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

Michelle Torres
Rice University
smtorres@rice.edu

Francisco Cantú
University of Houston
fcantu10@uh.edu

Abstract

We provide an introduction of the functioning, implementation, and challenges of Convolutional Neural Networks (CNNs) to classify visual information in social sciences. This tool can help scholars reduce the resources necessary for the tedious task of classifying images and extracting information from them. We illustrate the implementation and impact of this methodology by using two relevant examples: coding handwritten information and identifying those elements in newspaper images that craft the tone of a particular political message. 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 their useful comments. All errors are our own. Replication materials will be available in the websites/Github pages of the authors.

1 Introduction

Over the most recent decades, computers have increasingly taken over tedious and repetitive tasks. 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), and social media posts (Barberá, 2015).

And yet, despite computers' effectiveness in analyzing textual or numeric information, their performance in classifying images was disappointing until a few years ago. In principle, we could ask a computer to find, for example, all images of a specific political leader if we provide specific rules describing the physical characteristics of such 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 chief limitation. The directives we give to a computer could be insufficient when identifying 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 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, and textures across images. This learning process allows the computer to identify and track an object under various conditions.

This paper introduces to political science the use of Convolutional Neural Networks (CNNs) 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 general guidelines on how CNNs work and to discuss the

challenges and problems that researchers might encounter when using this tool.

We illustrate the advantages of this methodology with two examples in which we code relevant information. First, we use CNNs to code handwritten information. In particular, we code the handwritten election results on the vote tallies for Mexico's 2015 federal election. The low complexity of this set of images of interest (i.e., handwritten numbers) makes this case ideal for illustrating the structure and functioning of CNNs. It allows us to: 1) illustrate the implementation of the CNNs in an intuitive and reliable way, and 2) conduct an objective test of the performance of CNNs. The application may be of interest to recent works collecting archive data with, for example, military (Huff, 2018), demographic (Lladós et al., 2007; Coüasnon, Camillerapp and Leplumey, 2007), or electoral information (Homola, 2018; Cantú, 2018).

The second example highlights the applicability of CNNs to the analysis of the framing of political events using images. Analyzing image content in this fashion can expand research on topics like the visual tone of campaign coverage (Lawson and McCann, 2004; Druckman and Parkin, 2005), the portrayals of protests (Webb Williams, Casas and Wilkerson, Forthcoming; Torres, 2018; Won, Steinert-Threlkeld and Joo, 2017; Wuhs, 2014; Zhang and Pan, 2019), and the camera-recorded interactions between police officers and citizens (Makin et al., Forthcoming). For our example, we collected the front pages of 450 U.S. newspapers in the aftermath of the El Paso shooting on August 3, 2019, and study the different visual frames used to communicate gun-violence-related news. This exercise shows that, when reporting the same event, newspapers are more likely to show pictures with heavily armed police in states where gun popularity is high, and vice versa in states where such popularity is low.

While our hope is to encourage scholars to use the methodology, we also want to stress its limitations. 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. We emphasize the importance of establishing transparent goals when using the model, restricting its application to tasks that could be performed, in principle, by a human, and avoid post-hoc interpretations of the outcomes (Lipton, 2016).

We also acknowledge that the method 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 article thus translates some of the terms applicable to CNNs into concepts known to social scientists. The Appendix includes a glossary of these terms, which we identify with italics in the text.

The manuscript is organized as follows. First, we introduce what CNNs are and explain each stage in the process. Next, we list a few recommendations when applying these tools, as well as discuss the limitations of CNNs for certain measurement and classification tasks. Then, we illustrate the practicability of this tool for the two examples described above. We conclude by suggesting potential ways to expand the use of visual analysis in social science.

