# Learning to See: Convolutional Neural Networks for the Analysis of Social Science Data*

Michelle Torres
Washington University in St. Louis
smtorres@wustl.edu

Francisco Cantú
University of Houston
fcantu10@uh.edu

**Abstract**

This paper presents the functioning, implementation, and challenges of Convolutional Neural Networks (CNNs), one of the most popular tools for classifying visual information. We propose that the adequate use of CNNs reduces the resources necessary for the tedious task of classifying images and extracting information from them. To illustrate the advantages and implementation of this methodology, we describe a potential application of CNNs to the collection and analysis of vote results on Election Day. We use this tool to code handwritten information from the vote tallies of the 2015 federal election in Mexico. Our paper not only demonstrates the contributions of CNNs to both scholars and policy practitioners, but also presents the practical challenges and limitations of the method, providing advice on how to deal with these issues.

*We are grateful to Sarah Bouchat, Bryce Dietrich, Jonathan Homola, Chris Lucas, Jacob Montgomery and participants of the 2018 Latin American Political Methodology Meeting and the Political Science Data Lab at Washington University in St. Louis for useful comments. All errors are our own. Replication materials will be available in the websites/Github pages of the authors.

# 1 Introduction

Computers have increasingly taken over tedious and repetitive tasks previously assigned to humans. From counting words in a file to performing long and complex calculations, machines are able to follow a set of instructions in a repetitive manner without fatigue or cognitive bias. Their capacity to perform quickly and reliably allows us to analyze information from a large amount of data such as roll-call votes (McCarty, Poole and Rosenthal, 2006), congressional floor speeches (Dietrich, Hayes and O'Brian, 2019; Dietrich, Enos and Sen, 2019), or social media posts (Barberá, 2015).

And yet, despite their capacity to analyze textual or numeric information, computers have historically underperformed when classifying visual data. In principle, computers would be able to find all images containing a specific candidate if we provide specific rules describing the physical characteristics of the individual (e.g., the form of her nose, the distance of her nose to her eyes, and the size of her forehead). But computers' ability to follow a set of rules when classifying an image can actually be their main limitation. The directives we give to a computer could be insufficient to identify those pictures in which the individual is not facing the camera or wearing sunglasses. In an even subtler dimension, the computer might struggle to identify the individual if the lighting differs substantially from one picture to another. Of course, we can provide more instructions for the computer, but the instruction list would be as vast as the number of ways in which an individual may appear in a picture.

To overcome those challenges, recent developments in computer science propose an alternative approach to retrieve information from images. Rather than following a set of given rules, computers are now exposed to multiple examples that allow them to identify visual patterns across images. This process, inspired by the way in which humans learn to digest visual content, allows computers to gradually glean patterns of colors, edges, contours and textures that correspond to a given object. This learning process allows the computer to identify and track an object under various conditions.

This paper introduces to political science the use of CNN as a reliable, cost-effective way to classify pictures. In particular, we present this method as an alternative tool for the tedious task of analyzing, coding and classifying large-scale image collections. Our main goal is to provide gen-

eral guidelines on how CNNs work and to discuss the challenges and problems that researchers might encounter when using this tool by focusing on a canonical example of data collection and processing. To illustrate this methodology, we apply a CNN to a relevant issue in election science: coding voting results. We present a way to collect the handwritten results registered in vote tallies from the 2015 federal election in Mexico, captured in a set of comprehensive images. This example allows us to illustrate the characteristics of this method and the intuition behind its performance, providing an exhaustive guide for its implementation while still highlighting the challenges it poses, as well as offering recommendations and alternatives to overcome those challenges.

The use of CNNs can help social scientists expand research on topics like the visual tone of campaign coverage (Lawson and McCann, 2004; Druckman and Parkin, 2005), the portrayals of protests (Casas and Webb Williams, Forthcoming; Torres, 2018; Won, Steinert-Threlkeld and Joo, 2017), and the camera-recorded interactions between police officers and citizens (Makin et al., Forthcoming). Further, analyzing images of documents allows researchers to recover and collect information such as signatures, annotations or signs of alterations to answer questions related to, for instance, the decision to continue fighting in a war (Huff, 2018), historical political participation (Homola, 2018), or electoral fraud (Cantú, 2018).

On the other hand, this paper stresses the limitations of the methodology and the risks of its mindless application in social sciences. A valid concern regarding CNNs stems from their opacity for linking the inputs and the model outputs (Nguyen, Yosinski and Clune, 2015; Sabour, Frosst and Hinton, 2017; Zeiler and Fergus, 2014). Such warning should deter scholars from applying this method to identify latent dimensions or ambiguous features in the data. Since post-hoc interpretations to the outcomes of a deep learning model largely depend on the researcher's subjective intentions (Lipton, 2016), we emphasize the importance of establishing transparent goals when using the model and restricting its application to tasks that could be performed, in principle, by a human.

The article intends to guide social scientists through the mechanisms in which CNNs work, to "translate" the language of computer science into terms and concepts that are more familiar to social scientists. We acknowledge that the methodology includes plenty of jargon. While some of these names have to do with exclusive concepts from computer science, many of them have an equivalent label that is familiar to most political scientists. The Appendix includes a glossary of

these terms, which we identify with italics in the text.

We first introduce what CNNs are and explain each stage in the process. Next, we illustrate the practicability of this tool by applying CNNs to capture the vote results of the 2015 election in Mexico directly from vote tallies. Finally, we provide a list of challenges and recommendations when applying these tools, as well as a discussion of the limitations of CNNs for certain measurement and classification tasks.

## 2 A Primer on Convolutional Neural Networks (CNNs)

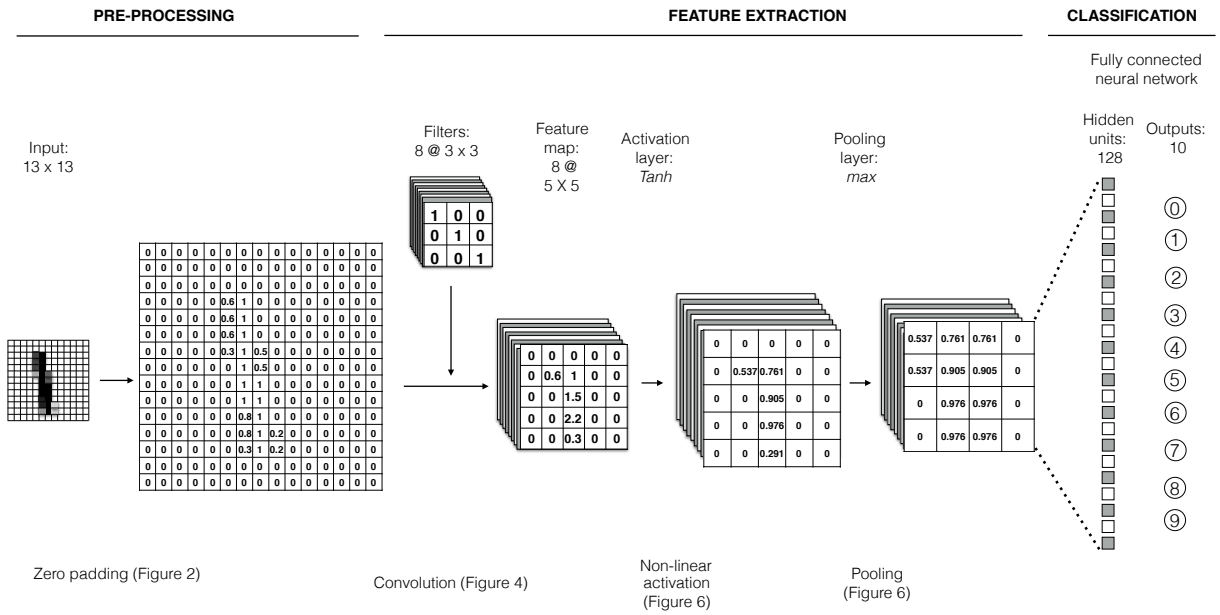
A CNN is a special case of a neural network, which consists of a set of inter-related nodes that classify data after processing a set of training examples. The main components of these networks are the *neurons*, which have the task of receiving an input, transforming it, and sending the new output to other neurons. Neurons transform the input received according to their *activation function*. The output of this activation function becomes a signal that another neuron will use as an input for its own computation. The relevance of every released signal for the final outcome is determined by the *weights* between the emitting and receiving neurons. Positive weights amplify the signals and highlight their contribution to the output, while negative weights weaken the signal to which they are attached. At the end of the network, there are as many nodes as labels of interest. Each final node delivers the probability that the original input belongs to each specific label. The neural network tries to minimize the errors in predictions by gradually modifying the weights of the preceding nodes (Schrodt, 2004; Lucas, 2018).

CNNs distinguish themselves from other types of neural networks by their capacity to process visual inputs. CNNs are organized into layers, or sets of neurons representing specific visual features of edges, blobs, textures or colors. Each neuron convolves with sub-regions of the original image to search for similar visual patterns. In essence, this technique addresses the complexity of representing simple objects by reducing the high dimensionality of images. Thus, instead of attributing meaning to raw pixel intensities, the CNN summarizes the content of an image by using informative features and patterns that are learned throughout the process.[1].

[1] *Deep belief networks* use the raw pixel intensities as inputs. However, beyond concerns for computational capacity, they show a poor performance given that they do not have built-in invariance with respect to translations or local distortions of the data. Further, the risk of overfitting using these techniques is high (LeCun and Bengio, 2003)

To describe the functionality of CNNs, we divide the process into three parts. First, the pre-processing stage transforms the image into a format that can be read by the computer. Next, the feature extraction stage deconstructs the image into multiple visual components, each representing a specific visual feature. Finally, the classification stage uses the image's components to classify the image into one of the available categories Figure 1 illustrates these stages and provides a road map for the figures in the manuscript, detailing and illustrating each step. We explain below the logic behind each of the stages, as well as the decisions available to the researcher at each of them.

