Part II
Appendix

Table of Contents

A Supplementary Statistics

Table A: Vote Results

	Salinas	Cárdenas	Clouthier	Total Votes	Polling Stations
Vote Tallies	9,294,147	5,314,667	3,269,208	18,207,388	52,288
	(0.510)	(0.292)	(0.179)		
Official Data	9,641,329	5,956,988	3,267,159	19,145,012	54,493
	(0.503)	(0.311)	(0.171)		

Notes: This table compares the vote total and vote shares of the three main candidates using the official results and the information from the tally sheets. Vote shares are in parenthesis.

Table B: Summary Statistics on the Information from the Vote Tally Sheets

	N	Mean	S.D.	Min	Max
Salinas (PRI)	53288	174.413	208.27	0	6080
Clouthier (PAN)	53288	61.35	106.14	0	4436
Ibarra (PRT)	53288	2.18	12.10	0	592
Castillo (PDM)	53288	4.00	17.03	0	1802
Cárdenas (FDN)	53288	99.73	131.65	0	2280
Total Votes	53288	349.76	303.57	29	9429
PRI Agent	53288	0.62	0.47	0	1
PAN Agent	53288	0.50	0.50	0	1
FDN Agent	53288	0.47	0.50	0	1
PDM Agent	53288	0.09	0.30	0	1
PRT Agent	53288	0.07	0.25	0	1
Poll Worker Signature 1	53288	0.93	0.25	0	1
Poll Worker Signature 2	53288	0.93	0.25	0	1
Poll Worker Signature 3	53288	0.90	0.29	0	1
Poll Worker Signature 4	53288	0.88	0.32	0	1

Notes: This table reports summary statistics for the information obtained from the vote tally sheets. The unit of observation is the polling station. The information of party agents and poll workers' signatures are dummy variables that equal 1 for each observation where the individual signed the tally sheet.

B Experiment Description

The survey experiment discussed in Section 4.2 used Qualtrics survey technology with two population samples. The respondents for the first sample were recruited through Amazon's Mechanical Turk via a HIT advertised as "Find altered tallies. A 15 minute survey" Respondents were restricted to those in the United States with HIT approval rates greater than or equal to 95% and at least 1,000 HITs approved. Respondents were provided $1.70 compensation for taking the survey. Survey data from 200 respondents was collected on February 14, 2017. Each respondent was presented with 10 random images from the Training Set and were asked to identify those files that present alterations in their numbers. The average response time was 7.4 minutes (SD=2.8 minutes) and 62% of the respondents correctly answered the attention check.

The second sample used the answers of 4 students at the University of Houston. Students' responses were collected during March 14 and July 8, 2017. Each respondent got a sample of 50 random images from the Training Set and were asked to identify those files that present alterations in their numbers. Respondents received $15 compensation for taking the survey. The average response time was 92 minutes (SD=56.4 minutes). 75% of the respondents correctly answered the attention check. Neither Amazon Turk respondents nor undergraduate students were informed about the label that the images were originally assigned.

Both studies were approved by the University of Houston Institutional Review Board (STUDY00000131 and STUDY00000301).

