Character-Based Models for Adversarial Phone Number Extraction: Preventing Human Sex Trafficking

Nathanael Chambers **Timothy Forman** **Catherine Griswold**
Yogaish Khastgir **Kevin Lu** **Stephen Steckler**
Department of Computer Science
United States Naval Academy
nchamber@usna.edu

Abstract

Illicit activity on the Web often uses noisy text to obscure information between client and seller, such as the seller's phone number. This presents an interesting challenge to language understanding systems; how do we model adversarial noise in a text extraction system? This paper addresses the sex trafficking domain, and proposes some of the first neural network architectures to learn and extract phone numbers from noisy text. We create a new adversarial advertisement dataset, propose several RNN-based models to solve the problem, and most notably propose a *visual* character language model to interpret unseen unicode characters. We train a CRF jointly with a CNN to improve number recognition by 89% over just a CRF. Through data augmentation in this unique model, we present the first results on characters never seen in training.

1 Introduction

One reason people intentionally obscure textual content is to evade automatic extraction systems. There are good reasons for wanting to do this, privacy being at the forefront. However, illicit activity is another reason, and human sex trafficking is one of the most egregious uses. We draw inspiration from this domain, but extracting information from *adversarial* noisy text is a more general challenge for the NLP community. It is a language understanding task that humans can easily do, but which presents difficulty for automated methods. This paper presents the first deep learning models for adversarial phone number extraction, and releases new datasets for future experimentation.

An obscured example number is shown here:

(9I4) Too.46-callme-ÖÖ1/4

The true phone number is 914-246-0014, but this breaks even the most comprehensive rule-based extractors. It contains examples of visual substitution (I for 1 and unicode for 0), word substitution ("Too" for 2), and character confounders (separators '.', '-', '/' and other words). Any one challenge might be solvable in isolation, but they often combine together:

n1ne0one 7n1ne3 n1ne351

Rather than swapping letters for digits (I for 1), this example swaps digits for letters (1 for i) which are also part of a word swap ('nine' for 9). There are four '1' characters in the string, but only one of them maps to one of the two 1 digits in the number 901-793-9351. Beyond this, the most challenging noise occurs when unicode is injected, thus rendering finite character models ineffective since they've never seen these characters in training. This paper proposes to model all of this noise with several neural network architectures.

The domain of focus for our study is human sex trafficking, although our proposed models apply to any domain with obscured information (social media, for instance, often mixes unusual characters, confounding normal language models). This topic is important in terms of global need, but it also has attractive language properties for research. Since our datasets come from people who need to post contact information, they can't obscure the text *too much*, or nobody could call them. This results in an interesting cognitive challenge that humans can solve, but state-of-the-art extraction struggles.

The main contributions in this paper are (1) the first neural models for noisy phone number extraction, (2) a visual language model over images of characters, (3) a combined CRF with CNN input, (4) a data augmentation technique for training that helps recognize unseen unicode, and (5) state-of-the-art extraction results on new datasets.

2 Previous Work

A number of papers have looked into the sex trafficking domain. Some focus on classifying entire ads as trafficking or not (Alvari et al., 2016, 2017), while others build knowledge graphs of mentioned entities (Szekely et al., 2015) or focus on normalizing attributes like geolocations (Kapoor et al., 2017; Kejriwal and Szekely, 2017; Kejriwal et al., 2017). Most of these use phone numbers as features, and several found them to be among the most important input (Dubrawski et al., 2015; Nagpal et al., 2017; Li et al., 2018). In fact, phone numbers are used as gold truth to connect similar ads or link traffickers (Rabbany et al., 2018; Li et al., 2018). Phone numbers have also been shown to be some of the most stable links to entities (Costin et al., 2013), so are important for entity linking tasks. Almost all of these threads assume correct phone extraction and ignore the difficulty of ads with obscured numbers. Although sometimes unspecified, they all appear to use rule-based extractors.

Most relevant to this paper is TJBatchExtractor, a rule-based regular expression system (Dubrawski et al., 2015) which is still state-of-the-art for extraction, and is used by other work on trafficking ID (Nagpal et al., 2017). We employ TJBatchExtractor to identify the ads with obscured text from which it *fails* to extract a number. Our paper thus focuses on only the difficult ads with noisy phone numbers.

