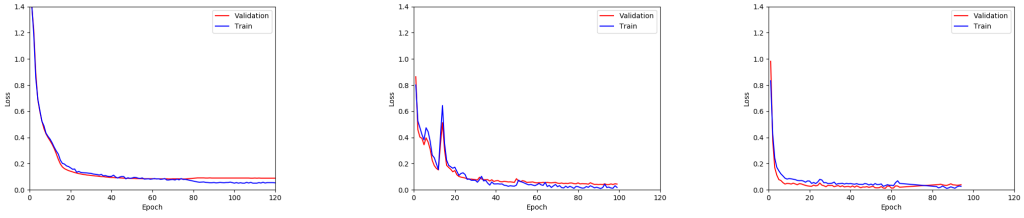
We perform a grid search for hyper-parameter tuning. More specifically, we take the learning rate (α) and the structure of the fully-connected layers in the networks as our hyper-parameters. For the learning rate, we consider 0.001 and 0.0001 and for the fully-connected layers we consider: the case where we described our model architecture in section 3.3 which we indicate with (1) for the rest of this paper and the case where we change the output size of the first fully-connected layer after the flatten process to 256 then we add another fully-connected layer with the output size of 128, finally we pass through another fully-connected layer with output size 4, we specify this case with (2) for the rest of this paper. The alternative architecture is available in the appendix. Therefore, for each experiment we will consider the effect of these hyper-parameters on the model's performance. The results are shown in table 1.

6 Results

After applying hyper-parameter tuning on our approach and the image and text classification CNNs we can compare their results on the same test set for evaluating the experiments in section 5. For detecting *ok* pages and *block pages*, the image CNN works best with $\alpha = 0.0001$ and the alternative fully-connected layer design (2), our proposed model performs best with $\alpha = 0.0001$ and the text CNN is optimized with $\alpha = 0.0001$ and the default architecture (1). The highest F1 score for each class is specified with a bold font in table 1. For evaluating our experiments, we train each of the models, namely the image CNN, text CNN and our proposed CNN with their optimized hyper-parameters and compare the F1 score of each of the 4 classes and the macro-averaged F1 score for all the classes. For the *ok* and *block page* classes, our proposed model receives a 0.998 and 0.99 F1 score, which is higher than the text CNN and is the same as the image CNN but with fewer epochs. The heuristic based method achieves a 0.80 and 0.47 F1 score for the *ok* and *block* classes, which is a clear indication of it being out-of-date. For the *server error page* class, all models perform the same but our proposed model has the least number of epochs by 35. Finally, for the *connection*

Model	Ok page			Block page			Server error page			Connection error page			Epoch
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	
Image (1) $\alpha = 0.001$	0.997	0.998	0.998	0.993	0.988	0.99	0.98	0.962	**0.971**	0.981	0.981	0.981	92
Image (2) $\alpha = 0.001$	0.996	0.998	0.997	0.986	0.979	0.982	0.971	0.962	0.966	0.993	0.981	**0.987**	85
Image (1) $\alpha = 0.0001$	0.99	0.99	0.99	0.988	0.97	0.979	0.953	0.962	0.957	0.993	0.981	0.987	91
Image (2) $\alpha = 0.0001$	0.998	0.998	**0.998**	0.993	0.988	**0.99**	0.935	0.962	0.948	0.987	0.981	0.984	115
HTML (1) $\alpha = 0.001$	0.981	0.999	0.99	0.995	0.935	0.964	0.98	0.962	0.971	0.991	0.759	0.86	59
HTML (2) $\alpha = 0.001$	0.982	1.0	0.99	0.997	0.938	0.96	0.98	0.96	**0.971**	0.99	0.759	**0.86**	95
HTML (1) $\alpha = 0.0001$	0.982	1.0	**0.99**	1.0	0.938	**0.968**	0.971	0.962	0.966	0.991	0.759	0.86	125
HTML (2) $\alpha = 0.0001$	0.981	0.999	0.99	0.997	0.935	0.965	0.971	0.962	0.966	0.991	0.759	0.86	105
Our model (1) $\alpha = 0.001$	0.995	0.999	0.997	0.995	0.983	0.99	0.98	0.962	**0.971**	0.98	0.95	0.965	57
Our model (2) $\alpha = 0.001$	0.997	0.998	0.998	0.995	0.965	0.0.98	0.962	0.971	0.967	0.939	0.981	0.959	98
Our model (1) $\alpha = 0.0001$	0.998	0.999	**0.998**	0.997	0.983	**0.99**	0.971	0.952	0.962	0.968	0.981	**0.974**	92
Our model (2) $\alpha = 0.0001$	0.997	0.999	0.998	0.997	0.933	0.964	0.971	0.962	0.966	0.829	0.987	0.90	67
Length difference $\alpha = 0.0001$	0.93	0.71	0.80	0.31	1.0	0.47	0	0	0	0	0	0	0