C Supplementary Figures

C.1 Vote Tally Sheet

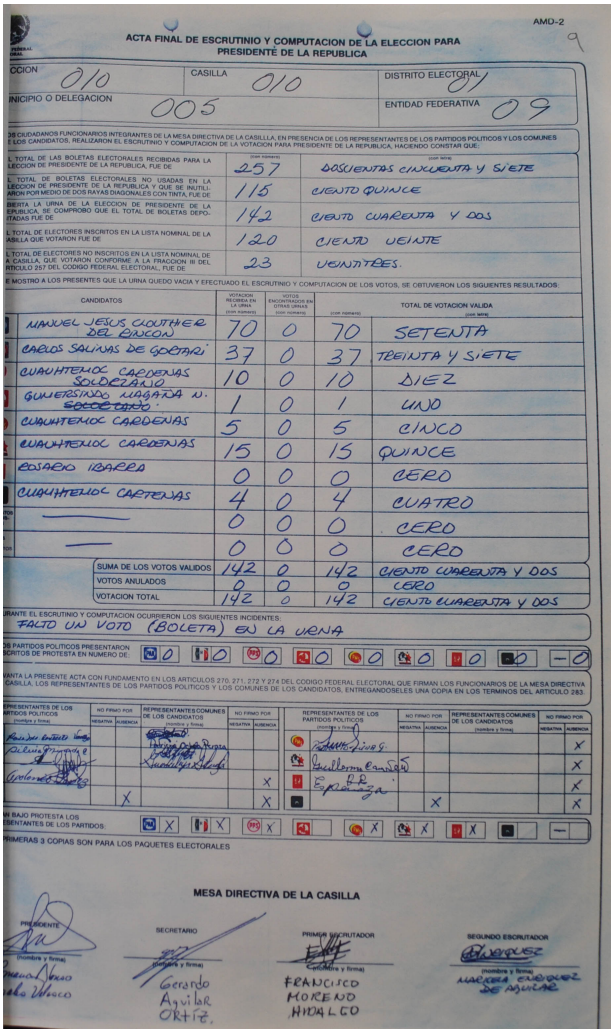


Figure C.1: Example of a Digitized Vote Tally Sheet. Mexico, 1988

C.2 Examples of Altered Tallies

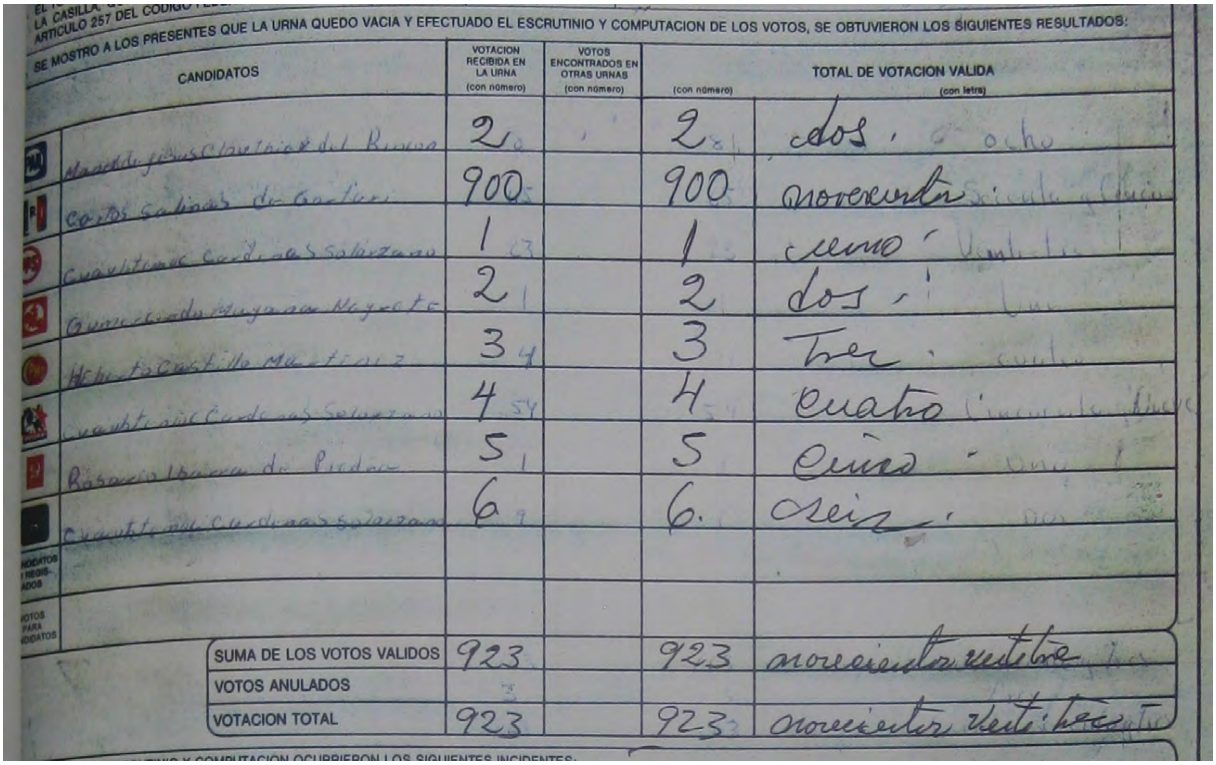
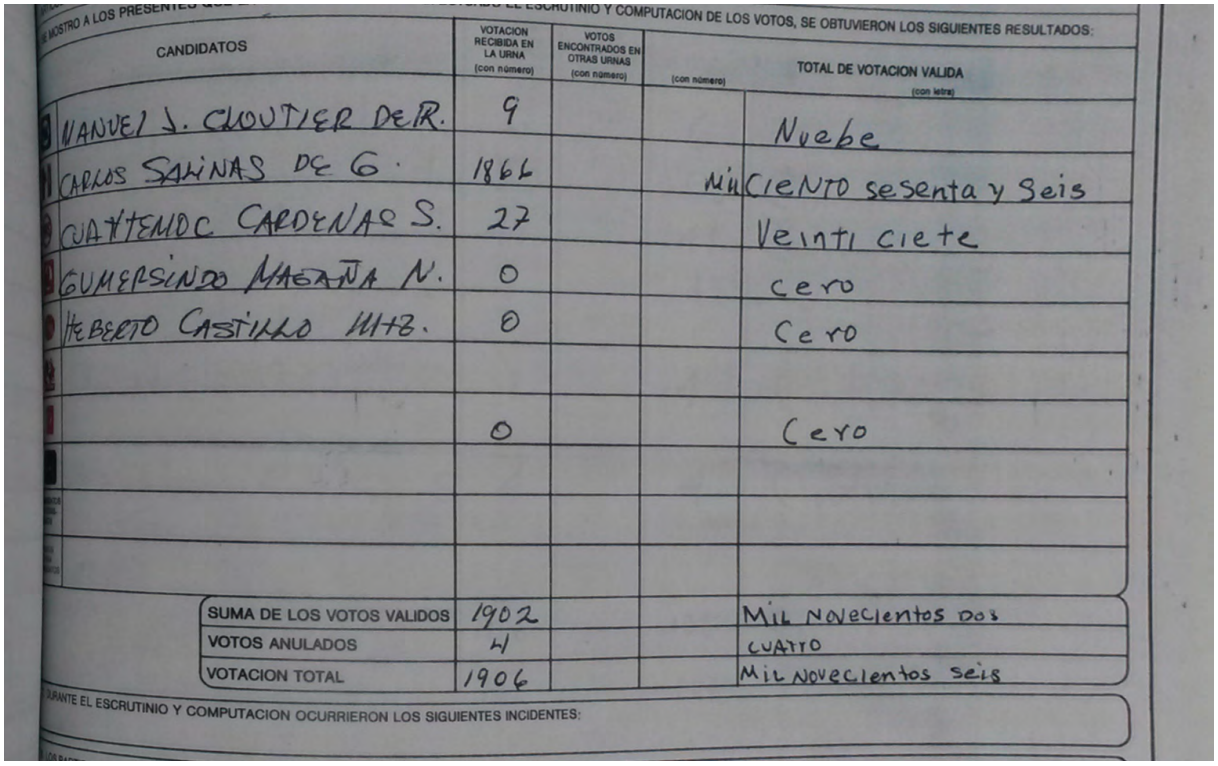


Figure C.2: Example of an Altered Vote Tally Sheet. Puebla, District 14. Mexico, 1988
Notes: This picture shows an example of an altered tally in the State of Puebla. The votes for the first candidate, Manuel Clouthier, were eight, as it is written in words ("ocho" in Spanish) and numbers in blue. However, the amended tally shows the number 2 on top of it. Carlos Salinas, the second candidate on the tally, got originally 65 votes, but the amended tally gave him 900 votes as it is shown with a different handwriting and ink color than the original numbers. The amendments for the rest of the candidates' vote totals follow a sequential order: 1, 2, 3, 4, 5, and 6.