Most language models use words or characters as their base inputs. One of our contributions is a visual model of characters. We use an image database of 65k unicode characters developed by BBVA Next Security Lab[1] for phishing prevention. Most similar is Liu et al. (2017) who use CNNs for Asian-language classification. They aren't addressing noise like our paper, but rather the semantics inherent to their visual characters.

Finally, we employ data augmentation (Ding et al., 2016; Xu et al., 2016) during training of our visual character model. This is commonly used in the visual community (Salamon and Bello, 2017; Zhong et al., 2017) and we adopt their overall idea to randomly perturb our character images to learn a robust character recognizer.

[1]https://github.com/next-security-lab

3 Data and Attributes

3.1 Noisy and Obscured Data

We begin by highlighting the main methods people use for adversarial noise in written text. This is not an exhaustive list, but it covers the vast majority of cases observed in this paper's datasets.

1. **Digits as Lexemes**. The most basic approach to obscuring numbers is to substitute lexemes (words) for digits. These are often easy to identify, and regular expressions with a dictionary are usually sufficient for detection. Words might be capitalized (FOUR) or camel case (foUr), such as in the text, "**threeoh2FOUR070six22**".

2. **Homophones**. This method replaces digits with homophones or near-rhymes, thereby confusing dictionary approaches as in "**337 9twennyfo 06juan9**". Tokens "twenny" and "juan" share phonological similarities with the digit pronunciation. Regular expressions cannot capture these without complex phoneme modeling.

3. **Letters as Digits**. This method substitutes ASCII letters for their digit lookalikes (e.g., *6I5 093 93B6*). The 'I' and 'B' are representing 1 and 8 respectively. These substitutions can grow more complicated with things like '()' for 0 and what was popularized as leetspeak in the 1980's with 'E' for '3' and other such inversions.

4. **Visual Deception and Unicode**. This is a variant of 'Letters as Digits' above, but goes beyond ASCII substitution to use Unicode characters. Unicode presents a huge challenge to extraction as these rely entirely on visual similarities in the character images. Below are just *some* unicode options that resemble the ASCII character '8':

8 8 ȣ 8 8 𝟖 Ȣ Ȣ 8 8 𝟴 8

A rule-based approach would have to manually map all possible characters to their digits, an impossible task for 138k current unicode characters (with future room for 1mil). This would also fail on the larger problem of capturing visually ambiguous close-matches. For instance, an emoticon smiley face can be used for the ASCII letter 'o':

(4 😶 n e 2) 456 9412

We are the first to our knowledge to model visual noise with a language model architecture.

5. **Confounding Separators**. Another common noise tactic is to insert arbitrary characters as separators. For example: *–270**1tree&&822==31–*. The noise in this obscured text is meant to confuse a pattern matcher as to when a digit's substring begins and ends. Other difficult versions of this method uses digit characters themselves as the separators: **111 410 111 897 111 3245 111**

6. **Human Reasoning**. The most difficult class of obscured text is that which requires reasoning to solve. For instance, including arithmetic (3+1) or instructions to invert digits. This type is a small minority in obscured phone numbers, but they prove most challenging.

Some of these challenges have rule-based solutions in isolation, but combined together, they overlap and build on each other for an exponential number of noisy combinations. This paper addresses all of these challenges except for homophones and human reasoning. We leave phoneme modeling to future work, and reasoning requires a different approach than discriminative classifiers. The most significant challenge this paper addresses is that of the visual deceptions (letters as digits, unicode, and visual sim). We propose the first neural model for visual similarity detection with a unique visual model based on a CNN.

4 Corpora

4.1 Real-World Noisy Advertisements

Our initial corpus started from a 250k advertisement crawl of Backpage and Craigslist escort sections, shared with us by the Global Emancipation Network. The majority of these ads (180k) are one line with a standard phone number and no actual text. We filtered these out to focus on ads with written descriptions.

After removing one-liners, we ran the state-of-the-art extractor (Dubrawski et al., 2015) to identify all ads where the extractor failed to extract anything. This remaining subset contains ads that either don't have a phone number, or they contain an obscured number that fooled the rule-based extractor. Figure 1 shows one such explicit ad.

Undergraduate volunteers manually inspected the remaining ads, removed those without numbers, and identified minimal text spans that encompassed any obscured phone numbers. These annotations resulted in approximately 200 real-world obscured ads with their obscured text spans.