2 A Primer on Convolutional Neural Networks (CNNs)

Figure 1: Example of a Convolutional Neural Network Structure

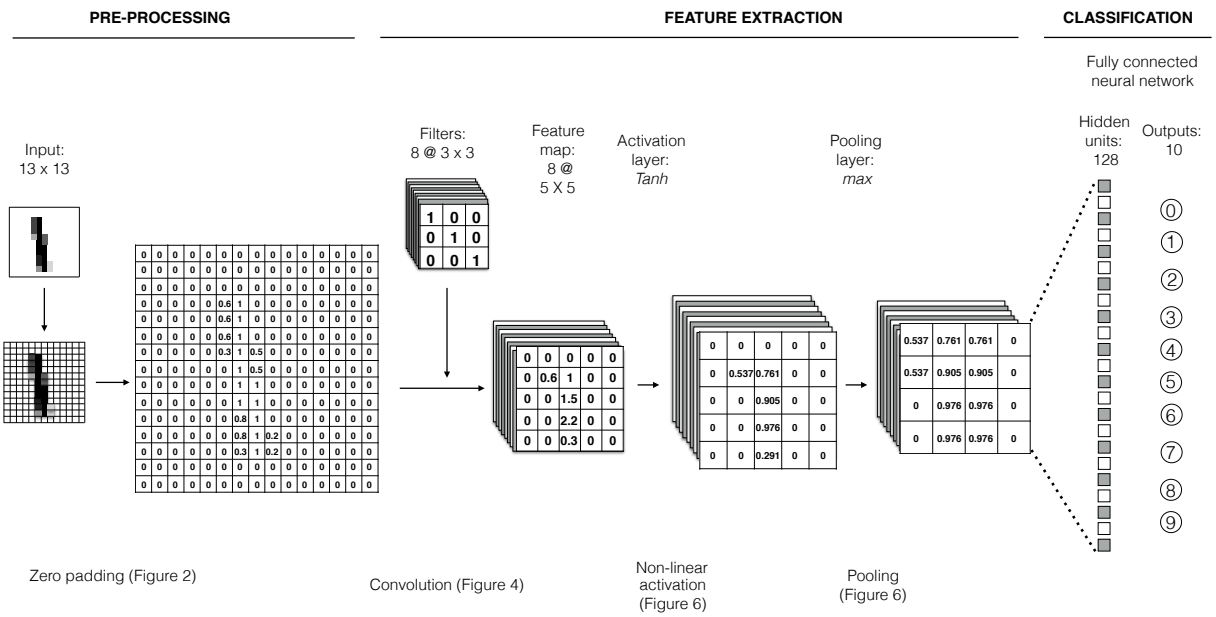


Figure 1 illustrates the basic architecture of a CNN. It is organized into layers, or sets of *neurons* representing specific visual features that can be found in an image such as edges, blobs, textures or colors. Each neuron convolves with sub-regions of the original image to search for similar visual patterns. This structure 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.

The functionality of CNNs can be divided into three stages. 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 learning and classification stage uses the image's components to categotize the image into one of the available categories. Each stage requires the researcher to specify a few *hyperparameters* to define the structure of the network and the training process. We explain below the logic behind each stage and the corresponding decisions available to the researcher.

2.1 Image Pre-Processing

We begin by preparing the raw data for the analysis, making sure that images share the same shape, size, and contrast range. In other words, this pre-processing step transforms all files so they have the same arrangement of pixels. When the images vary in size or orientation, a standard practice is to convert all images into squares of the same size. It can be done by squashing the image's largest side, adding black bordering to its shortest one, or center-cropping it. The latter is the most common practice because it also reduces the number of pixels that the model will process, leading to faster learning.

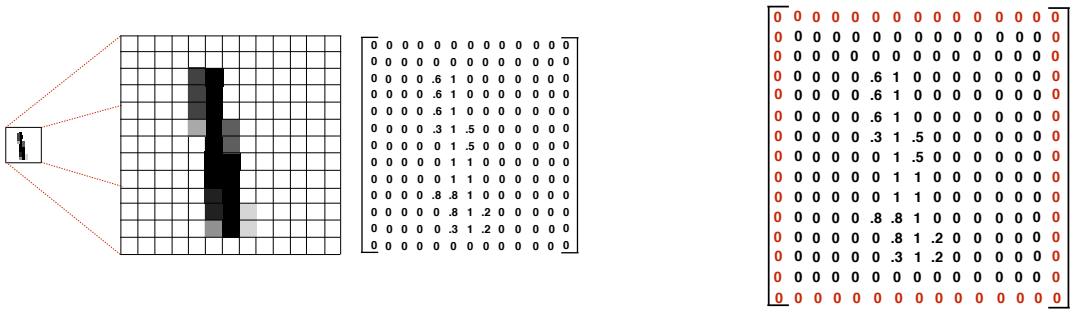
We also scale the pixel values to make sure the model will not be biased towards images with large-value pixels. A common scaling approach constrains the pixels to the $[0, 1]$ range after dividing the raw values by 255, the largest possible pixel value. Additionally, we can divide the difference between each pixel and the mean pixel value in the image by the standard deviation of the pixel values in the data. This operation normalizes the pixel values to have a mean of zero and a variance of one. Depending on the data, color images can also be transformed to grayscale to reduce their dimensionality.

Once the images share the same specifications, we need to transform them in a format that the computer can understand. This process involves representing each image as a numerical array, where each entry depicts a specific pixel value. Figure 2(a), for example, illustrates how a 13

(height) $\times$ 13 (width) pixels picture showing a handwritten "1" can be transformed into a matrix of $13 \times 13 = 169$ units, each of them specifying the light intensity of a specific pixel.[1] In 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) in which the values in each cell represent the intensity of the corresponding color.

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 downplay the features along the edges of the image. 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 size of the numerical array to 15×15.