CANDIDATOS	VOTACION RECIBIDA EN LA URNA (con número)	VOTOS ENCONTRADOS EN OTRAS URNAS (con número)	TOTAL DE VOTACION VALIDA (con número)	TOTAL DE VOTACION VALIDA (con letra)
MANUEL J. CLOUTIER DEL R.	9			Nueve
CARLOS SALINAS DE G.	866		Mil	CIENTO sesenta y Seis
CUAUHTEMOC CARDENAS S.	27			Veinti ciete
GUMERSINDO MAGAÑA N.	0			cero
HEBERTO CASTILLO MTZ.	0			Cero
	0			Cero
SUMA DE LOS VOTOS VALIDOS	1902			Mil Novecientos Dos
VOTOS ANULADOS	4			CUATTO
VOTACION TOTAL	1906			Mil Novecientos seis

Figure C.3: Example of an Altered Vote Tally Sheet. Puebla, District 5. Mexico, 1988

Notes: This picture shows an example of an altered tally in the State of Puebla. The votes for PRI's Carlos Salinas were 166 as it is written in words ("Ciento sesenta y seis" in Spanish). There were three types of amendments to the tally. First, they edited the first digit to transformed the "1" to "8." Second, they added and additional "1" at the left of the number. Finally, they added the word "a thousand" ("Mil" in Spanish) at the end of the letter-written number, showing a different alignment and handwriting than the rest of the words on the tally.

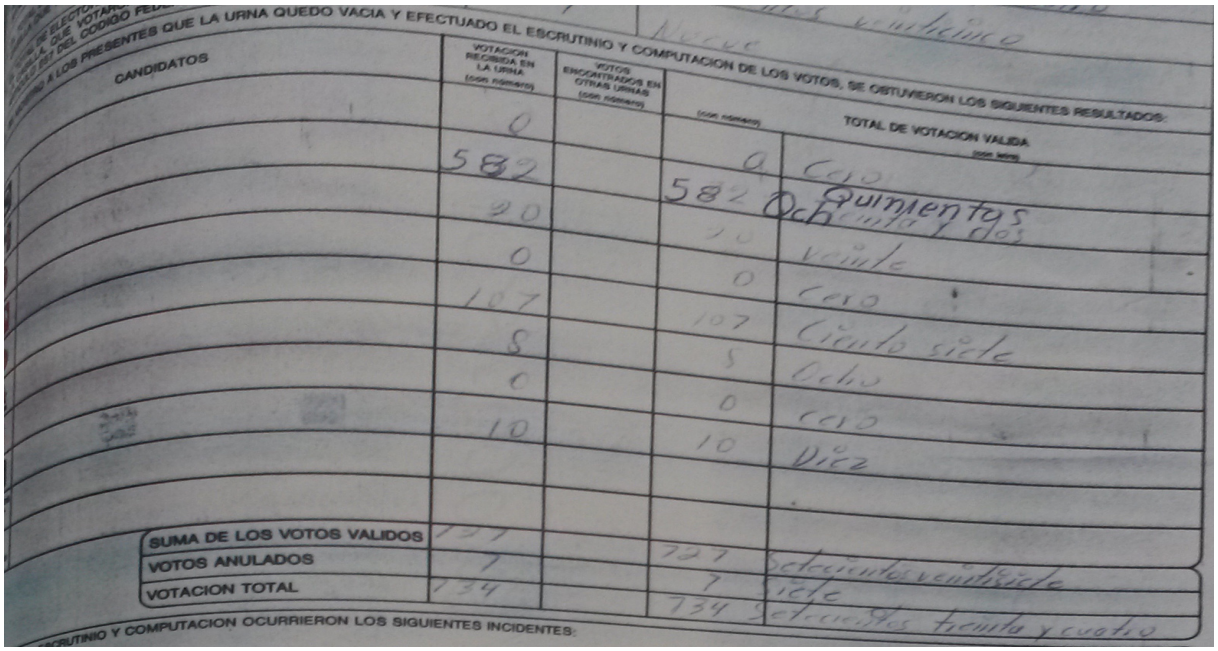
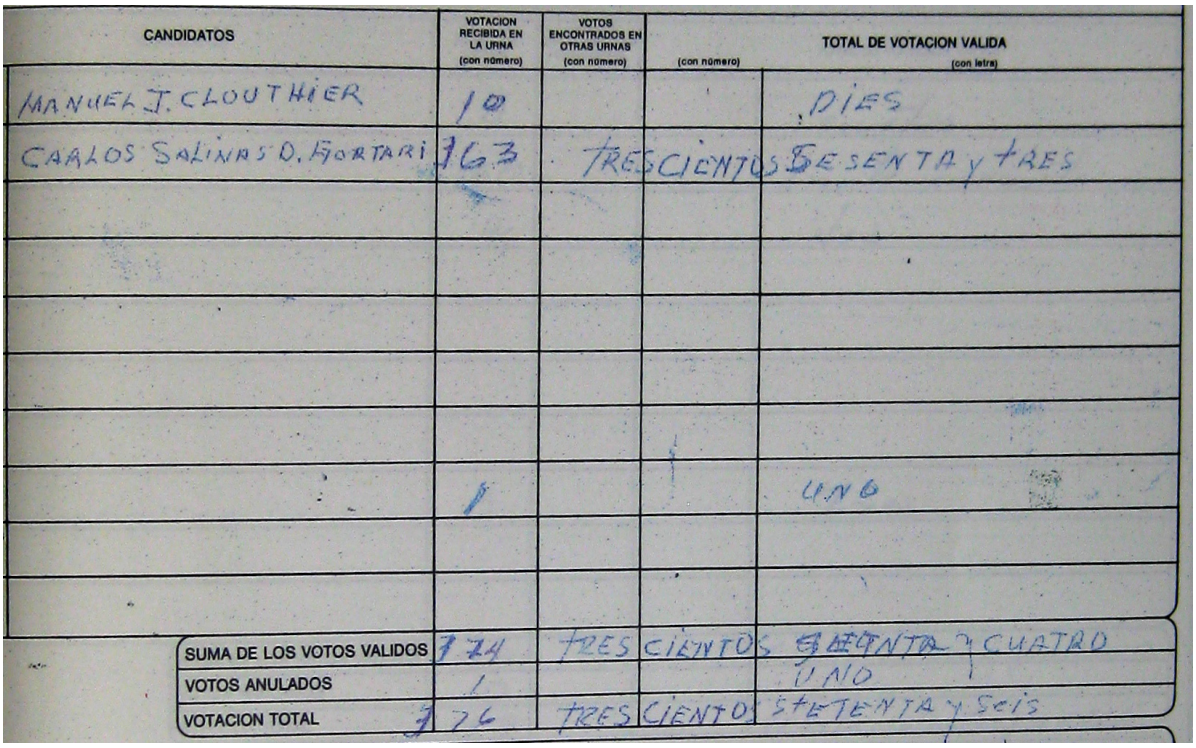