Ad for Phone 555-584-4630

Sexy Slim 555 Ready for fun let me 584 satisfy your 4630 every desire no disappointments..!! **IF YOUR NOT SERIOUS PLEASE DON'T CALL ME..!!Kik Me-*censored* ____****CAR PLAY ONLY****

Figure 1: An example advertisement from the escort section of Backpage. Phone and username changed for anonymity. This ad illustrates an obscured number with normal digits, but text is interspersed in between.

Desiring a larger test set for evaluation, we created an adversarial data collection environment for undergrads to try and "beat" the TJBatchExtractor. This small-scale collection resulted in about 200 more obscured phone examples.

Merging the crawl with these adversarial obscured numbers, we had 390 real-world examples. We split into 250 test numbers and 140 for development (dev). The dev set was used for model improvement and parameter tuning, and the test set only for final results. Two examples from the dev set are given here:

Gold Phone	Ad Text
3189481720	tree1ate_nein 48-one7 twenty
4177015067	4!7 70! fifty6svn

Due to the nature of this domain, training data is difficult to obtain so neural models are stymied. We instead chose to "fake" the training data, creating our own computer-based adversarial dataset. Though training data is artificial, all experiments use the above *real-world* data annotations.

4.2 Artificial Noisy Adversarial Data

A core research question is now whether artificial training data can train this real-world task. This section describes our approach.

The generation algorithm starts with a 10 digit number string (randomly selected[2]), and then transforms the string with a sequence of obfuscation operations. Space prevents a full description of this process and its details, but we will release the code upon publication. Example transformations are as follows:

1. Insert separator ASCII chars between digits.
2. Replace a digit with an ASCII lookalike.

[2]We used a US area code dictionary, and followed the constraint that the 4th digit must be [2-9] whereas the 5th to 10th digits are [0-9]. Numbers were then chosen randomly.

Artificial Obscured Phone Numbers

```
2 1tree\6-zero0###33\~15
778cinco7five688 PaRtyGiRL 6
*forejuan*for 55!826ate
5 1290 si&te4 ~~~~~~~~~~135
ate0 5 ***2 08–88 8nine
```

Figure 2: Examples from the artificial phone number training set.

3. Replace a digit with its English, Spanish, or homonym (2 to 'two')
4. Capitalize letters or replace with an ASCII lookalike (C to '(')
5. Replace two digits with its English word ('18' to 'eighteen')
6. Insert random English words as separators

These occur in sequence, each with random chance, so the original digit '2' might become 'too' which then becomes 'To0' after character conversion. The output of this process is arguably more difficult than many real-world examples. See Figure 2 for generated examples. We ultimately trained on 100k of these.

5 Models for Obscured Extraction

5.1 Baseline Models

We use two baselines: one from prior work and another with a basic RNN model.

5.1.1 Rule-Based Baseline

The state-of-the-art for phone number extraction is the TJBatchExtractor from Debrawski et al. (2015). This is a large set of regular expressions designed to capture phone numbers even with variation and noise, mostly focused on what we've named "Digits as Lexemes" and "Letters as Digits". Their previous results showed 99% extraction accuracy, however, we found that 72% of ads are one line with just unobscured digits, so their result masks a more challenging subset.

5.1.2 RNN Baseline

Our baseline neural architecture is a character-based bi-directional LSTM. Input is a 70 character span of obscured text, and each character is mapped to its embedding vector. The embeddings are randomly initialized and learned during training. Each embedding is fed into the biLSTM, and the final hidden state of the biLSTM is treated as the representation of the obscured text. The hidden state is then passed to 10 independent dense layers, one for each of the 10 digits in the phone number. A softmax is then used on the output of each dense layer to predict the digit in that position of the 10-digit phone number.

We also tested GRUs instead of LSTMs, but performance did not significantly change.

5.2 Obscured Models

5.2.1 RNN with Positional Attention

The RNN baseline transforms the input text to a single vector from the biLSTM, and then predicts the digits in the phone number from this vector. We found that the model quickly learns to predict the first digits and the last digits, but learning for the middle digits is hindered. This intuitively makes sense because the vector represents the entire text without directed guidance on identifying *where* in the text the digits exist. How does the final dense layer know where the 4th and 5th digits begin? The initial digit, in contrast, is easier to identify because it leads the string.