Figure 1: Convolutional Neural Network Structure




## 2.1 Image Pre-Processing

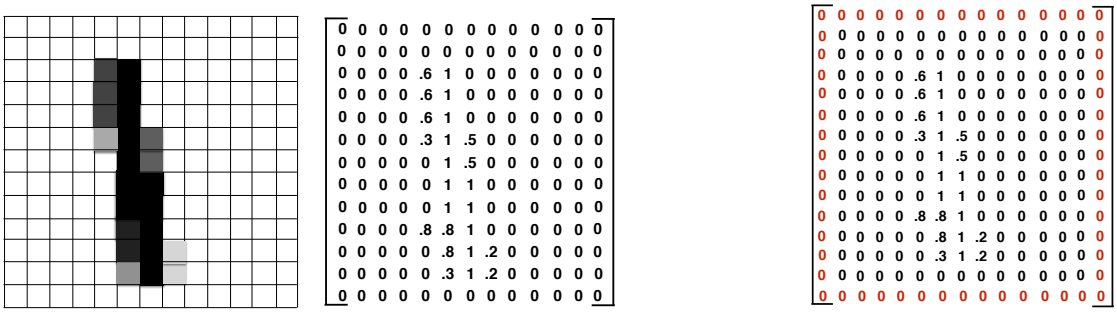
For the computer to analyze visual information, it is first necessary to represent the image as a numerical array where each of the entries has a specific pixel value. Figure 2(a), for example, illustrates how a handwritten number of 13 (height) $\times$ 13 (width) pixels can be transformed into a matrix of $13 \times 13 = 169$ units, each of them specifying the light intensity of a specific pixel.[2] In

[2]The concept of "amount of light" might seem counterintuitive when expressed in mathematical form: In practice, a value of "0" corresponds to a black pixel, while "255" represents a white pixel. To avoid confusion and for illustrative

the case of a color image, the transformation would produce three matrices of the same size, one for each primary color channel (red, green, and blue).

The resulting input matrix is the core unit of analysis. The goal of the CNN is to extract the most relevant information from this matrix while gradually reducing its dimensionality. However, the way in which CNNs extract the information tends to give less importance to the features on the edges of the image given that the convolution in these areas is partial. We prevent this problem by applying *zero-padding*, or appending a perimeter of zeros to the input matrix. The example in Figure 2(b) shows a zero padding of $p = 1$, increasing the numerical array to $15 \times 15$.

Figure 2: Image Pre-processing of a handwritten "1"



(a) Image Transformation

(b) Zero Padding

## 2.2 Feature Extraction

Once the image enters into the network, it is decomposed into single components. The decomposition process consists of computing the dot product of the input matrix with a set of smaller matrices, called *filters*, which represent a particular visual feature. The first layers include representations as basic as straight lines (see Figure 3 as an illustration), and subsequent layers build up on those features and transition from lines to contours, to shapes, and to objects (Buduma and Locascio, 2017). The more layers a CNN has, the more complex features of the image will be

purposes, we take higher numbers in the matrixes presented as higher concentrations of "ink". Therefore, higher numbers correspond to darker pixels.

recognized (Qin et al., 2018).

Within each layer, filters slide across the width and height of the input matrix to obtain the dot product with a particular area of the image, also called *receptive field*. This dot product is the *convolution* part of the CNN. The entries in each filter matrix can be understood as *weights* on pixel intensity that altogether form relevant or informative patterns. We are interested in learning the relevant combinations of weights that are associated with our outcome labels, just as we want to learn the importance of coefficients in a regression model. Thus, a convolutional layer is a collection of filters extracting different information from the same input matrix.

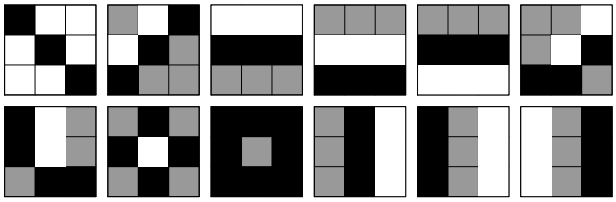
To describe in detail how convolution works, we need to specify three parameters of this operation. First, the *filter size* is the product of the filter's width and height. This parameter sets out the type of features identified during the convolution. Small filters capture fine-grained details, but they are likely to mix up the relevant information from an image with its noise. On the other hand, large filters look for details of a larger size at the cost of a lower specificity. For example, Figure 3 presents simplified examples of filters of size 9. Second, the *filter stride* is an integer number defining how many pixels the filter will slide through the image. The smaller the stride, the more information from the image is preserved during the convolution.[3] Finally, the *layer depth* defines the number of filters in the layer. This value, therefore, indicates how many features will be searched for in the layer. The optimal choice for these elements depends on our data and classification goal in every case.[4] Figure 3, for instance, represents the filters of a layer with depth 12.

We illustrate how a convolution process works in Figure 4. In this example, we take the top-left filter displayed in Figure 3 and use a stride of 3. The convolution process then involves computing the dot products of the filter and the values of every equivalent pixel space in the image. In this example, the dot product between the entries of the filter and the input of the highlighted image area is 2.2. The filter then slides three steps to the right and computes again the dot product of

[3]The magnitude of this parameter depends on the size, dimensions and characteristics of the data and CNN. For a comparison of model performances using different strides and filter size, see Simonyan and Zisserman (2014).

[4]For the CNN used in this article, the filters in the first layer are initialized randomly using the Glorot uniform method. Also known as Xavier, this initializer draws samples from a uniform distribution: $W \sim \mathcal{U}\left(\frac{-6}{u_{in}+u_{out}}, \frac{6}{u_{in}+u_{out}}\right)$, where $u_{in}$ is the number of input units in the weight tensor, and $u_{out}$ is the number of output units in the weight tensor. In most canned architectures, it is not necessary to define the initialization of these filters.

Figure 3: Examples of filters

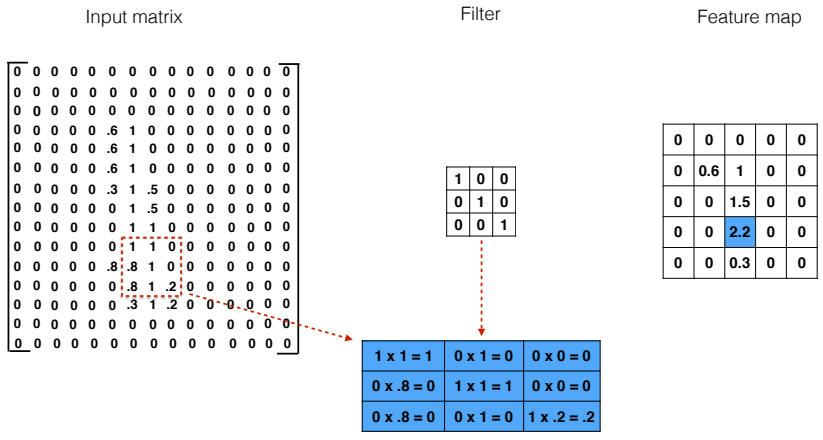


*Note:* This figure displays examples of filters of size $3 \times 3 = 9$ that were randomly initialized in the first layer of a CNN.

its entries. The result of this operation is the *feature map* at the right of Figure 4, which shows the image regions with the largest dot products for this filter. The convolution process will create as many feature maps as filters specified in the layer depth, and the size of each feature map is defined by:

$$\text{feature map size} = \frac{(\text{input width} \times \text{input length}) - \text{filter size} + (2 \times \text{zero padding})}{\text{stride} + 1} \tag{1}$$

Figure 4: Illustration of the Convolution Stage

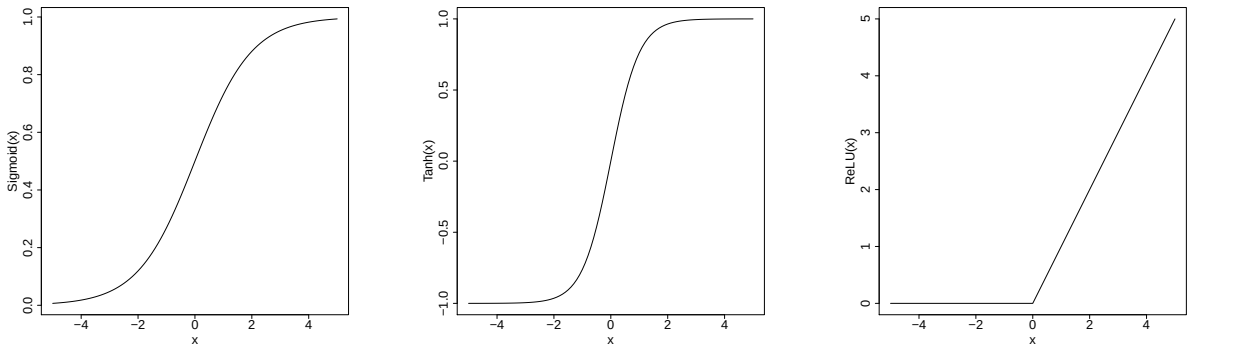



Since the resulting feature map is a linear transformation of the input matrix, adding more

convolutional layers at this point would be redundant—the result could be obtained with a single linear product. Such feature maps are unlikely to produce smooth gradients, a necessary input for the learning phase described in Section 2.3. To address this problem, it is necessary to include an *activation layer*, which performs a non-linear mathematical operation upon the input values. The non-linearity property allows CNNs to stack multiple layers and extract more information from the image.