Figure C.4: Example of an Altered Vote Tally Sheet. Veracruz, District 9. Mexico, 1988
Notes: This picture shows an example of an altered tally in the State of Veracruz. The votes for PRI's Carlos Salinas were 32. The amended tally shows a loop-closed "3" to make an "8" as well as an additional 5 at the left of the original number. The amendments to the letter-written vote total shows similar amendments.



CANDIDATOS	VOTACION RECIBIDA EN LA URNA (con número)	VOTOS ENCONTRADOS EN OTRAS URNAS (con número)	TOTAL DE VOTACION VALIDA (con número)	(con letra)
MANUEL J. CLOUTHIER	10			DIES
CARLOS SALINAS D. GORTARI	163	TRES	CIENTOS	SESENTA y TRES
	1			UNO
SUMA DE LOS VOTOS VALIDOS	174	TRES	CIENTOS	SETENTA y CUATRO
VOTOS ANULADOS	1			UNO
VOTACION TOTAL	176	TRES	CIENTOS	SETENTA y Seis

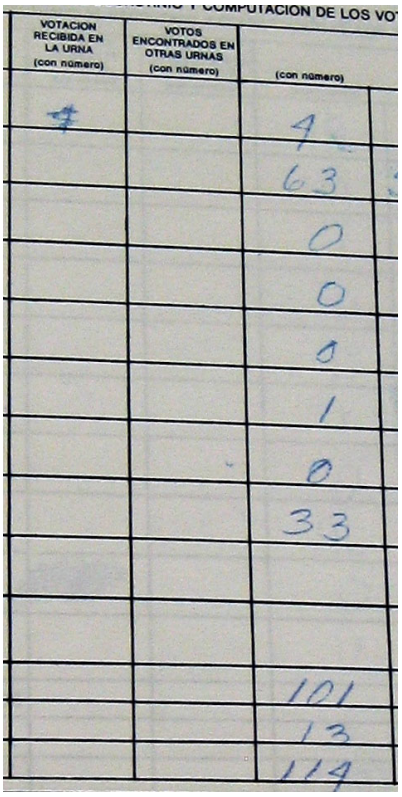
Figure C.5: Example of an Altered Vote Tally Sheet. Nuevo León, District 6. Mexico, 1988
Notes: This picture shows an example of an altered tally in the State of Nuevo León. The votes for PRI's Carlos Salinas were 63. It was first added a "1" to the left of the number, but this number insertion was later amended to transform it into a "3." and t. The amendments on the letter-written numbers show different alignment and handwriting that the original numbers.

C.3 Potential replaced tallies

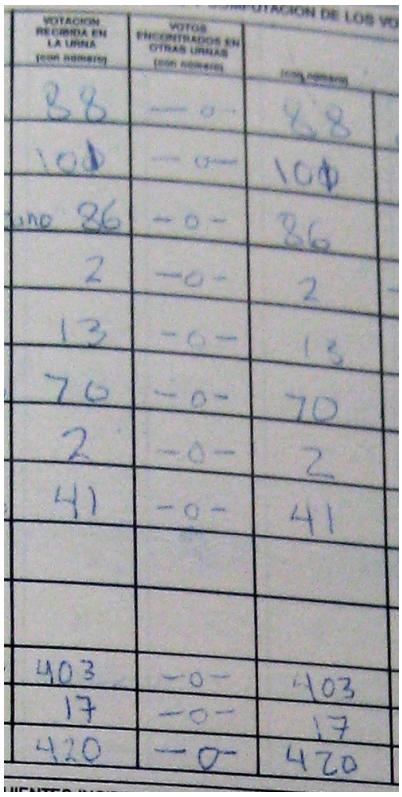
Figure C.6: Examples of vote tallies from the Second District of Chiapas. Mexico, 1988

C.4 Examples of cases included in the training set

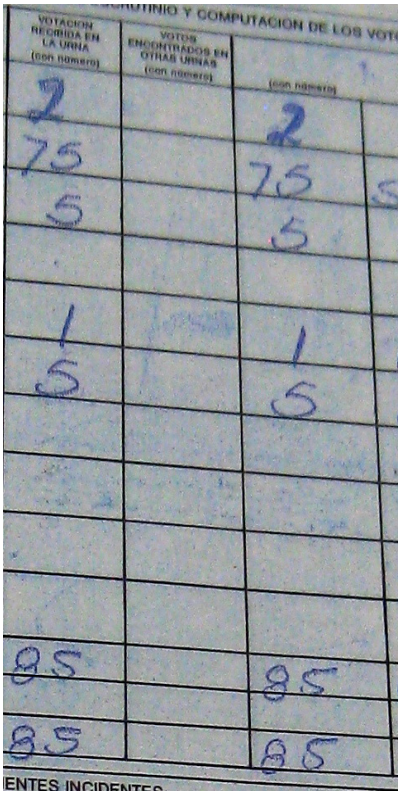
The examples labeled as "without alterations" were selected from images that did not present deliberate alterations in their numbers. To make sure that the model can only distinguish deliberate alterations on the tally, I included in this set of images two kind of examples. First, I incorporate images of tallies showing benign amendments or accidental errors, such as misplaced numbers or marginal corrections to a candidate's vote return (Figure C.7). Including these examples helps the model to distinguish among different adjustments on the tally. Second, I also included images where a candidate gets all the votes registered in the tally but there are no clear patterns of alterations in their numbers (Figure C.8). These examples force the model to focus on the amendments on the results rather than their distribution across candidates.