Our solution to this deficiency was to add *positional attention* to the LSTM. Instead of using its final LSTM state, the vector is a weighted sum of *all* hidden states. The weight vector $\boldsymbol{\alpha}$ is the learned positional attention. Formally, the ith digit in the 10 digit phone number is predicted by a dense layer over context vector input W_i:

$$W_i = \sum_{j=0}^{N} \alpha_{ij} * V_j \tag{1}$$

where N is the length of the LSTM, V_j is the jth LSTM hidden state, i is the ith digit in the phone, and $\boldsymbol{\alpha_i}$ is the ith digit's positional attention vector. This allows the network to learn which part of the text is relevant for each digit. The first digit in the number should learn a weight vector $\boldsymbol{\alpha_0}$ that weights the front of the LSTM more than the end, and vice versa for $\boldsymbol{\alpha_9}$. Figure 3 shows this model.

We experimented with individual attention (each digit i has its own learned $\boldsymbol{\alpha_i}$) and a single shared attention (all digits use the same learned $\boldsymbol{\alpha}$). We only report on individual attention since it outperformed shared attention.

We also tested multiple stacked LSTM layers. Stacking showed no further improvement.

5.2.2 RNN with Conditioned Prediction

One characteristic of our task is that each digit prediction is *mostly* independent from the previous digit. Unlike many domains in NLP, this is

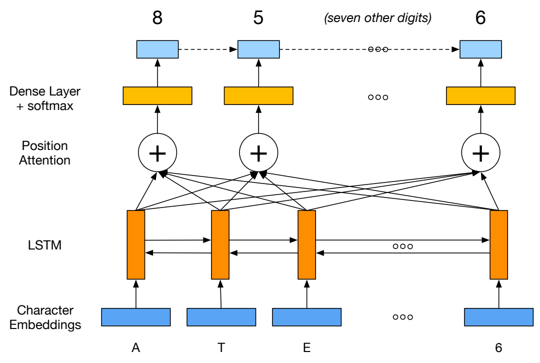

Figure 3: LSTM with position attention. Dotted lines included with conditioned prediction (Sec 5.2.2).

not a sequence modeling problem where knowing the previous digit semantically assists in guessing the next. For instance, a 5 is not more likely to be followed by a 4.[3] Despite position attention, the model still had difficulty distinguishing which portion of the context vector was relevant to a middle digit. It sometimes repeats an inner digit because the 4th and 5th positions were too nearby in the obscured text. Observe these 2 examples:

41093four 2830
4109threeefour tooo830

The seventh digit is a 2, but it starts five characters later in the second string. We observed repeated digit predictions like: 4109**44**830. It would predict the same number twice, and then skip over the next due to the shifting positions.

Our quick solution to avoiding repeats was to pass the predictions forward. We added a simple conditional dependency that feeds the softmax output of the previous digit to the current digit. The dotted lines in Figure 3 illustrate this new link. This removed many of our repeated digits, and also increased accuracy in other examples that weren't even repeated but just made mistakes.

5.2.3 Conditional Random Field Model

Given that providing the previous digit prediction showed slight improvements on the development set, we wanted to formalize the sequence predictions with proper transition probabilities. If a digit prediction leads to an unlikely next prediction (according to the model), then perhaps the previous digit should switch to its 2nd most likely in order to maximize the joint prediction.

[3]There are exceptions and phone numbers do have some constraints, such as having a limited set of 3 leading digits. However, the remaining 7 digits are mostly random in the US.

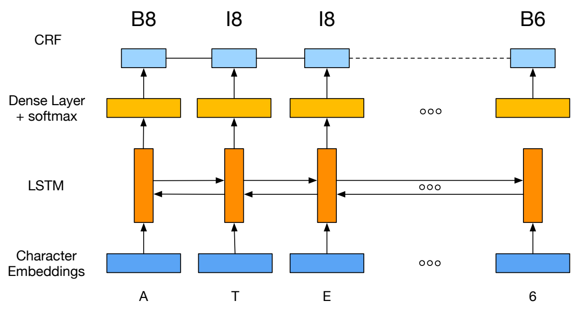

Figure 4: Neural architecture with a CRF top layer.