We illustrate three of the most common activation functions in Figure 5. The first one, *Sigmoid*, is simply the inverse of the logistic function. As the figure shows, this function bounds the activation values to the $[0, 1]$ range. The second activation function is *Tahn*, is a linear transformation of *Sigmoid* that zero-centers the outputs and bounds them to the $[-1, 1]$ interval. While both functions are very sensitive to input values closer to $0$, their output becomes flat near its boundaries, limiting the network to learn from inputs with either very low or very high activation values. Addressing these issues, the *Rectified Linear Unit* (*ReLU*) is a non-saturated function that keeps the original input value when it is positive and transforms all negative values to $0$. $ReLU$ usually increases the learning speed of the network, and is now the standard activation function in practice (Nair and Hinton, 2010). Again, the choice of any of these or other activation functions depends on the performance of each of them to the specific data in question.

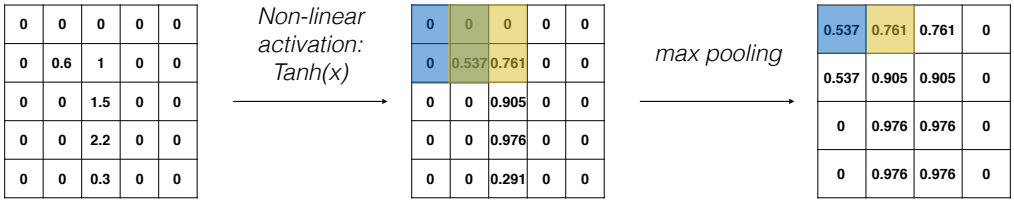
Figure 5: Example of Activation Functions




(a) $Sigmoid(x) = \frac{1}{1+e^{-x}}$ (b) $Tanh(x) = \frac{2}{1+e^{-2x}}$ (c) $ReLU(x) = \begin{cases} 0 & \text{if } x < 0, \\ x & \text{otherwise.} \end{cases}$

Once the activation map includes non-linear outputs, we proceed to reduce its dimensionality using a *pooling* layer. A pooling layer shrinks the size of the matrix while keeping the most im-

Figure 6: Illustration of the non-linear activation and pooling




portant information in the feature map. The information of a submatrix is summarized differently depending on the type of pooling layer applied. For example, it can get the largest value (*max pooling*), the smallest value (*min pooling*), or the average value (*mean pooling*) of a specific pixel area.

Figure 6 illustrates the nonlinear transformation of our feature map using a *Tanh* function. After that, we apply *max-pooling* to keep the largest value from every 2x2 pixel area of the matrix. The resultant matrix generalizes the properties of the image, forcing the CNNs to pay more attention to whether a feature fits in the image regardless of the location of such feature.

The process is repeated for each of the filters in the first layer, and each of the resultant feature maps becomes the input for the second convolutional layer. This new convolutional layer repeats all of the steps described above, where the new filters slide through the feature maps to now look for more complex features, such as a combination of lines or edges. The more layers we include in the network, the more complex features it is able to extract and learn from the images.

### 2.3 Learning

During the learning stage, the network processes the input information by identifying those feature maps and their corresponding weights that are more likely to define a given outcome label. The learning process occurs throughout several iterations in which the predictions from the network are compared to the actual true outputs to have a measure of error. The model's goal is to find an optimal combination of feature maps and weights that minimize the disparities between true and predicted labels.

The predicted labels are the outputs of the last layer, in which the feature maps are flattened and stacked into a single vector to create a fully connected neural network. This is, a network

where all the nodes are connected, so changes in one node alter the output of another node. More concretely, if we have 10 outcomes of interest as shown in Figure 1, the network outputs a set of probabilities of belonging to each of these outcomes. The dependency between nodes in these fully connected layers implies that if we have an increase in the probability that an input digit is a "3", then the probability of *not* being a "3" (the combined probability of being one of the rest of the digits) has to decrease.

If the classification goal is only between two labels, the CNN maps predictions to these probabilities using a Sigmoid layer (recall Figure 5(a)) or inverse logit. When the number of labels is greater than two, the probabilities are estimated through a *softmax layer* layer, which is the equivalent of a multinomial logistic function. In the example described above, since we want to identify the image from Figure 2(a) as a digit value, the last layer of our network has 10 neurons or outputs, one for every digit from 0 to 9. Each neuron will provide a probability that the image belongs to each digit, and the CNN will attach the label with the highest estimated probability.

The model gradually calibrates the estimated probabilities as it revisits the examples in the training set. This process tries to emulate the way in which infants learn to recognize object categories. As they are exposed to multiple examples, young children subconsciously assimilate the distinctive properties of an object until they can identify an object without observing all of its features.[5] Similarly, CNNs review multiple examples of an object to gradually identify salient features, determine their relevance, and estimate how much each of them helps to identify the object across other images. Computer science labels such process as *backpropagation* (Rumelhart, Hinton and Williams, 1988), and it consists of minimizing the error of the model's predictions by gradually calibrating the weights in each feature map among the neurons in the network.

This error optimization process requires finding those points in a high-dimensional space where the derivative of the error function is zero. To achieve that goal, the CNN marginally alters the weight of an individual connection within the layer and looks for the largest variations on the resulting error. This estimation of change in the multi-dimensional plane, or *gradient descent*, allows us to figure out the steepest path that leads to the bottom of the error function. We present a more technical description of this process in the Appendix.

[5]However, humans execute this process better, faster and more efficiently than computers. While the number of samples that a child requires to learn how a particular object looks like is relatively small, a computer requires a significantly larger number of training samples to learn features more accurately.

The learning phase requires training the model after specifying a few practical features. First, it is necessary to define the number of *epochs*, or the number of times that all training examples pass through the network. Since the gradient descent is an iterative process, we need several epochs to optimally fit the weights to the model. Too many epochs, however, are likely to produce overfitting and reduce the generalizability of the model's predictions (see Subsection 4.1.1). Unfortunately, there is not an optimal number of epochs with which to train a model, and its final setting depends on the specific characteristics of the training database and a close tracking by the researcher.

Second, we need to define the *batch size*, or the number of training images that will pass through the network before the model updates its weights. It is possible to set the batch size to the total number of training images, so the model would update its weights once per epoch. This modality requires accumulating the prediction errors across all of the images in the training set, a task that can demand a lot of computational memory. It also produces a static error surface, where the gradient descent is likely to get stuck in a local minimum. Alternatively, we can set the batch size to one, where the model updates its weights for each training example. This modality produces a dynamic error surface, decreasing the risk of getting stuck on a flat region. However, updating the model with every example produces a very noisy signal, making the gradient descent jump around. The sweet spot between both extremes splits the training set into *mini-batches*, allowing the model to update its parameters several times during an epoch. Research on the topic suggests setting the batch size to 32, balancing the noisy training problems of smaller batches as well as the slower convergence of larger batches (Bengio, 2012; Masters and Luschi, 2018).

Finally, we need to set the *learning rate*, or the speed at which the gradient descent travels along the downward slope. This rate specifies the degree to which the CNN will update its weights after every iteration. A large learning rate will produce large-scale updates on the network weights, jumping around the function and overshooting its minimum. In contrast, a very small learning rate is more likely to find a local minimum, but it will take a long time to converge. It is suggested, then, to start with a large learning rate and gradually decrease it at every iteration (Buduma, 2017). Finding the optimal learning rate for every parameter can be a demanding task. Fortunately, there is a variety of optimizers that adaptively tune the learning rates for all parameters in the model.[6]

[6]For a very helpful comparison of the most common optimizers, see Karpathy, Andrej. 2019. "CS231n Convolutional

### 2.4 Software

There are multiple sources of software to design and run a Convolutional Neural Network. All the analyses in this paper, including the manipulation and processing of visual material, were conducted using Python 3, and within it, `OpenCV` and `Keras` (with a `TensorFlow` backend). Keras is a neural networks API written in Python that supports models like CNNs and recurrent networks, and allows a very accessible, efficient and user-friendly interaction with libraries like `TensorFlow`, `CNTK` and `Theano`. These are libraries that allow the design, training and implementation of machine learning models. However, there are other tools that facilitate the design of the architecture of a CNN, and that also allow researchers to take advantage of pre-trained models. These include, but are not limited to, *Amazon AWS Machine Learning Training*, *Google Cloud AutoML* or *Google Cloud Machine Learning Engine*[7].

## 3 Implementation: Coding Electoral Results from Vote Tallies

This section illustrates the use of CNNs for coding factual content from images. Examples of this type of information involve handwritten notes in treaties and documents, signatures, annotations, or vote counts. The human transcription of such material is historically either delegated to multiple coders or ignored given the high costs that its processing implies.

We apply CNNs to code the vote results for Mexico's 2015 federal election. This example demonstrates the benefits of visual analysis not only to scholars but also to policy practitioners and election officials looking for a cost-efficient way to speed up the vote tabulation process. For the specific case of Mexico, the automatic capturing of the electoral results could decrease the rate of tallies with accidental errors when adding up the votes, which occurs in almost two out of five tallies in the country (Challú, Seira and Simpser, 2018). Moreover, this technology can also shorten the time between the closing of the polls and the results announcement, a period of distress for candidates and voters. Similar problems are found in Florida, Arizona, Haiti, or Argentina, where the delays for announcing the results spotlight the importance to register the results in a fast and accurate way.[8] We thus propose this tool as a way to increase not only the efficiency of the

Neural Networks for Visual Recognition." https://cs231n.github.io/neural-networks-3/ (March 30, 2019).