Table 1: Performance metrics in various scenarios



(a) Text CNN (b) Image CNN (c) Proposed Model

Figure 6: Training and validation loss curves for the best settings of our 3 experiments

error pages, the Image CNN seems to outperform the other two. The macro-average F1 score for the Image CNN, text CNN and our proposed model are 0.985, 0.9456 and 0.981 respectively. This shows that the Image CNN outperforms the two other models and our proposed model performs close to the image CNN and better than the text CNN. However, this might be because of the fact that our dataset does not have many missing images of web pages. We can not come to a complete conclusion before analyzing on a larger dataset. Figure 5 plots the F1 score of each class vs the number of epochs each model had to train in order to obtain the results. Since we have more instances for the *ok* class, we see that in Figure 5a the starting F1 score is close tos 0.85 which is relatively high. Whereas, for the other classes the F1 score starts from 0. In order to ensure that the training converges without overfitting, we have plotted the training and validation loss values of our 3 experiments for the image CNN, text CNN and our proposed model which can be seen in Figure 6. We see no signs of overfitting as the validation loss curve closely follows the training loss curve.

7 Discussion and Conclusions

In this work, we design and implement a novel deep convolutional neural network which works with text and images simultaneously. We apply our model to the task of block page detection. We collect a dataset of web page crawls consisting of 21246 screenshots and HTML documents of the Alexa top 500 web pages from 50 Tor exit relays. We then study and compare the performance of our proposed model on our dataset, with an image convolutional neural network that works only with images, a textual convolutional neural network which only works with text and previous work on block page detection using heuristics. Our hypothesis is that our designed model will work better in cases when there are missing data in either the text or the screenshots. We use the F1 score and the macro-average F1 score as a performance metric. Our results prove that the previous techniques on block page detection are no longer functional and our proposed model performs remarkably better. However, our model performs slightly worse than the image CNN but requires fewer epochs. In conclusion, a larger dataset is required to evaluate the true performance of our proposed model with sufficient missing data cases.

7.1 Future work

As a future work, our proposed model can be extended by using stacked autoencoders with unlabeled data in order to learn high-level features [10] before we start the training phase. Furthermore, we plan to investigate the effect of using data augmentation in order to avoid overfitting. Data augmentation is used to enlarge the dataset by transforming the images and using a thesaurus to replace words with their synonyms [10, 17]. We also plan to evaluate the model on larger datasets.

References

[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[2] Kevin Borgolte et al. "Meerkat: Detecting Website Defacements through Image-based Object Recognition". In: *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, 2015, pp. 595–610. ISBN: 978-1-931971-232. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/borgolte.

[3] Yoav Goldberg. "A Primer on Neural Network Models for Natural Language Processing." In: *J. Artif. Intell. Res.(JAIR)* 57 (2016), pp. 345–420.

[4] Yoav Goldberg. "Neural Network Methods for Natural Language Processing". In: *Synthesis Lectures on Human Language Technologies* 10.1 (2017), pp. 1–309.

[5] Geoffrey E. Hinton et al. "Improving neural networks by preventing co-adaptation of feature detectors". In: *CoRR* abs/1207.0580 (2012). arXiv: 1207.0580. URL: http://arxiv.org/abs/1207.0580.

[6] Ben Jones et al. "Automated Detection and Fingerprinting of Censorship Block Pages". In: *Proceedings of the 2014 Conference on Internet Measurement Conference*. IMC '14. Vancouver, BC, Canada: ACM, 2014, pp. 299–304. ISBN: 978-1-4503-3213-2. DOI: 10.1145/2663716.2663722. URL: http://doi.acm.org/10.1145/2663716.2663722.