The other RNN problem is that input varies in length and noise. Some input is only about digits:

4treeTOO564ateSVN33

Others contain varying complex separators:

–4**tree**TOO sms 564ate+SVN+33

RNNs must learn to ignore separators in ways that don't confuse the subsequent dense layers. The network is remarkably adept at this, but we hypothesized that a better model should make a prediction *on each and every input character* rather than merging all into the same hidden state.

Conditional Random Fields (Lafferty et al., 2001) are a natural way of modeling the above. A CRF tags each character as it goes, and performs both training and inference, using viterbi search to find the most likely output prediction sequence. Figure 4 shows this model. We used the CRF implementation in Keras inspired by (Huang et al., 2015) to overlay a CRF on top of the RNN-based models (see also Ma and Hovy (2016)).

The output of a CRF is different since it must output a label for every character (rather than just 10 phone digits). We use the standard CRF labels to mark the beginning (B) and included (I) characters. This means that instead of a single label for each possible phone digit (e.g., 8), we now have two labels which represent a character that begins a digit (B8) and a character in the middle or end of a digit (I8). We additionally use an Other label 'O' to label the noisy separator characters that aren't part of any digit's substring. The following is an example:

B2	I2	I2	B4	B7	O	B6	I6	I6	B9	B9
T	O	O	4	7	-	s	i	x	9	9

The mapping from CRF labels (B2,I2,I2) to actual digits (2) is deterministic. Evaluation metrics for the previous RNNs also apply to the CRF output after it is mapped. However, training for the CRF is done entirely on the CRF label loss.

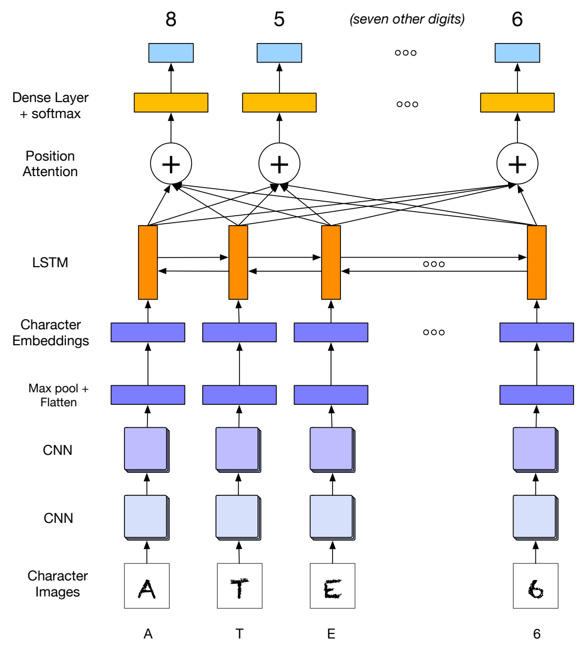

Figure 5: CNN architecture for visual image input to the LSTM model.

5.2.4 Visual Characters with CNNs

As with most NLP tasks, out of vocabulary (OOV) input is an issue. Our adversarial task is even more severe because visual substitutions are intentional, and often OOV as there are 138k current unicode options. If the character is unseen in training, only context can try to guess the digit. Below are examples of such replacement:

Digits	ASCII	Unicode
410	41o	41𝟘

Why are these easy for humans to decipher? It's purely due to visual similarity. In a "normal" NLP neural model, each character (or token) is mapped to an embedding, so unseen characters have no representation. We might use the default approach of mapping all unknowns to a shared 'UNK' embedding, but this loses the different visual characteristics of each character.

All of this motivates our new *Visual-Based Character RNN*. Our model does not learn a dictionary of character embeddings, but instead uses a series of CNN layers that transform 34x34 images of the characters. The transformations then feed into our above models. This is now a model that can interpret *unseen* (in training) characters.

Figure 5 shows the CNN combined with our positional attention RNN. We use two 3x3 convolution layers with 4 and 8 filters respectively. Each layer is followed by a relu layer and a batch normalization layer (not shown in the figure). The convolutions are followed by a max pooling layer and then flattened. A dense layer with softmax then reduces the flattened vector. We experimented with up to 3 convolution layers, up to 32 filters, and varied the size of the dense layer.