[7] For a review of these platforms and their performance, see (Williams, 2019).

[8] *The New York Times*, "Races in Arizona Still Hang in the Balance." November 9, 2012. (http://www.nytimes.

vote count but also citizens' trust on the impartiality of the process (Atkeson and Saunders, 2008; Bowler et al., 2015; Pastor, 1999).

Figure 7 shows an example of one of the tallies under analysis.[9] We compiled a dataset with 104,979 images of tallies. All ballots have the same content, structure and format. One of the few differences is the number of parties competing in each district, so therefore the number of rows with handwritten digits range between 13 and 15.

The first step involves extracting the handwritten numbers of the the tally. Because the alignment and orientation might differ from image to image, we decided to develop a function that identifies the coordinates of three focal points of the tally: the yellow banner at the top of the page, the bright pink rectangle at the bottom left of the tally, and the pink circle below the table. The coordinates of these elements, shown inside red rectangles in the first element of Figure 7, allow us to identify the bottom, top and left lines of the table containing the digits. The green dashed lines and yellow area in the second element of the diagram illustrates this process. Once we isolate the table, we divide it into 3 $\times$ *the number of parties/candidates in the district* cells. We then cut and save each cell under the assumption that it contains a digit.[10]

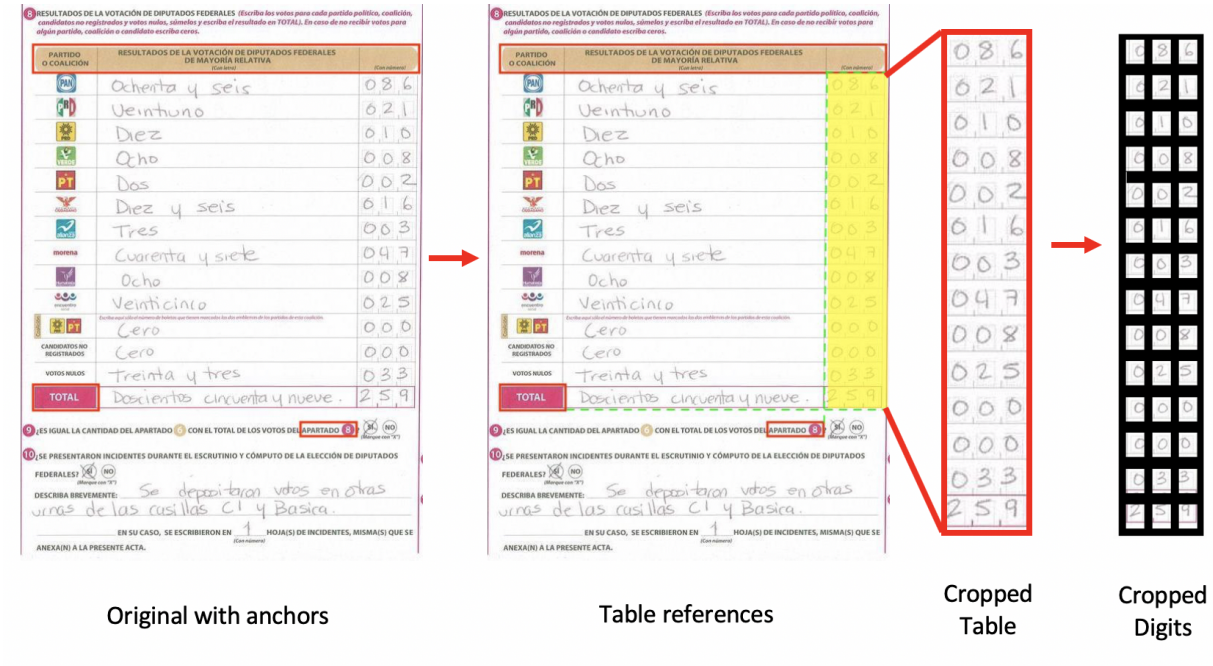
Figure 8 summarizes the architecture of the CNN for this task. The network consists of two convolutional and pooling layers interleaved, followed by two fully connected layers and a terminal *softmax*. To train the model, we apply *transfer learning*, which involves using the weights of an

com/2012/11/10/us/politics/arizona-races-still-hang-in-the-balance-over-uncounted-votes.html); *Los Angeles Times*, "Arizona ballots finally counted – and Latinos ask, Why so long?" November 21, 2012. (http://articles.latimes.com/2012/nov/21/nation/la-na-nn-arizona-latinos-voting-20121121); and *Tucson Sentinel* "Why is Arizona still counting votes?" November 21, 2012. (http://www.tucsonsentinel.com/local/report/112012_az_vote_count/why-arizona-still-counting-votes/); *The New York Times*, "Vote Count Confirms Obama Win in Florida." November 10, 2012. (http://www.nytimes.com/2012/11/11/us/politics/florida-to-address-delays-as-it-confirms-obama-victory.html); *National Public Radio*, "Four Days Later, Florida Declares For Obama." November 10, 2012. (http://www.npr.org/sections/thetwo-way/2012/11/10/164859656/florida-finishes-counting-obama-wins); *BBC*, "Haiti starts counting votes in long-delayed election." November 21, 2016. (http://www.bbc.com/news/world-latin-america-38042585); *Reuters*, "Haiti police clash with demonstrators ahead of election results." November 22, 2016. (https://www.reuters.com/article/us-haiti-election/haiti-police-clash-with-demonstrators-ahead-of-election-results-idUSKBN13I05K); *Clarín*, "Elecciones PASO 2017: Cristina Kirchner denunciará la "trampa electoral" del Gobierno y apuntará a todos los votos peronistas." August 14, 2017. (https://www.clarin.com/politica/elecciones-paso-2017-cristina-kirchner-denunciara-trampa-electoral-gobierno-apuntara-votos-peronistas_0_SJghMNJ_Z.html).

[9] For the purposes of our example, we only show the center panel of the full tally. The original full tally contains a horizontal panel composed of three sheets: the first one with information about the polling station, the second one with the tabulation of the votes per party, and the third one with relevant signatures from party representatives and polling station authorities.

[10] This, however is not fulfilled in some cases. Although polling staff is supposed to fill all cells and use leading zeros for 1 and 2-digit numbers, or parties with no support, several ballots have empty cells.

Figure 7: Example of the image of a tally

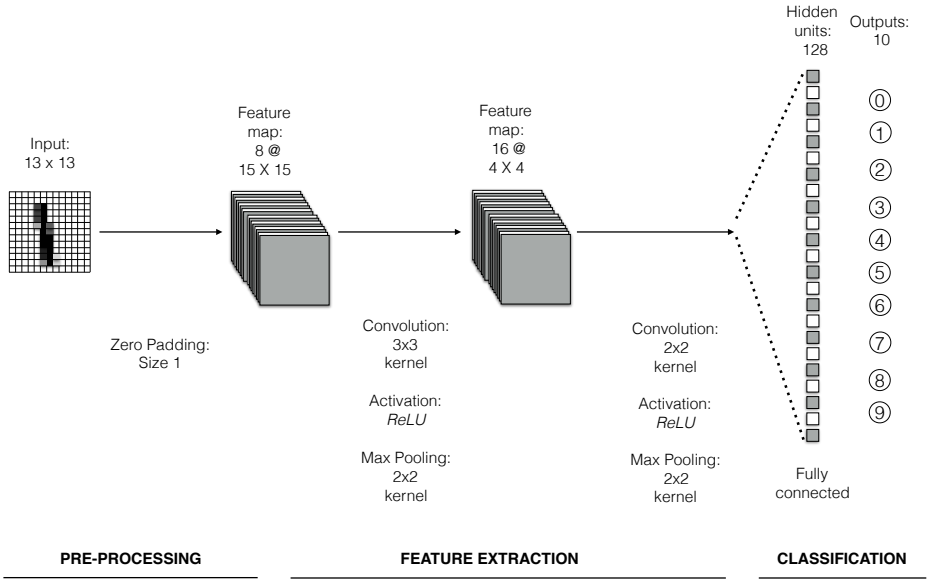



existing trained model and tuning them to the new database. Transfer learning allows the model to adapt information previously learned to reach an acceptable accuracy rate with fewer data and in a shorter span of time.

Our transfer learning uses the Modified National Institute of Standards and Technology's (MNIST) database of handwritten digits (LeCun et al., 1989). This is one of the seminal databases on visual recognition, and it includes a training set of 70,000 examples of digits written by about 250 writers. The digits in the MNIST dataset are perfectly centered white digits on plain black backgrounds, without stains, blobs, or inconsistencies. Since our data is similar to the MNIST database, it is recommended to freeze the first convolutional layer and allow training for the rest of the components in the network.[11] Using the MNIST dataset as a baseline will help the CNN to find the most core features of every digit. On the other hand, the training process will allow our model to adapt the weights from the MNIST to our noisier data examples, which are surrounded by stains, guiding boxes and pencil marks.

[11] Ananthram, Aditya. 2018. "Deep Learning for Beginners Using Transfer Learning in Keras." Towards Data Science. https://towardsdatascience.com/keras-transfer-learning-for-beginners-6c9b8b7143e (accessed March 21, 2019).