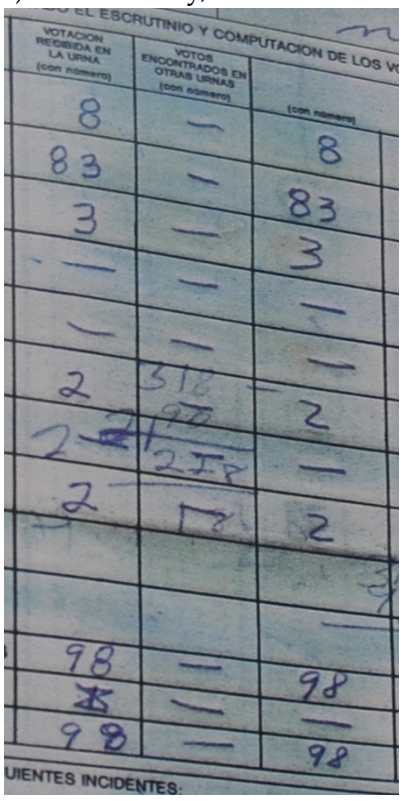
(a) Coahuila, District II



(b) Mexico City, District XXVII

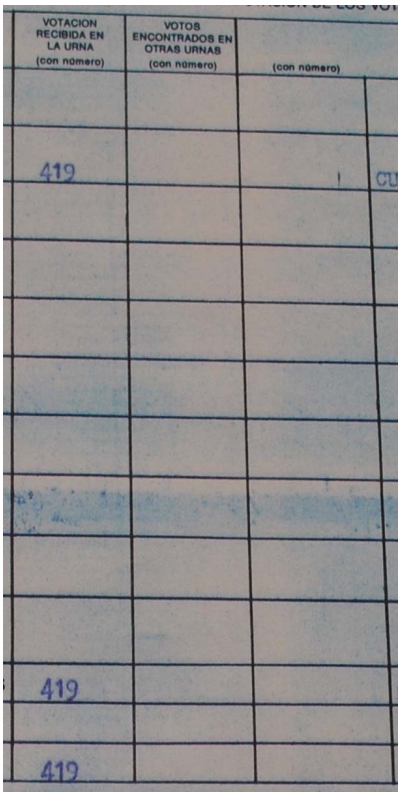


(c) Guanajuato, District I

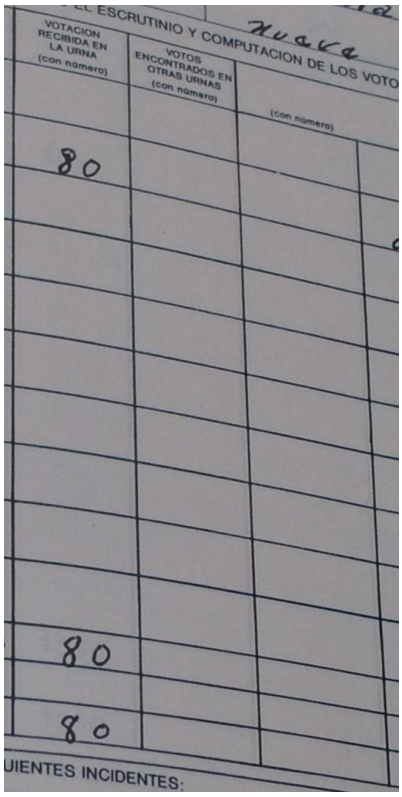


(d) Jalisco, District VII

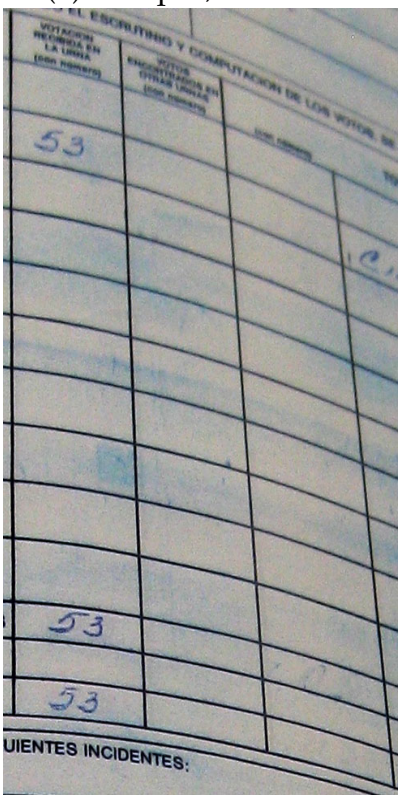
Figure C.7: Examples of vote tallies with no alteration in their numbers. Mexico, 1988



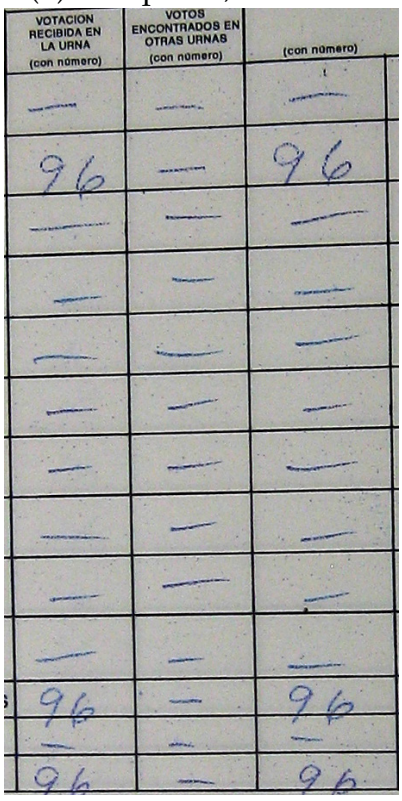
(a) Chiapas, District II



(b) Campeche, District II



(c) Durango, District III



(d) Nuevo Leon, District XI

Figure C.8: Examples of vote tallies with no alteration in their numbers. Mexico, 1988