Visual input changes the model significantly. It is no longer learning an NLP-style character *embedding*, but rather learning CNN parameters to transform an image input into that embedding. Our first models ran into problems because they simply memorized each 34x34 image. Since all ASCII '3' characters map to the same flattened representation, the model memorizes it, and unicode variations fail no matter how similar. We thus introduced *data augmentation* during training. Each 34x34 input is 'jiggled' with random transformations: (1) translation of the image up/down or right/left, (2) darken/lighten the image, (3) stretch or widen, and (4) rotate up to 20 degrees. This provided different inputs for the same ASCII chars, so the CNN was encouraged to learn key visual features across all variants. Data augmentation led to our most significant improvements on unseen unicode character input.

6 Experiments

All models were trained on the 100k artificial obscured phone dataset (Section 4.2). 90k was used for training and 10k to determine convergence. The RNNs were set to $N = 70$ in length, and inputs were padded to that length. The rare input text longer than 70 is cropped. Embedding size *N=100* and LSTM internal dimensions *M=200* were chosen for all RNNs based on dev set performance. The CRFs performed best at *N=200*. We also applied dropout of 0.2 for the LSTMs and 0.5 CRF.

We report results with three metrics: digit accuracy, Levenshtein edit distance, and perfect accuracy. Digit accuracy is the simple alignment of predicted digit with gold digit (# correct / 10). If a predicted phone number is longer than 10 digits (CRFs are not bound to strictly 10 predictions), digit accuracy is computed only over the first 10 predicted digits.

Digit accuracy is flawed because a model might insert one extra digit, but still guess correct for the remainder. For example:

Gold: 4109342309
Guess: 41$\underline{1}$09342309

The CRF inserted an extra 1 digit, just one mistake, but digit accuracy is now a very low 0.2.

Model	Development Set			Test Set		
	Digit	Lev	Perfect	Digit	Lev	Perfect
TJBatch Rules	0.0	0.0	0.0	0.0	0.0	0.0
LSTM (5.1.2)	77.0	79.7	48.1	74.4	78.2	40.3
LSTM-2	77.5	79.4	49.7	74.8	77.6	40.6
LSTM +att (5.2.1)	78.5	80.5	48.6	76.6	78.9	43.5
LSTM +cond (5.2.2)	**79.7**	81.6	48.5	76.5	79.5	39.8
LSTM +att +cond	79.1	81.2	48.3	**77.2**	79.8	42.3
CRF with LSTM (5.2.3)	72.9	**84.0**	**58.1**	67.7	**83.4**	**48.2**

Table 1: Results on dev and test. Though flawed, digit accuracy is included for completeness. The +att and +cond options are not compatible with the CRF which does not need attention since it predicts at every input character.

We thus use the Levenshtein edit distance to better evaluate performance. Levenshtein's measure judges string similarity based on the minimum number of "edits" required to transform the prediction into the gold: $(1.0 - edits/10)$. In the above case, one deletion is required to make the strings the same, so the score is $(1 - 1/10) = 0.9$.

Finally, perfect accuracy is the number of perfect phone numbers (all 10 digits) that were correctly guessed, divided by the size of the test set.

Real-world Test: We report results only on the *real-world* test set from Section 4.1. The artificial data was solely used for training. We did not run models on the test set until the very end after choosing our best settings (on the dev set).

Real-world Challenge Test: To further illustrate the challenge of noisy text, we enhanced the real-world test set with *unicode injections*. Using a hand-created character lookup of visually similar unicode characters, we replaced 10% of the characters with randomly chosen unicode lookalikes not in the training data. This results in a very challenging test set to further benchmark the models.

Finally, all results in the next section are the average of 4 train/test runs of the same model.

7 Results

Table 1 contains results *without CNNs* for the baselines, RNNs, and CRF. The models listed are those that showed consistent improvement on development, and the test set columns were run only at the end for publication results. Adding position attention and conditional dependence each showed improvements of 1-2% Levenshtein. Stacking two LSTMs showed little gain. The CRF excelled with a 11% relative gain (on test) for perfect prediction over the best LSTM setup.

CNN Comparison (Perfect Acc)

	Test		Challenge	
	Lev	Perf	Lev	Perf
Best LSTM (no CNN)	**81.2**	**48.3**	72.9	22.1
CNN-LSTM	77.3	42.1	65.5	15.6
CNN-LSTM +aug	79.7	39.8	**75.2**	**27.3**
Best CRF (no CNN)	**84.0**	**58.1**	74.9	17.6
CNN-CRF	82.8	54.2	73.4	14.6
CNN-CRF +aug	83.3	56.1	**79.7**	**33.3**

Table 2: Results of the CNN models. Challenge has 10% unseen unicode injected. +aug used visual data augmentation during training.