Figure 8: Network Architecture



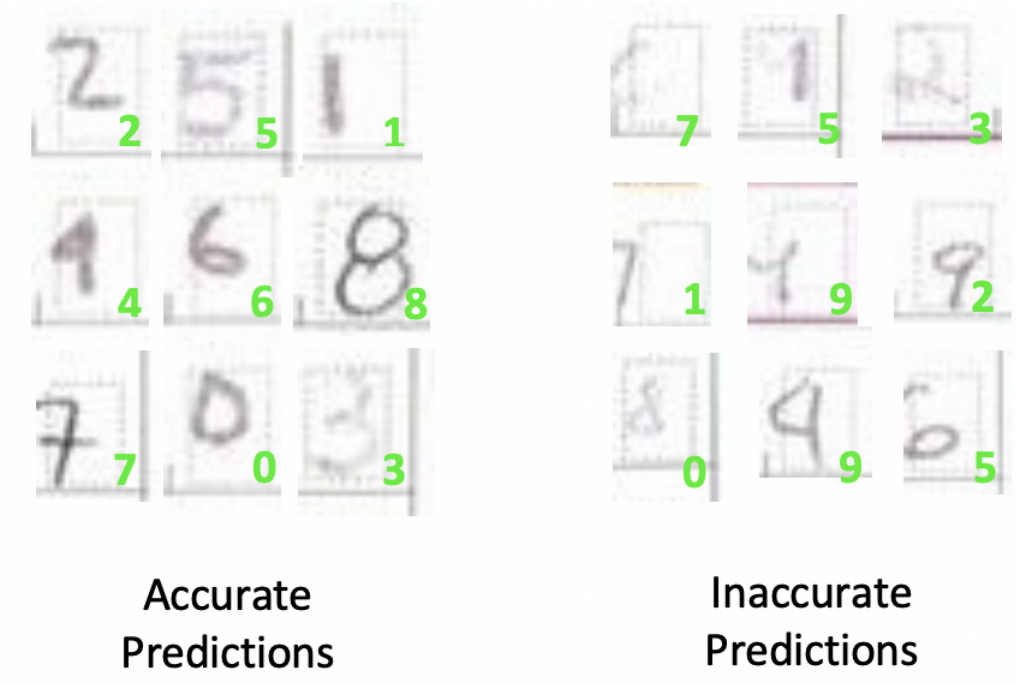

*Notes*: Figure 8 illustrates the CNN structure applied to identify digit numbers. The inputs of the images consist of numerical arrays of 28 (height) $\times$ 28 (width) pixel values. The network contains two convoluted layers of 16 and 32 filters, respectively.

Our new training sample for the subsequent layers of the model consists of 26,271 labeled digits from our tallies, and a testing sample of 2,616 digits. To address the quality issues of the digits in our tallies we implement *data augmentation*. This technique creates random variations of the existing training images by, for example, flipping, flopping, rotating, zooming them out, or combining all of these alternatives (Chatfield et al., 2014). The random transformations will force the model to pay less attention to the specific location of a feature on an image and instead grasp its relationship to other image features. For our example, every time a training image passes through the network it will be (1) rotated within a [-15,15] degrees range and (2) zoomed in or out no more than 20% of the original image size.

### 3.0.1 Prediction of electoral results and trouble-shooting

The accuracy of this model on the testing dataset with actual digits from the tallies is 96.46%. Figure 9 shows some of the classifications that we perform on the tallies with the CNN. While the CNN reach accurate predictions for most of the numbers, it makes a few mistakes implausible to be human-made. In most cases, however, the errors are due to an inaccurate registration of the official results on the tallies. For example, the right panel of Figure 9 demonstrates that numbers written outside the guiding box are likely to be misclassified. The pressure of the handwriting also affects the accuracy of the predictions. If the digits are almost illegible, the model pools other elements of the image like the background square to make a prediction. Other mistakes, as in the case of the 4 classified as a 9, are due to handwriting styles and shared features between numbers. These examples spotlight the importance of the post-classification process to understand the sources of errors, identify problematic cases, and conduct parameter tuning. Such process requires humans actively finding out the sources of error in the model and correcting the most important mistakes. This stage needs to make clear that using CNNs to classify images should not eliminate human intervention, but limit it to the most crucial or controversial decisions.

Figure 9: Examples of predictions



To check for the ultimate consequences of these errors on the estimated vote totals, we calculate the results for each party in the congressional district #15 in Mexico City. We chose this district because the quality of the scanning was high, allowing us to conduct a more careful analysis of the sources of errors and misclassification. The vote totals per party in this district range from 0 to 176 with 50% of the vote counts equal or lower than 12. If we consider that some of the small parties and coalitions did not obtain a single vote in certain polling stations, and that there is a cap on the number of people allowed to vote in each station, then we end up with a right-skewed distribution of vote counts (see Figure 12 in the Appendix showing the density and quartiles of the distribution).

Figure 10 presents the comparison of detected and real vote counts in the tallies of District 15. Because of the right skewed distribution, we applied a logarithmic transformation to both the predicted and real vote counts. Each point in the plot represents the comparison between the predicted vote counts of each of the parties (including null votes, non-registered candidates, and coalitions) and the actual votes. The size of the point indicates the frequency of each potential combination.

Notice that we also added to the plot information about the quality of the predictions of the digits. Recall that the last layer of the CNN, the *softmax* layer, outputs a list with the probabilities that each input digit has of belonging to each of the 10 possible outcomes (0-9). To classify the number, we take the category with the highest probability of the list. For most of these numbers, the maximum probabilities are pretty high (above 0.99). However, in cases where the number is ambiguous, or the model does not have enough information (e.g. the digits in the tally are not legible), the predictions that the CNN makes are less likely to be accurate. Therefore, we created an indicator for each vote count registered in each tally that we then use to evaluate its overall quality.

Most of the vote counts are composed of three digits (including leading zeros), and each of these digits has a probability of being one of the 10 potential numbers. We created a weighted average of these probabilities as an indicator of the uncertainty around each vote count. The reason to use a weighted mean is the following: errors are more costly when they happened in the hundreds position of a number than in the units. For example, the bias in a vote count prediction is higher if we confuse a "9" with a "4" when registering a "931" than when the number is "139".

In the former we underestimate the vote count by 500, whereas in the latter only by 5.[12] Once we computed the weighted mean of all the vote counts per tally, we label those with a mean equal or below 0.9 as "moderate quality" and those above it as "high quality."

Thus, the triangles in Figure 10 show the vote counts in the tallies identified as "moderate quality", whereas the blue circles show the "high quality" ones. If the CNN is yielding accurate predictions, we should see a high density of observations concentrated along the 45 degree dashed line indicating that the prediction and the official vote counts are equal. We indeed observe a dense distribution of a large number of observations along the red line. This is especially true for the high quality tallies: very few deviate from the line. The "moderate quality" observations show greater deviations, but these do not follow a pattern that would suggest a systematic bias.

To assess the implication of these deviations in the final vote distribution, we compare the proportions in vote counts between the official information and our predictions. This helps with a more substantive interpretation of how accurate the results are, and can provide evidence regarding the randomness of the error. Each of the dashed lines in Figure 11 represents a party competing in District 15. The symbols indicate the proportion of the votes that each party achieved in the district according to different sources: the official results (circles), and the results using our CNN on a) all tallies (crosses) and only "high quality" tallies (triangles). As the plot shows, the CNN recovers similar proportions to those of the official results and this performance improves when using high quality tallies. Overall, the method is able to correctly identify the ranking and magnitude of vote counts. This illustrates the applicability of CNNs and their power for data collection tasks.

## 4 Suggestions and warnings

Just like any other method or tool designed to predict outcomes, CNNs face challenges and limitations. In some cases, these roadblocks require additional steps to ensure the quality of results. In other cases, they demonstrate the limits of CNNs when it comes to performing complex tasks. In this section we provide a list of both practical and technical issues to consider when training and running a CNN for classification purposes. Further, we also discuss some of the shortcomings of CNNs with respect to their scope, interpretability and validation.

[12] The weights for the hundred, ten and unit positions are 0.5, 0.35 and 0.15 respectively.