[7] Yoon Kim. "Convolutional Neural Networks for Sentence Classification". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1746–1751. DOI: 10.3115/v1/D14-1181. URL: http://www.aclweb.org/anthology/D14-1181.

[8] Diederik Kingma et al. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[9] Alex Krizhevsky et al. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105. URL: http://dl.acm.org/citation.cfm?id=2999134.2999257.

[10] Quoc V. Le et al. "Building High-level Features Using Large Scale Unsupervised Learning". In: *Proceedings of the 29th International Coference on International Conference on Machine Learning*. ICML'12. Edinburgh, Scotland: Omnipress, 2012, pp. 507–514. ISBN: 978-1-4503-1285-1. URL: http://dl.acm.org/citation.cfm?id=3042573.3042641.

[11] Quoc V. Le et al. "Tiled Convolutional Neural Networks". In: *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 1*. NIPS'10. Vancouver, British Columbia, Canada: Curran Associates Inc., 2010, pp. 1279–1287. URL: http://dl.acm.org/citation.cfm?id=2997189.2997332.

[12] Jeffrey Pennington et al. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[13] Sagar Shivaji Salunke. *Selenium Webdriver in Python: Learn with Examples*. 1st. USA: CreateSpace Independent Publishing Platform, 2014. ISBN: 1497337364, 9781497337367.

[14] Rachee Singh et al. "Characterizing the Nature and Dynamics of Tor Exit Blocking". In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017, pp. 325–341. ISBN: 978-1-931971-40-9. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/singh.

[15] *Tor Project*. https://www.torproject.org.

[16] Haibing Wu et al. "Towards dropout training for convolutional neural networks". In: *Neural Networks* 71 (2015), pp. 1–10.

[17] Xiang Zhang et al. "Character-level Convolutional Networks for Text Classification". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 649–657. URL: http://papers.nips.cc/paper/5782-character-level-convolutional-networks-for-text-classification.pdf.

Appendix A Image CNN

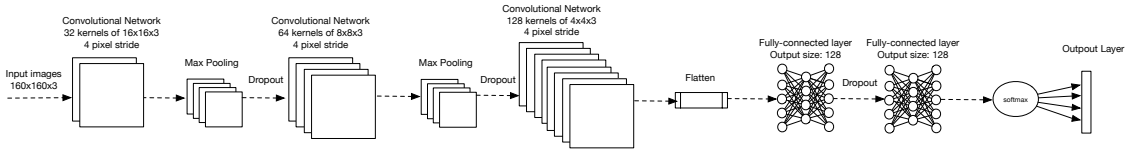


Figure 7: Image classification neural network

Appendix B Text CNN

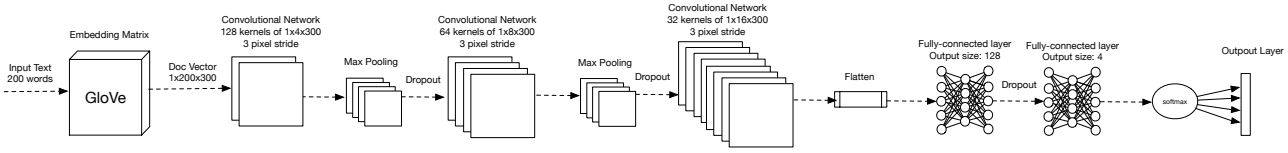


Figure 8: Text classification neural network

Appendix C Alternative Combined CNN (2)

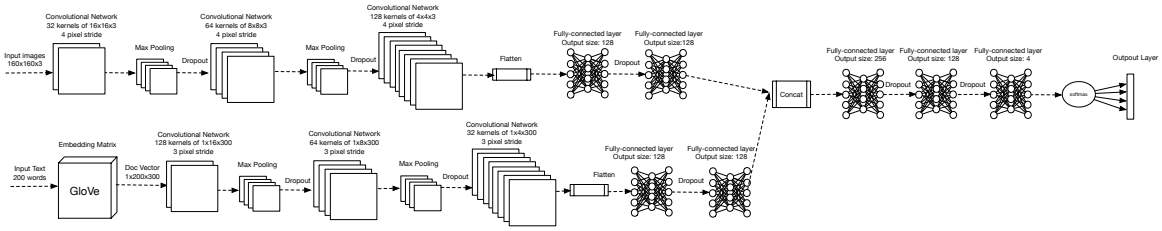


Figure 9: Alternative proposed model neural network