For CNN results, Table 2 shows test set performance. Adding just the CNNs does not improve recognition, but in fact are slightly worse. However, more compelling is the challenge set with injected unicode confounders. Recall the importance of *data augmentation* during training so the models learn real visual features. These "+aug" results show why it is needed with a 89% relative improvement in perfect phone accuracy (from 17.6% to 33.3%). The non-CNN LSTM and CRF struggle at 17-22%. They simply cannnot represent unseen characters.

Our new CRF model (no CNN) outperforms the RNNs on the test set by 10% absolute. When comparing challenge test performance, the best CRF-CNN outperforms the best non-CNN LSTM by 11% absolute. To further illustrate the effect of unicode confounders, we varied how much we injected and graphed performance in Figure 3. The CNN models consistently outperform.

8 Full Ad Extraction

We wrapup with a pilot for full ad extraction. The models presented so far extract from one span of text (it assumes a phone number exists). This for-

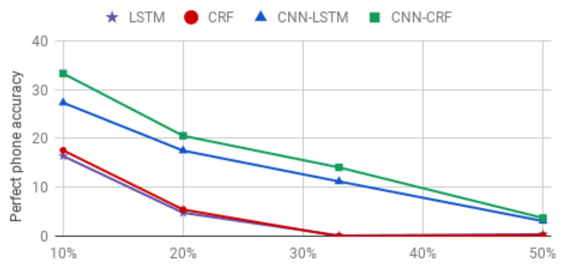

Table 3: Phone accuracy as a higher % of unicode substitutions are made for lookalike ASCII characters.

mulation is a well-defined task for research, but we also propose how one might apply these extractors to the more difficult task of full document extraction when the location of the phone number is unknown. We briefly describe initial tests.

The most straightforward way to extract from a document is to split it into text windows (spans) and try all possible extractions. Since these are probabilistic models, we can compute $P(phone|span)$, and find the window span that maximizes the probability.

$$best = max_{span}P(phone|span) \tag{2}$$

$$P(phone|span) = \prod_{i=0}^{9} max_j P(d_i = j|span) \tag{3}$$

The phone number extracted from the best span is the phone number in the text.

We collected a set of real advertisements that don't have phone numbers, and artificially inserted an obscured number from our artificial dataset. This allows us to track which span contains the phone number, and then evaluate an extractor.

The difficulty with this task is that our models are trained on precise text spans, whereas this full document dataset contains lots of non-phone-related text. To address this difference, we stopped padding our snippet input with null values (up to the length of the RNN), and instead pad with randomly selected text snippets from real ads. The models are exactly the same, we just change how padding works when the training text is shorter than length 70. We refer to this as the "ad pad".

Datum:	6I5 093 93B6
Null-Pad:	6I5 093 93B6_____________________
Ad-Pad:	6I5 093 93B6always in town call

To be clear, no models were changed, just how training input is padded. Can the models identify the correct text span that contains a phone number? Table 4 shows these results for standard null-padding versus ad-padding, as well as cross-domain tests. We trained on Craigslist and Backpage separately, then tested on only Backpage ads.

Text Span ID of Phone Numbers

	Full	**Full+Partial**
Zero pad	70.3%	92.1%
Craigslist ad pad	99.3%	99.7%
Backpage ad pad	98.0%	99.6%

Table 4: Results of choosing text spans with the full phone number, or a partial match. Partial matches contained on average 7-8 of the 10 digits.

Window identification works very well as long as training padded its input with real ad text. This is encouraging in that it seems these models can reliably identify *where* a phone number is present.

Finally, we tested how the models also extract from these spans after identifying them. Extraction showed 80% accuracy on full numbers, compared to 98% when train/test only on artificial phone snippets. We attribute the drop to the difficult task - window spans contain more noise than a precise text span. Future work will focus on this full document task with real-world numbers.

9 Discussion

This is the first work to model noisy phone number extraction with neural models. Most notably, our CNNs explore how to use visual characteristics of the characters, rather than standard NLP-style models with trained embeddings. To the best of our knowledge, this is the first proposal for a visual language model in an extraction architecture.