Figure 10: Number of votes registered in tallies: Official vs. Predicted

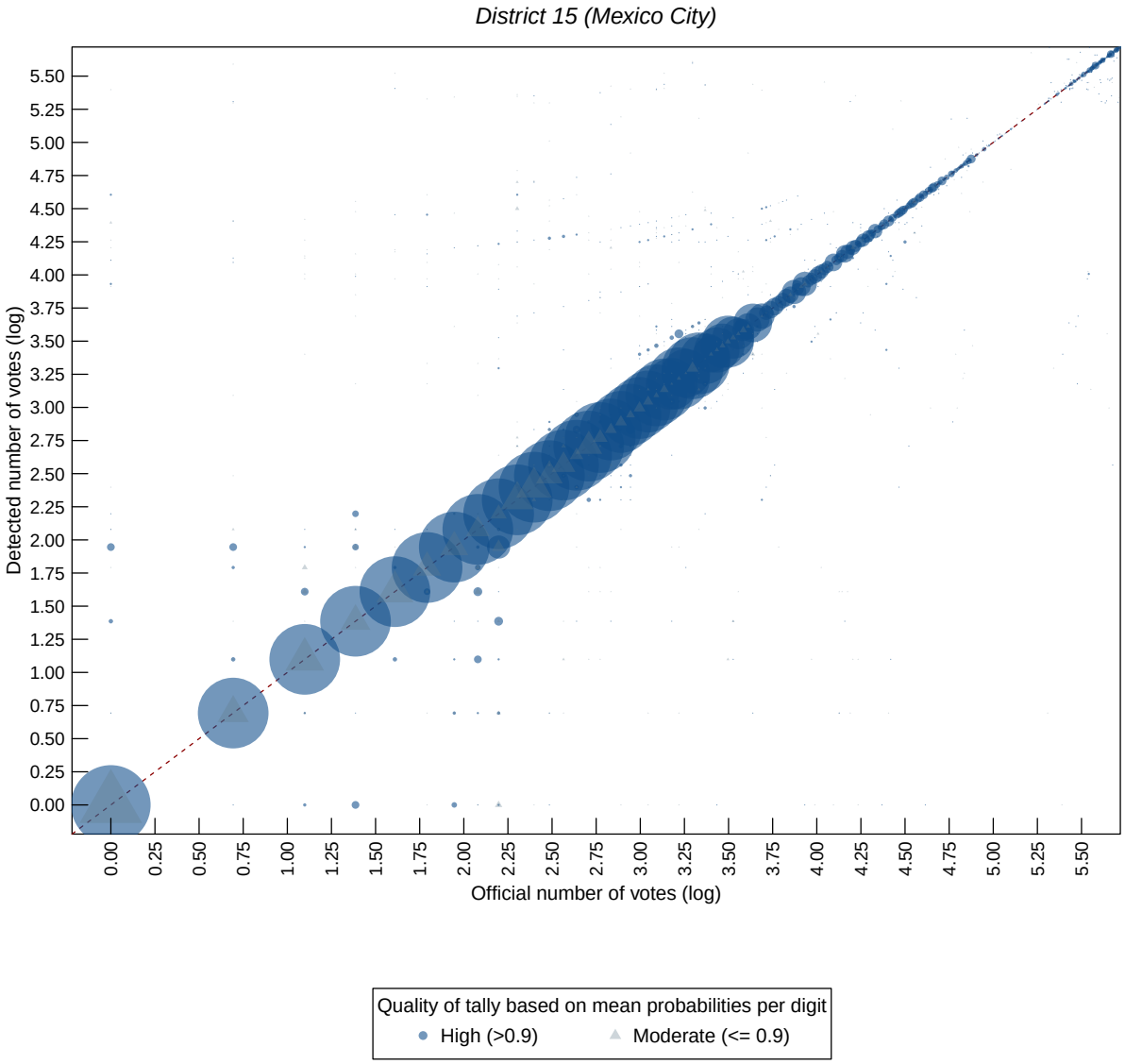
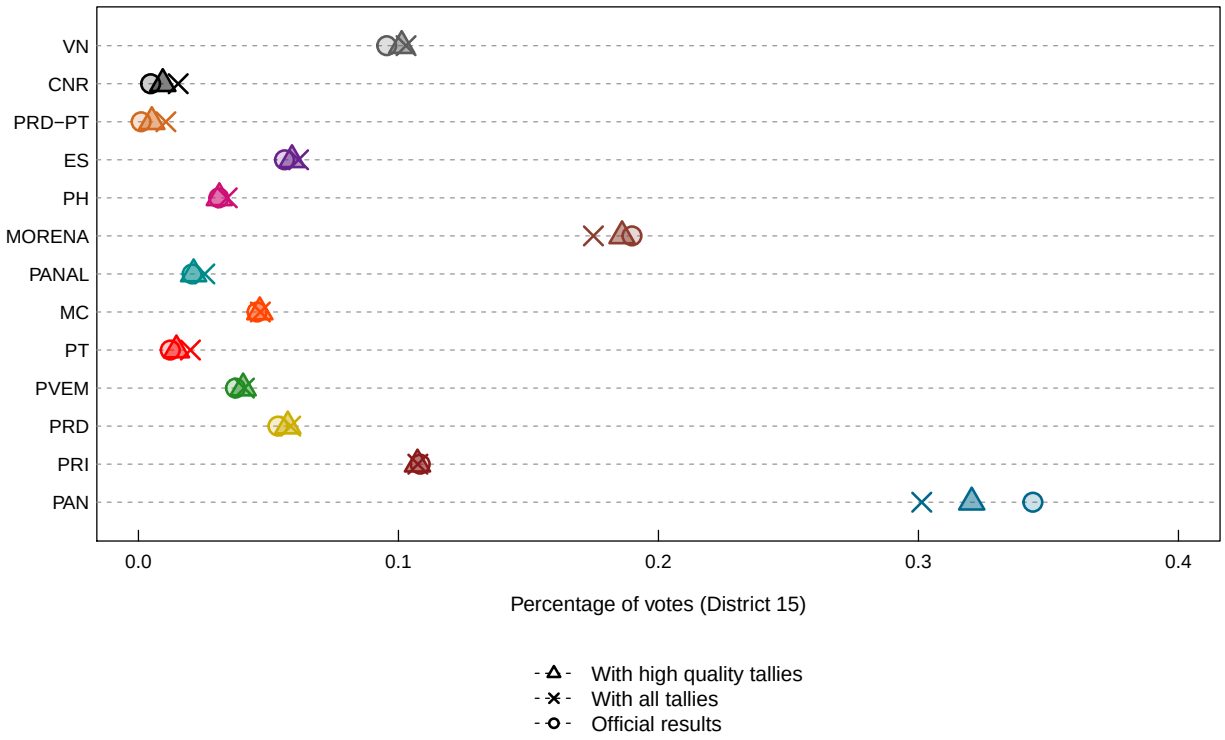


Figure 11: Vote proportions by party in District 15: Official vs. Predicted




## 4.1 Recommendations

### 4.1.1 Deciding when enough training is enough

As with any human-performed task, prediction accuracy for a CNN model comes only with practice. Every time an image passes through the network, the model reduces its classification error by calibrating the importance it gives to each filter. The more opportunities for the model to review an image, the more accurate it becomes at classifying the examples in the training set.

However, training the model for too many iterations will eventually lead to overfitting. This occurs when the model "memorizes" the features of every training image in a very detailed way, decreasing its ability to generalize its predictions outside the training examples. A way to avoid both under-training and overfitting a model requires tracking the performance of the model after every epoch and comparing the loss values in the training and validity sets. In principle, the reported loss values for both sets should decrease with the number of epochs. Overfitting takes place when the validity loss starts increasing even when the training loss continues decreasing.

This situation dictates the moment to stop training the model and start evaluating its overall performance on the validity set.

We suggest three ways to prevent overfitting while training the model. The first one is to increase the number of training images. Having a larger training set allows the model to focus not on the random fluctuations of a few examples, but rather on the visual features that appear repeatedly across the images. In other words, it forces the model to look for patterns that are more likely to appear outside the training set. The second suggestion is helpful when we cannot collect new training examples. In this case, it is recommended to artificially expand the training set with data augmentation, as argued in Section 3. A final solution involves a technique called dropout (Srivastava et al., 2014). As its name suggests, this technique "drops out" a random set of neuron activations before being transferred to the next layer. By ignoring some information units during a forward or backward pass, we increase the opportunities for the model to learn more robust features that activate multiple neurons. Dropout also expands the number of training iterations required to overfit the model.

#### 4.1.2 Optimize your training set

**Active learning** We recommend picking the most useful instances of each class to train the model (Settles, 2009). This suggestion is particularly convenient when obtaining additional training examples is a difficult, time-consuming, or expensive task. Selecting those examples should be based on two goals: informativeness (i.e., how much the instances help the classifier to improve its performance) and representativeness (i.e., how well the instances represent the overall input patterns of the entire dataset). Both are rarely achieved simultaneously, and researchers must often choose which one to prioritize at the cost of the other (Huang, Jin and Zhou, 2014).

**Class balance** It is also useful to make sure that all classes in the training set are represented by a similar number of examples (Buda, Maki and Mazurowski, 2018). Class balance prevents the skewing the model's predictions toward the label with more training instances (Japkowicz and Stepehn, 2002). This is a recurrent issue in situations where the positive cases represent a minority of all the cases, such as locating oil-spills (Kubat, Holte and Matwin, 1998) or identifying fraudulent bank operations (Chan and Stolf, 1998).

**Image cleaning** Another risk when training a model is that it may learn visual features that are alien to those defining the categories of interest (e.g., ink stains that are not relevant to the content of a document). To prevent this problem, images should be pre-processed to make sure they appear as similar as possible. This step may require modifying and cropping irrelevant parts of the image.

"Denoising" the images involves multiple procedures such as RGB conversion, histogram equalization, and normalization. RGB conversion converts a color image to a grayscale one and reduces its dimensionality. This alternative can be useful when, for example, the background color of an image correlates with each output label. Histogram equalization improves the contrast in images. It accomplishes this by stretching out the intensity range of the image, increasing the local contrast and enhancing the definitions of edges in each region of a picture. Normalization scales all of the images into the same pixel range.

It is also possible to homogenize the data of every training batch. In this case, *batch normalization* transforms the outputs of the convolutional layers to parameters with zero mean/unit variance, allowing the layer activations to be appropriately handled by any optimization method (Ioffe and Szagedy, 2015). This technique keeps the network from focusing on outlying activations that decelerate its learning.

#### 4.1.3 Validate and check the results

An insightful way to improve the model is to review the misclassified images in the validity set. In our case, this exercise allowed us to find the problems of our model in those instances where the digit-number images include some of the printed tally features. We also found out that the model is less accurate when identifying those numbers that show a soft pressure in the handwriting. These insights will help us include more training images with these characteristics in the next stage of our project.