C.5 Analysis of the tallies from the 2015 legislative election

To further validate the model inferences, I test its accuracy on a different dataset. In particular, I use the images of the tallies for the 2015 legislative election in Mexico. Unlike the 1988 election, the 2015 vote-counting process was open to all political parties at every stage, and the images of all tallies filled at the polling stations were available online 24 hours after closing the polls. The Expert Survey of Perceptions of Electoral Integrity (Norris et al., 2015) evaluates the integrity of the 2015 legislative election in Mexico. In a scale from 1 to 5, where 1 means "Strongly Disagree" and 5 means "Completely Agree," experts' mean answer to the statement " Votes were counted fairly" was 4. Also, their mean answer to "The authorities allowed public scrutiny of their performance" was 3.5. To pre-process the images, I used a computer script to download all the pictures and crop the tally-area with the vote numbers. The images of all tallies are available at http://prep2015.ine.mx. The model labeled these tallies as "with alterations" only 5 percent of the time—within the expected measurement error. A further inspection to the misclassified cases suggests that most of them correspond to tallies that were slightly misplaced on the website, and the resultant cropped images included printed features of the tally alien to the examples in the training set.

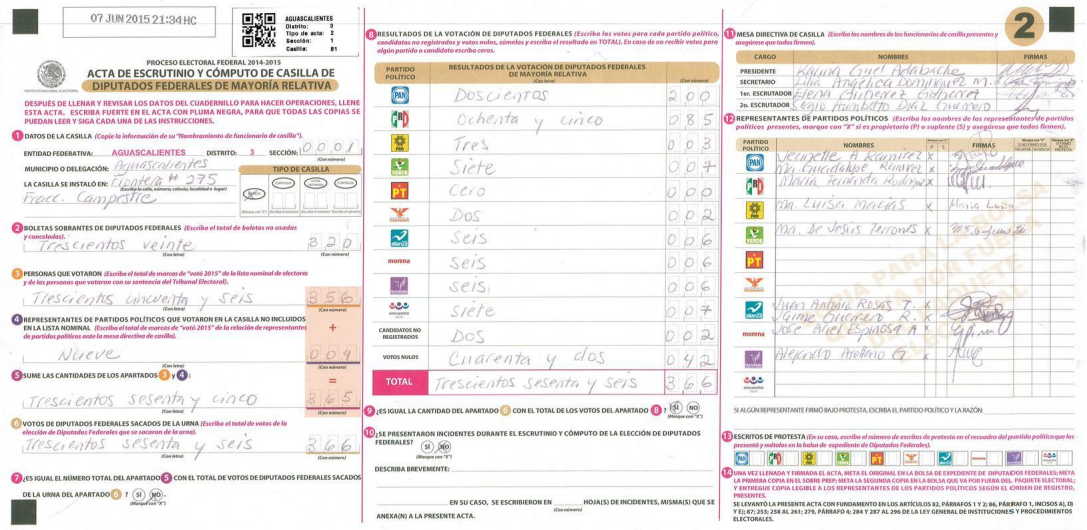


Figure C.9: Example of a Digitized Vote Tally Sheet. Mexico, 2015

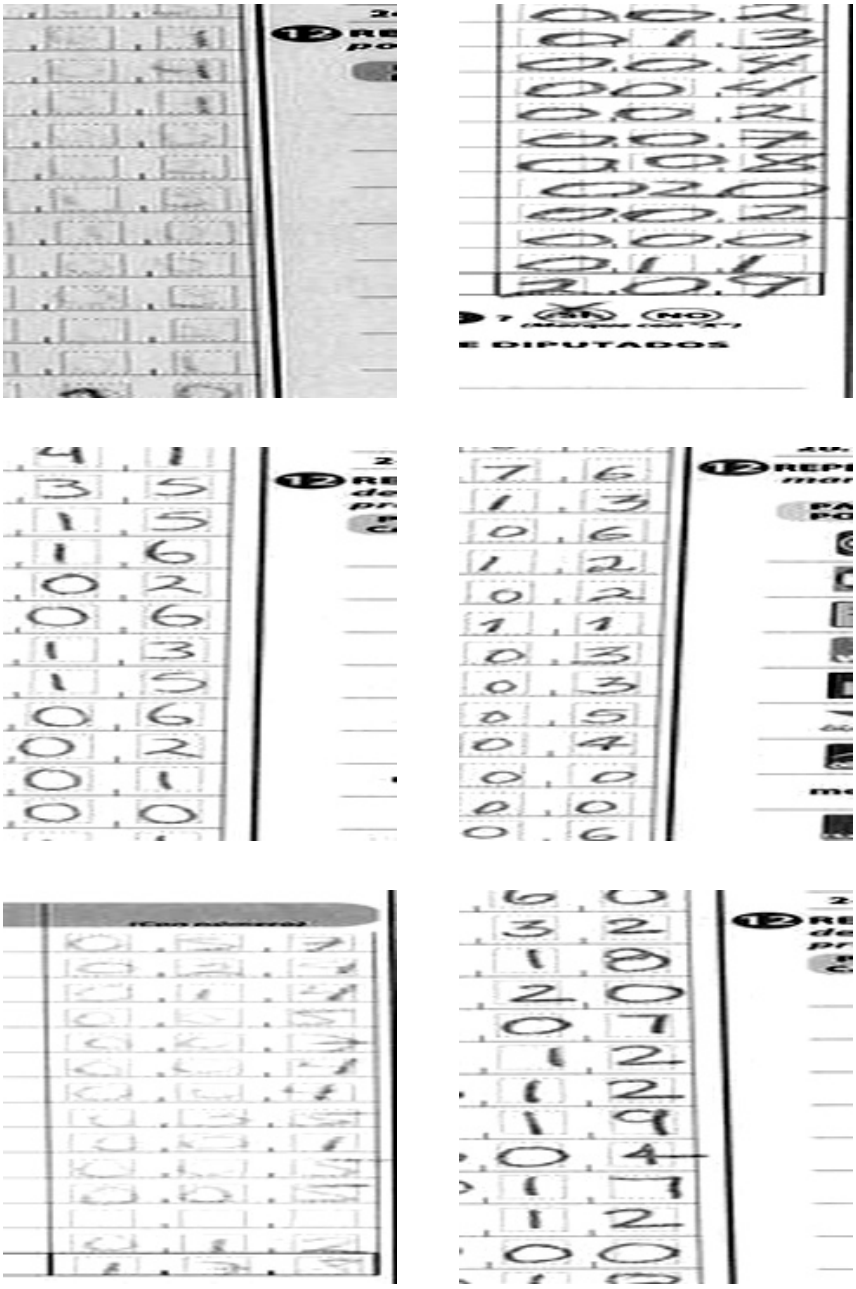


Figure C.10: Examples of misclassified images for the 2015 election in Mexico.