We showed results on new challenge datasets with injected unicode. These results illustrate the challenge for extractors, but also the usefulness of CNN recognizers. In fact, current rule-based extractors cannot extract any of the numbers in our test sets. Our CRF outperformed an LSTM-only model by 10% absolute, and data augmentation improved on unicode tests by a relative 89% gain.

Possible future work could investigate a Generative Adversarial Network (GAN) (Goodfellow et al., 2014). GANs have become popular in vision tasks, but the normal GAN setup requires training data to start from, and this sparse domain prohibits its straightforward use.

Data from this work's training and evaluation are available online[4], and we hope this spurs further work on this important societal challenge.

[4]www.usna.edu/Users/cs/nchamber/data/phone/

10 Acknowledgments

This work would not be possible without the help of the Global Emancipation Network. Many thanks also to Jeff Kosseff for bringing this issue to our attention in the first place. We recognize and appreciate the support of the DoD HPC Modernization Office for enhancing our undergraduate education and research. Finally, thanks to Rebecca Hwa for helpful conversations early on in this work.

References

Hamidreza Alvari, Paulo Shakarian, and J. E. Kelly Snyder. 2017. Semi-supervised learning for detecting human trafficking. In *Semi-supervised learning for detecting human trafficking*.

Hamidreza Alvari, Paulo Shakarian, and J.E. Kelly Snyder. 2016. A non-parametric learning approach to identify online human trafficking. In *IEEE Conference on Intelligence and Security Informatics (ISI)*.

Andrei Costin, Jelena Isacenkova, Marco Balduzzi, Aurélien Francillon, and Davide Balzarotti. 2013. The role of phone numbers in understanding cybercrime schemes. In *2013 Eleventh Annual Conference on Privacy, Security and Trust*, pages 213–220. IEEE.

Jun Ding, Bo Chen, Hongwei Liu, and Mengyuan Huang. 2016. Convolutional neural network with data augmentation for sar target recognition. *IEEE Geoscience and remote sensing letters*, 13(3):364–368.

Artur Dubrawski, Kyle Miller, Matthew Barnes, Benedikt Boecking, and Emily Kennedy. 2015. Leveraging publicly available data to discern patterns of human-trafficking activity. *Journal of Human Trafficking*, 1.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

Rahul Kapoor, Mayank Kejriwal, and Pedro Szekely. 2017. Using contexts and constraints for improved geotagging of human trafficking webpages. In *Proceedings of the Fourth International ACM Workshop on Managing and Mining Enriched Geo-Spatial Data*.

Mayank Kejriwal, Jiayuan Ding, Runqi Shao, Anoop Kumar, and Pedro Szekely. 2017. Flagit: A system for minimally supervised human trafficking indicator mining.

Mayank Kejriwal and Pedro Szekely. 2017. Information extraction in illicit web domains. In *WWW*.

John Lafferty, Andrew McCallum, and Fernando CN Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

Lin Li, Olga Simek, Angela Lai, Matthew P. Daggett, Charlie K. Dagli, and Cara Jones. 2018. Detection and characterization of human trafficking networks using unsupervised scalable text template matching. In *IEEE International Conference on Big Data (Big Data)*.

Frederick Liu, Han Lu, Chieh Lo, and Graham Neubig. 2017. Learning character-level compositionality with visual features. In *ACL*.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*.

C. Nagpal, K. Miller, B. Boecking, and A. Dubrawski. 2017. An entity resolution approach to isolate instances of human trafficking online.

Reihaneh Rabbany, David Bayani, and Artur Dubrawski. 2018. Active search of connections for case building and combating human trafficking. In *KDD*.

Justin Salamon and Juan Pablo Bello. 2017. Deep convolutional neural networks and data augmentation for environmental sound classification. *IEEE Signal Processing Letters*, 24(3):279–283.

Pedro Szekely, Craig Knoblock, Jason Slepickz, Andrew Philpot, et al. 2015. Building and using a knowledge graph to combat human trafficking. In *International Conference on Semantic Web (ICSW)*.

Yan Xu, Ran Jia, Lili Mou, Ge Li, Yunchuan Chen, Yangyang Lu, and Zhi Jin. 2016. Improved relation classification by deep recurrent neural networks with data augmentation. *arXiv preprint arXiv:1601.03651*.

Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. 2017. Random erasing data augmentation. *arXiv preprint arXiv:1708.04896*.