Further, as in other fields involving prediction and measurement, validation is key to achieving better, stronger results. Validating the results will reveal potential sources of errors and provide information about the model fit. We also suggest another type of validation that involves the assessment and visualization of the components of CNNs. Zeiler and Fergus (2014); Won,

Steinert-Threlkeld and Joo (2017) provide a series of tools to identify and visualize the most relevant features driving the predictions (a process analogous to finding the most reliable and important coefficients in a regression). These tools provide information not only about the mechanisms behind the predictions, but also about the structure and composition of the data, like salient features or common patterns among them. For example, some of these tools provide "maps" that reveal what parts of the image or specific features were most determinant in reaching a prediction and would then provide the researcher with information to complement or tune the samples included in the training pool. A good example of this is a case where the goal was to identify and tag "sheep" in images. Although the CNNs were doing an optimal job at finding sheep in open fields, the accuracy decreased drastically when the picture featured a human holding a sheep. A more profound analysis of the situation evidenced that the features helping the algorithm to reach a conclusion of whether there were sheep in the pictures were related to the characteristics of the field (green, grass, pasture) rather than to the actual features of the animals. Therefore, the examination of the outputs improves our understanding of the data and the mechanisms behind predictions.

### 4.2 Warnings: The limits of CNNs and deep learning

Throughout this article, we present the functioning and components of CNNs, as well as an illustration of their applicability to data collection for social science purposes. The examples show that CNNs are powerful tools to automatize the extraction of information from large pools of images in an efficient, fast and reliable way. However, we have also illustrated some of the challenges and complications that arise in even relatively straightforward, textbook cases, such as handwriting detection. While it is possible to ameliorate the most common technical issues, it is also important to acknowledge the limitations of CNNs to complete certain tasks.

We mention some of the most important flaws for CNNs when solving complex tasks. First, unlike humans, CNNs do not account for the pose and orientation of objects in a picture. Ideally, we would like to emulate how humans process visual content: identify an object regardless of its size or whether it is rotated. However, CNNs focus only on routing the pixel information to the neurons in charge of detecting features without adding any information about their relative

position and orientation. Similarly, the neurons ignore information about the location and surroundings of those features. For example, if a CNN is in charge of detecting faces and it identifies features associated with eyes, a nose and a mouth, then it does not matter if the eyes are below the nose and above the mouth, as in an abstract puzzle representation; the CNN would still predict a face in the image. The implication of this process is that without a comprehensive and extensive training dataset, the classification of pictures becomes inaccurate and subject to error, even in cases involving simple tasks (Sabour, Frosst and Hinton, 2017).

Another criticism of CNNs lies in their lack of uncertainty measures. Unlike traditional models like regression, these tools do not yield quantities like standard errors that aid with inferences or assessments of confidence. While the last layers of the CNN provide the "probabilities" of an image belonging to a certain class, these quantities should be used with caution. Recent evidence shows that seemingly imperceptible alterations to an image can cause drastic changes in the outcome probabilities (Nguyen, Yosinski and Clune, 2015). This research implies that the likelihood for an image to be identified might depend not only on its basic features, but also on stochastic aspects, such as its illumination or the proportion of a picture it occupies.

Finally, CNNs cannot discover dimensions that humans themselves cannot identify. The aforementioned limitations of this methodology should warn us about applying CNNs to discover and measure latent dimensions in data or for the classification and scaling of abstract concepts. Recall that some crucial parts of information that give context to a visual message, such as surroundings and positions, get lost during the classification process. Similar to what occurs in text analysis, if a human cannot code or validate a trait, a CNN will not be able to do it, either.

In summary, researchers need to be aware of the limitations of CNNs, cautious about the objectives they expect CNNs to fulfill, fully knowledgeable of the data under analysis and training, and careful about their interpretation of the outputs of the CNN.

## 5 Conclusion

Using computer vision techniques for image-retrieval and classification can extend the scope of the data, theory and implications of several social phenomena. In this paper we presented a comprehensive guide for researchers interested in using Convolutional Neural Networks for visual

content coding and classification. We presented the intuition behind CNNs, highlighted their potential, and described their structure and implementation.

CNNs have a wide variety of applications in multiple fields of the social sciences. They can be applied to similar data collection problems like the one outlined in the text: retrieving signatures or the votes that were whipped for a given policy registered in historic documents, classifying written notes, or even the extraction and interpretation of symbols. They can also be applied to the coding of more complex political phenomena: measuring gender composition in pictures of groups, identifying the sentiment of material from electoral campaigns, recording the activities of crowds in a protest, counting the number of people waiting to vote in polling stations, etc. The extraction of information from images and visual content invites a wider variety of questions.

The present article illustrated the benefits of CNNs for data collection purposes and image classification by focusing on the recognition of handwritten tallies from the 2015 Mexican election. It also presented some of the challenges and practical issues that researchers should consider when dealing with this type of data. We concluded by discussing the strengths and limitations of CNNs.

The study of new data sources both complements and enhances the knowledge that we already have about the political world. However, these opportunities should be paired with a deep understanding of the characteristics, mechanisms, and consequences of these models.

## References


Atkeson, Lonna Rae and Kyle L. Saunders. 2008. Election Administration and Voter Confidence. In *Democracy in the States: Experiments in Election Reform*, ed. Bruce E. Cain, Todd Donovan and Caroline Tolbert. The Brookings Institution pp. 21–34.

Barberá, Pablo. 2015. "Birds of the Same Feather Tweet Together. Bayesian Ideal Point Estimation Using Twitter Data." *Political Analysis* 23(1):76–91.

Bengio, Yoshua. 2012. "Practical recommendations for gradient-based training of deep architectures." *CoRR* abs/1206.5533.
**URL:** *https://dblp.org/rec/bib/journals/corr/abs-1206-5533*

Bowler, Shaun, Thomas Brunell, Todd Donovan and Paul Gronke. 2015. "Election Administration and Perceptions of Fair Elections." *Electoral Studies* 38(1):1–9.

Buda, Mateusz, Atsuto Maki and Maciej A Mazurowski. 2018. "A systematic study of the class imbalance problemin convolutional neural networks." *Neural Networks* (106):249–259.

Buduma, Nikhil. 2017. *Fundamentals of Deep Learning*. O'Reilly Media.

Buduma, Nikhil and Nicholas Locascio. 2017. *Fundamentals of deep learning: Designing next-generation machine intelligence algorithms*. " O'Reilly Media, Inc.".

Cantú, Francisco. 2018. "The Fingerprints of Fraud: Evidence from Mexico's 1988 Presidential Election." Working Paper.

Casas, Andreu and Nora Webb Williams. Forthcoming. "Images that Matter: Online Protests and the Mobilizing Role of Pictures." *Political Research Quarterly* .

Challú, Cristian, Enrique Seira and Alberto Simpser. 2018. "The Quality of Vote Tallies: Causes and Consequences." Working Paper.

Chan, Philip K and Salvatore J Stolf. 1998. Toward scalable learning with non-uniform classand cost distributions: A case study in credit card fraud detection. In *KDD 1998*.

Chatfield, Ken, Karen Simonyan, Andrea Vedaldi and Andrew Zisserman. 2014. "Return of the Devil in the Details: Delving Deep into Convolutional Nets." *eprint arXiv:1405.3531* .

Dietrich, Bryce J., Ryan D. Enos and Maya Sen. 2019. "Emotional Arousal Predicts Voting on the U.S. Supreme Court." *Political Analysis* pp. 1–7.

Dietrich, Bryce, Matthew Hayes and Diana O'Brian. 2019. "Pitch perfect: Vocal pitch and the emotional intensity of congressional speech on women." Working Paper.

Druckman, James N. and Michael Parkin. 2005. "The Impact of Media Bias: How Editorial Slant Affects Voters." *The Journal of Politics* 67(4):1030–1049.

Goodfellow, Ian, Yoshua Bengio and Aaron Courville. 2016. *Deep Learning*. Cambridge, MA: MIT Press.

Homola, Jonathan. 2018. "The Political Consequences of Group-Based Identities." Working Paper.

Huang, Sheng-Jun, Rong Jin and Zhi-Hua Zhou. 2014. "Active Learning by Querying Informative and Representative Examples." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36(10):1936–1949.

Huff, Connor D. 2018. "Why Rebels Reject Peace." Working Paper.

Ioffe, Sergey and Christian Szagedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Technical report arXiv:1502.03167.

Japkowicz, Nathalie and Shaju Stepehn. 2002. "The Class Imbalance Problem: A Systematic Study." *Intelligent Data Analysis* 6(5):429–449.

Kubat, Miroslav, Robert C. Holte and Stan Matwin. 1998. "Machine learning for the detection ofoil spills in satellite radar images." *Machine Learning* 30(2-3):195–215.

Lawson, Chappell and James A. McCann. 2004. "Television News, Mexico's 2000 Elections and Media Effects in Emerging Democracies." *British Journal of Political Science* 35:1–30.

LeCun, Yann, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard and Lawrence D Jackel. 1989. "Backpropagation applied to handwritten zip code recognition." *Neural computation* 1(4):541–551.

LeCun, Yann and Yoshua Bengio. 2003. Convolutional networks for images, speech, and time series. In *The handbook of brain theory and neural networks*, ed. Michael Arbib. 2 ed. Cambridge, MA: pp. 276–280.

Lipton, Zachary C. 2016. The Mythos of Model Interpretability. In *2016 ICML Workshop on Human Interpretability in MachineLearning (WHI 2016)*. New York: .

Lucas, Christopher. 2018. "Neural Networks for the Social Sciences." Working Paper.

Makin, David A., Dale W. Willits, Wendy Koslicki, Rachael Brooks, Bryce J. Dietrich and Rachel L. Bailey. Forthcoming. "Contextual Determinants of Observed Negative Emotional States in Police-Community Interactions." *Criminal Justice and Behavior* .

Masters, Dominic and Carlo Luschi. 2018. "Revisiting Small Batch Training for Deep Neural Networks." *CoRR* abs/1804.07612.
**URL:** *https://dblp.org/rec/bib/journals/corr/abs-1804-07612*

McCarty, Nolan, Keith T. Poole and Howard Rosenthal. 2006. *Polarized America*. The MIT Press.