D Network Structure

Table C: Network configuration summary

Layer (type)	Output Shape
Zero Padding 2D	(3, 233, 233)
Convolution 2D	(32, 229, 229)
Activation (ELU)	(32, 229, 229)
Pooling 2D	(32, 114, 114)
Zero Padding 2D	(32, 120, 120)
Convolution 2D	(32, 118, 118)
Batch Normalization	(32, 118, 118)
Activation (ELU)	(32, 118, 118)
Pooling 2D	(32, 59, 59)
Zero Padding 2D	(32, 65, 65)
Convolution 2D	(64, 63, 63)
Batch Normalization	(64, 63, 63)
Activation (ELU)	(64, 63, 63)
Pooling 2D	(64, 31, 31)
Zero Padding 2D	(64, 37, 37)
Convolution 2D	(64, 35, 35)
Batch Normalization	(64, 35, 35)
Activation (ELU)	(64, 35, 35)
Pooling 2D	(64, 17, 17)
Zero Padding 2D	(64, 23, 23)
Convolution 2D	(128, 21, 21)
Batch Normalization	(128, 21, 21)
Activation (ELU)	(128, 21, 21)
Pooling 2D	(128, 10, 10)
Zero Padding 2D	(128, 16, 16)
Convolution 2D	(256, 14, 14)
Batch Normalization	(256, 14, 14)
Activation (ELU)	(256, 14, 14)
Pooling 2D	(256, 7, 7)
Dropout	(256, 7, 7)
Flatten	(12544)
Dense	(2048)
Activation (ELU)	(2048)
Dropout	(2048)
Dense	(128)
Activation (ELU)	(128)
Dropout	(128)
Dense	(1)
Activation (sigmoid)	(1)

The most common concern when training a CNN model is the risk of overfitting, which occurs when the model "memorizes" image features that are not generalizable outside the training set. I tackle this problem in two ways. First, I artificially increase the size of my training set by producing new images derived from random shears, flips, rotations, and zooms of the original pictures (Chattfield, 2014). Second, I detract the model from focusing too much on specific features of an image by blocking a random set of units in the neural network. This technique helps the model to consider those features that can be generalizable to multiple images (Srivastava, 2014).

D.1 Specifications

Zero Padding: Zero padding adjusts the input volume by placing zeros around the image border. This technique prevents that the information at the borders of the image would be "washed away" after passing through the convolutional layer. It also allows the use of deeper networks because it slows down the volume size of the image.[17]

Convolution: Every time the image passes through a convolutional layer, each of its filters slides across every 3×3 pixel area of the image looking for basic features, such as a straight line, an edge, or a curve. The output of each filter generates a new representation of the image.

Activation (ELU): ELU stands for Exponential Linear Unit and is used during the convolution operation to identify positive values of the image input. Unlike other activation units—e.g., the Rectified Linear Units (ReLU) or the Parametrized Rectified Linear Units (PReLU)—ELUs consider negative values, which improves learning in a very efficient way (Clevert, Unterthiner and Hochreiter 2016).

[17] http://cs231n.github.io/convolutional-networks/.

Pooling: Pooling layers gradually reduce the spatial dimension of the input image by decreasing its number of parameters. They work by downsampling every depth slice in the image by 2 units of both width and height, reducing the number of parameters by 75%. The purpose of this layer is to speed the convolution process as the image goes deeper through the network. It also reduces overfitting by forcing the computer not to focus on the exact location of a feature but, instead, on its relative location to other features (Scherer, Müller and Behnke 2010).

Batch Normalization: The goal of normalization is to transform the outputs of the convolutional layers to parameters with zero mean/unit variance. This transformation allows the layer activations to be appropriately handled by any optimization method during the training phase. The goal of this technique is to avoid the network to focus on outlying activations and to speed its learning (Ioffe and Szegedy 2015).

Dropout: Dropout layers are included to reduce overfitting during the training stage. As its name suggests, these layers "drop out" a random set of activations in the layer. This function forces to provide the right classification based in more than one specific activation (Srivastava et al. 2014). The model included three dropout layers, each of them blocking 20%, 30%, and 50% of the neurons before moving to its respective fully connected network.

Dense: The resulting image representations from the last convolutional layer are transformed into a unidimensional vector and sent to three fully connected layers that gradually glean the features more likely to correlate with each class. The first vector has 2048 Exponential Linear Units, which then pass to a second vector with 512 Exponential Linear Units. The outputs of the second layer are sent to a third vector with only one unit which makes whether the image has been altered.

Activation (sigmoid): The last activation layer has a function of form $f(x) = \frac{1}{1+exp^{-x}}$. It therefore follows an S-shaped curve and produces value outcome between 0 and 1.

The model is compiled using a binary cross-entropy loss function. This function is the standard choice for binary classifications and it aims to maximize the accuracy of the predicted labels. The loss function is estimated as $Loss = -\frac{1}{N}\sum_{n=1}^{N}[y_n log(\hat{y}_n)+(1-y_n)log(1-\hat{y}_n)]$, where y and $\hat{y}$ are the vectors for the true and predicted labels, respectively (Rubinstein and Kroese 2004). During the learning process, the model uses an gradient descent optimizer that calibrates the filter weights to gradually minimize the loss function. In particular, I use the *Adadelta* algorithm, which does not requires to specify a learning rate for the gradient to reach the local minimum (Ruder 2016).

The author acknowledges the use of the Opuntia Cluster and the advanced support from the Center of Advanced Computing and Data Systems at the University of Houston to carry out the research presented here.

References:

Clevert, Djork-Arné, Thomas Unterthiner and Sepp Hochreiter. 2016. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)." arXiv:1511.07289v5

Ioffe, Sergey and Christian Szegedy. 2015. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." https://arxiv.org/abs/1502.03167

Rubinstein, Reuven Y. and Dirk P. Kroese. 2004. *The Cross-Entropy Method*. New York: Springer.

Ruder, Sebastian. 2016. "An overview of gradient descent optimization algorithms." https://arxiv.org/abs/1609.04747

Scherer, Dominik, Müller, Andread and Sven Behnke. 2010. "Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition." 20th International Conference on Artificial Neural Networks (ICANN), Thessaloniki, Greece. http://www.ais.uni-bonn.de/papers/icann2010_maxpool.pdf

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov. 2014. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." Journal of Machine Learning Research, 15: 1929-1958.

D.2 Code

```
#Upload Keras modules (available at keras.io)
#I am running keras with the tensorflow backend.
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import array_to_img
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense, ZeroPadding2D
from keras.callbacks import History
from keras.callbacks import ModelCheckpoint
from keras.layers.normalization import BatchNormalization
from keras.layers.advanced_activations import ELU
from keras.models import load_model

#os is just one of default python libraries
import os

#numpy is numeric python
import numpy as np

# Change the image size to 227x227.
# Accuracy is much higher for squared images.
# DO NOT MIX IT UP.
img_width, img_height = 227, 227

train_data_dir = 'TrainingSets/'
nb_train_samples = 900 # number of samples in the training set

validation_data_dir = 'Validation/'
nb_validation_samples = 150 # number of samples in the validation set

nb_epoch = 250 # how many epochs to train for. We are loading existing weights.
# so not needed unless training on new data

window_sz = 3 # how many pixels is the window that slides across the image is

# this will initiate a sequential backpropagation network
model = Sequential()

# this adds 3 rows of zeros (black color pixels) to top of images and 3 columns
# to the sides. This is to prevent "washing away" of the sides. Convolutional nets
# tend to assume that anything on the edge is not important.
```

```
model.add(ZeroPadding2D(padding=(3, 3),
input_shape=(227,227,3), data_format="channels_last"))

# 32 is the number of filters I first use. So it is the dimensionality of the output,
# or how many transformations the image goes through.
model.add(Conv2D(32, (window_sz, window_sz)))

# Batch Normalization helps the learning process to find
# consistent patterns in the batch
BatchNormalization()

# This adds a non-linear layer that is in our case Exponential Linear
# Unit. This is where learning happens through backpropagation.
model.add(ELU())

# Pooling layer, it is used to improve speed. Usually after we learned some things
# from initial image, it is harmless to downsample the image some. So we are pooling
# together every 4 pixels and taking an average, making it 1.
model.add(MaxPooling2D(pool_size=(2, 2)))

# The rest is just the above repeated five more times.
# As the network goes deeper, I include larger layers by increasing its filters
model.add(ZeroPadding2D(padding=(3, 3)))
model.add(Conv2D(32, (3, 3)))
BatchNormalization()
model.add(ELU())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(ZeroPadding2D(padding=(3, 3)))
model.add(Conv2D(64, (3, 3)))
BatchNormalization()
model.add(ELU())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(ZeroPadding2D(padding=(3, 3)))
model.add(Conv2D(64, (3, 3)))
BatchNormalization()
model.add(ELU())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(ZeroPadding2D(padding=(3, 3)))
model.add(Conv2D(128, (3, 3)))
BatchNormalization()
model.add(ELU())
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(ZeroPadding2D(padding=(3, 3)))
model.add(Conv2D(256, (3, 3)))
BatchNormalization()
model.add(ELU())
model.add(MaxPooling2D(pool_size=(2, 2)))

#with each iteration we randomly turn off 20% of the neurons. This is a biological
#idea that works quite well. Basically, we are forcing the network to not focus
#too much on one single thing. If it does that, it becomes obsessed with little
#patterns ignoring the big picture. So this is sort of like how brain reacts to
#a sensory overload - receptors just start ignoring further stimulation.
model.add(Dropout(0.2))

#now we take the output which is a square and turn it into a 1D vector
model.add(Flatten())

#now that we have a vector we can put into a vector of 4096 Rectified Linear Units
#so the final conclusion can be made
model.add(Dense(4096))
BatchNormalization()
model.add(Activation('elu'))
model.add(Dropout(0.3))
model.add(Dense(512))
model.add(Activation('elu'))
model.add(Dropout(0.5))

#the last layer makes the decision. Decision is made by just 1 neuron, it says
# fake or not.
model.add(Dense(1))
BatchNormalization()
model.add(Activation('sigmoid'))

#now the network is created.
model.compile(loss='binary_crossentropy',
              optimizer='adadelta',
              metrics=['accuracy'])

# this is the augmentation configuration we for training. This just creates
# additional images. So if we say choose to flip an image, we now have a normal image
# and a copy of it that is flipped.
train_datagen = ImageDataGenerator(
     rescale=1./255, #because neural nets like numbers in range 0-1, we divide by 255
     shear_range=0.3, #we shear the image a little
     zoom_range=0.3, #zoom in and out
```

```
    horizontal_flip=True, #randomly flip some images
    vertical_flip=True,
    samplewise_center=True,
    rotation_range=30,
    channel_shift_range=5)


# this is the augmentation configuration for testing:
# only rescaling
test_datagen = ImageDataGenerator(rescale=1./255,
                                  samplewise_center=True)


#so to train the model I uncomment the following
#train_generator = train_datagen.flow_from_directory(
#        train_data_dir,
#        target_size=(img_width, img_height),
#        batch_size=16,
#        class_mode='binary')

# uncomment this to validate on the test set
#validation_generator = test_datagen.flow_from_directory(
#        validation_data_dir,
#        target_size=(img_width, img_height),
#        batch_size=16,
#        class_mode='binary')

# this is where you train or fit the data, this line actually executes it.
model.fit_generator(
        train_generator,
        samples_per_epoch=nb_train_samples,
        validation_data=validation_generator,
        nb_val_samples=nb_validation_samples,
        nb_epoch=nb_epoch,
        callbacks=callbacks_list,
        verbose=2)
```