Nair, Vinod and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *ICML'10 Proceedings of the 27th International Conference on International Conference on Machine Learning*, ed. Johannes Fürnkranz and Thorsten Joachims. pp. 807–814.

Nguyen, Anh, Jason Yosinski and Jeff Clune. 2015. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pp. 427–436.

Pastor, Robert A. 1999. "The Role of Electoral Administration in Democratic Transitions: Implications for Policy and Research." *Democratization* 6(4):1–27.

Qin, Zhuwei, Fuxun Yu, Chenchen Liu and Xiang Chen. 2018. "How Convolutional Neural Networks See the World — A Survey of Convolutional Neural Network Visualization Methods." *Mathematical Foundations of Computing* 1(2):149–180.

Rumelhart, David E., Geoffrey E. Hinton and Ronald J. Williams. 1988. "Learning representations by back-propagating errors." *Cognitive Modeling* 5(3).

Sabour, Sara, Nicholas Frosst and Geoffrey E Hinton. 2017. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*. pp. 3856–3866.

Schrodt, Philip A. 2004. "Patterns, rules and learning: Computational models of international behavior." Working Paper.

Settles, Burr. 2009. Active Learning Literature Survey. Computer Sciences Technical Report 1648 University of Wisconsin–Madison.

Simonyan, Karen and Andrew Zisserman. 2014. "Very deep convolutional networks for large-scale image recognition." *arXiv* .

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.

Torres, Michelle. 2018. "Framing a Protest: Determinants and Effects of Visual Frames." Working Paper.

Williams, Nora Webb. 2019. "Automated Image Taggers from Amazon, Google, and Microsoft: Are They Useful for Social Science Research." Working Paper.

Won, Donghyeon, Zachary C Steinert-Threlkeld and Jungseock Joo. 2017. Protest activity detection and perceived violence estimation from social media images. In *Proceedings of the 25th ACM international conference on Multimedia*. ACM pp. 786–794.

Zeiler, Matthew D and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *European conference on computer vision*. Springer pp. 818–833.

# 6 Appendix

## Glossary

**activation function** Function that allows to generate non-linear outputs. In the context of CNNs, these are mathematical rules or functions that transform the elements of a matrix. 4

**activation layer** Step of the process where the activation functions are applied to the output matrixes that result from the convolutions. 9

**backpropagation** Long series of nested equations that have the objective of adjusting each weight in the network in proportion to how much it contributes to overall error. Backpropagation can be seen as an application the Chain rule to find the derivatives of a function with respect to any variable in the nested equation. 11

**batch normalization** Technique for improving the performance and stability of a neural network via a normalization step that fixes the means and variances of layer inputs. The normalization process occurs in "mini-batches" (e.g. subsets of the training dataset), to make the process more efficient. This is possible given that 1) the optimized loss over a mini-batch is an actual estimate of that in the full set whose quality improves as the size of the batch increases, and 2) takes advantage of parallel computation. For a layer with $d$-dimensional input $\mathbf{x} = (x^{(1)} \dots x^{(d)})$, we normalize each dimension with $\hat{x}^{(k)} = \frac{x^(k) - \mathbf{E}(x^{(k)})}{\sqrt{\text{Var}(x^{(k)})}}$, where the expectation and variance are computed over the training dataset. 23

**batch size** Number of images in a match, or subset of training images. 12

**convolution** In mathematics, a convolutional operation transforms two existing functions into an argument defining how the shape of one functions is modified by the other. The term in computer science extends to the idea of combining the values of a neuron with those from the different regions of the image. 7

**data augmentation** A technique to increase the size of the training dataset by reproducing the original training examples with a random perturbations on the image, such as rotations, flips, or zooms of the input. 16

**Deep belief networks** A series of Restricted Boltzmann Machines (RBMs, type of neural networks) in which the output of one RBM feeds into another one yielding a "stack" of nets. The main input of these networks are raw pixel intensities. Unlike feedforward networks where data only moves forward, in RBMs the connections are not directed and therefore the information can flow in both visible-hidden and hidden-visible directions. The hidden layers of an RBM can be used as a form of "feature vector".. 4

**epochs** A training iteration consisting on the single pass of the entire training database throughout the model. 12

**feature map** The matrix mapping the outputs from convolution of a given filter and the different regions of an image. 8

**filter size** Product of height and width, in pixels, of a matrix representing a filter. 7

**filter stride** Number of pixels that a filter slides through an image. 7

**filters** In a CNN, the filters represent the neurons of the network. These are matrixes of numbers representing patterns and combinations of pixels that permit the extraction of features of an image. The pixel combinations can represent edges, corners, blobs, color combinations, and textures. Filters are convolved with regions of the image to create feature maps that represent the prevalence of the patterns they represent in an image. 6

**gradient descent** Optimization algorithm used to to update the weights, or coefficients, of our model. Its objective is to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient. The gradient is built with the partial derivatives of the function with respect to its different parameters. It is represented by $x^{'} = x - \epsilon \Delta x f(x)$. 11

**layer depth** Number of filters used in a layer. 7

**learning rate** A positive scalar that defines the magnitude of the steps in which the gradient descends. Formally, the learning rate is defined as the parameter $\epsilon$ in the gradient descent function (see gradient descent). 12

**mini-batches** Subsets of the images in the training dataset used in the batch-normalization process. 12

**neurons** In the context of a Neural Network, the neuron is a mathematical function, like a sum, that transforms an input to produce a new output. When talking about Convolutional Neural Networks, the neuron is the filter, or kernel, that convolves with different areas of the image. 4

**pooling** Data reduction technique that applies a rule (e.g. keep the maximum) of a given quadrant of the matrix to retain the most important and salient information and improve computational efficiency and speed. 9

**receptive field** The area where a given filter, or neuron, is positioned to execute a convolution. 7

**ReLU** The name stands for REctified Linear Unit. It is the most commonly used activation function in CNNs formally defined as $y = max(0, x)$. It is computationally cheap due to its mathematical simplicity, converges faster due to the linearity for positive values and its sparsely activated given that it is zero for negative values. 9

**Sigmoid** Activation function defining a "S"-shaped curve, or sigmoid, formally defined as the inverse logit: $\frac{1}{1+e^{-x}}$. Useful when dealing with binary outcomes/labels. The function is differentiable and monotonic, and can cause a network to get stuck when training. 9

**softmax layer** A layer with a multinomial function embedded that transforms the output of the CNN layers up to that into probabilities that the input belongs to each of the potential labels. This is a fully-connected layer because its neurons are not independent and the output is based on this dependency (i.e. the probabilities summing to 1). 11

**Tahn** Also known as hyperbolic tangent, it is an activation function also with a sigmoidal shape but with a range between -1 and 1. Its formal definition is $\frac{22}{1+e^{-x}}$ implying that negative inputs are mapped strongly negative, and zero values would be near to that value when mapped. 9

**transfer learning** Exploiting a model trained in a particular setting to improve the generalization of the findings of a different setting. This is a valuable resource when the researcher considers that the factors that explain the variations of the original database are useful for the goal of the new database (Goodfellow, Bengio and Courville, 2016, p. 526-527). 14

**weights** The unknown parameters of the neural network that seek to improve the fit between the model and the data. In CNNs the weights are the numbers contained in the feature matrixes. 4

**zero-padding** A padding is a "frame" that we add to the border of an image to allow the convolution of the edges and corners of an image, and increase the information that is processed through the CNN. In this case, the zero-padding adds a vector of zeros with the length of the width of the image above and below it, and another vector of zeros with the length of the height of the image to the left and right of it. This is equivalent to adding a black frame of width 1 px to the image. 6

### Backpropagation

Suppose that a neuron $j$ in the last layer provides a classification outcome $y_j$.[13] To estimate the prediction error, the model compares such an outcome with the target label, $t_j$. In our digit recognition example, the prediction error of the neuron for the outcome "1" is the difference between the true outcome and the model's estimated probability for the image to belong to that category. After adding up the prediction error of all the neurons in the layer, $E = \frac{1}{2}\sum_{j\in 10}(t_j - y_j)^2$, we can estimate the error function derivative of the last layer:

$$\frac{\partial E}{\partial y_j} = -(t_j - y_j) \tag{2}$$

Similarly, we can express the error derivatives in terms of the logit of the neuron, $z_j$:

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j} = y_j(1-y_j)\frac{\partial E}{\partial y_j} \tag{3}$$

To minimize this error term, the network goes back to its prior layers and identifies those weights contributing the most to this error. In other words, it estimates how the neuron outcomes in layer $i$ affect the outputs of layer $j$ given the weighted connection between both layers, $w_{ij}$ :

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial y_i} = \sum_j w_{ij}\frac{\partial E}{\partial z_j} = \sum_j w_{ij}y_j(1-y_j)\frac{\partial E}{\partial y_j} \tag{4}$$

These partial derivatives allow us to estimate the contribution of a specific weight to the error term:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}}\frac{\partial E}{\partial z_j} = y_j(1-y_j)\frac{\partial E}{\partial y_i} \tag{5}$$

[13]The explanation and notation of this example come from Buduma (2017).

The partial derivative in Equation 5 allows the model to gradually modify its weights after reviewing a set $k$ of examples from the database $K$:

$$-\Delta w_{ij} = -\sum_{k \in K} y_i^{(k)} y_j^{(k)} (1 - y_j^{(k)}) \frac{\partial E^{(k)}}{\partial y_i^{(k)}} \tag{6}$$

**Distribution of votes per party in District 15**

Figure 12: Distribution of votes per party