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



(a) Image Transformation

(b) Zero Padding

Finally, we dispatch all of the images in our database into three subsets: 1) one that will fit the model, 2) another one that will evaluate its performance, and 3) a final one that will be labeled after training the model. In other words, we work with three datasets: train, validation, and test data. The train data include those examples that the model uses to learn the patterns corresponding

[1]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 only for illustrative purposes, we take higher numbers in the matrixes presented as higher concentrations of "ink". Therefore, higher numbers correspond to darker pixels.

to each outcome category. The validation dataset includes those examples that help us check the generalizability of the model's predictions. The model is only allowed to observe and predict the labels of the validity data, but it is not allowed to learn from it. Finally, the test set includes data from unlabeled examples and is never used during the training stage. The whole point of a supervised learning is to train a model that helps us label the examples in the test set.

To assign the images to the train and validity subsets, we can employ *k-fold cross validation*, which randomly assigns the images into k groups, or folds. The first fold is for validation and the $k-1$ remaining folds for training. The process is repeated k times, each of them using a different fold for validation and training on the rest of the images. Every iteration is independent, so learning is not transferred across iterations (Hastie, Tibshirani and Friedman, 2009, 242-245).

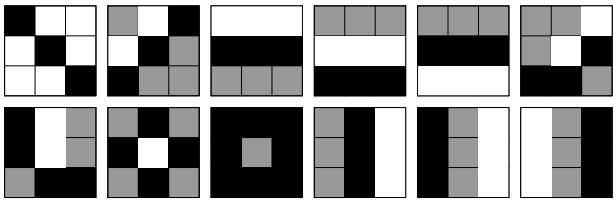
A common rule of thumb is to randomly split the database into 70% for training and validation and 30% for testing (Elkan, 2012). However, the ratio can be adjusted to the specific problem under consideration. Models with just a few hyperparameters are faster to train and may not require a very large training set. Similarly, a huge database makes it possible to train the model using a smaller share of images. When labeling the data implies a very difficult, time-consuming, or expensive task, we can use pre-trained models and then "curate" the training examples that the CNN will see according to our specific research objectives. This process is called active learning, and we describe it on Section 3.

2.2 Feature Extraction

The second stage of the CNN decomposes the image into single components. This process consists of calculating the dot product of sub-regions of the input matrix with a set of smaller matrices, called *filters*, each representing a particular visual feature. The filters in the first layer include basic representations of features, such as straight lines (see Figure 3 as an illustration). Subsequent layers build up on those features and transition to more complex features, 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 it will recognize (Qin et al., 2018).

Within each layer, filters slide across the width and height of the input matrix, computing the dot product of the filter and a particular image's area. This dot product is the *convolution* part

Figure 3: Examples of filters



Note: This figure displays examples of filters of size 3 (height)× 3(width) = 9 that were randomly initialized in the first layer of a CNN.

of the CNN. The output of this operation is a new matrix called *receptive field*. Intuitively, this convolution represents a measure of how well a given filter "matches" a region of the image. The entries in each filter matrix can be understood as the pixel intensities that altogether form relevant or informative patterns. We are interested in learning the relevant pixel combinations 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 hyperparameters for this operation. First, the *filter size* is the product of the filter's width and height. For example, Figure 3 presents simplified examples of filters of size 9. 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. 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 that is preserved during the convolution.[2] Finally, the *layer depth* defines the number of filters, or features to be searched for in the layer. Figure 3, for instance, represents the filters of a layer with depth 12.

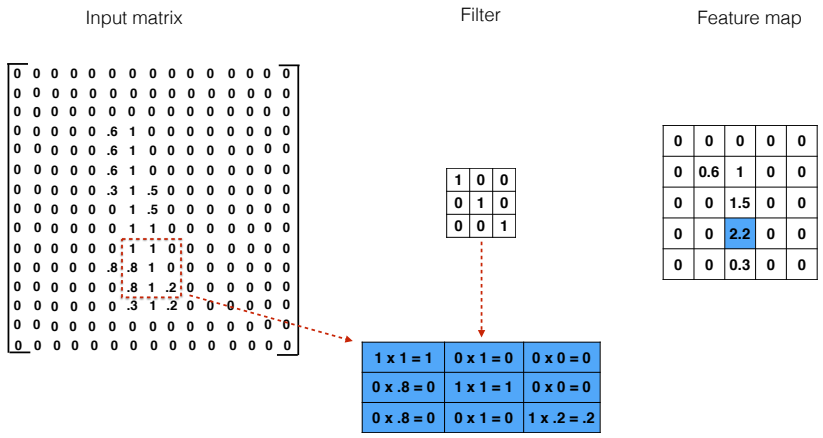
We illustrate how a convolution process works in Figure 4. In this example, we take the top-left filter displayed in Figure 3 and use a stride of 3. The convolution process then involves computing the dot products of the filter and the values of every equivalent pixel space in the image. In this example, 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

[2]For a comparison of model performances using different strides and filter size, see Simonyan and Zisserman (2014).

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

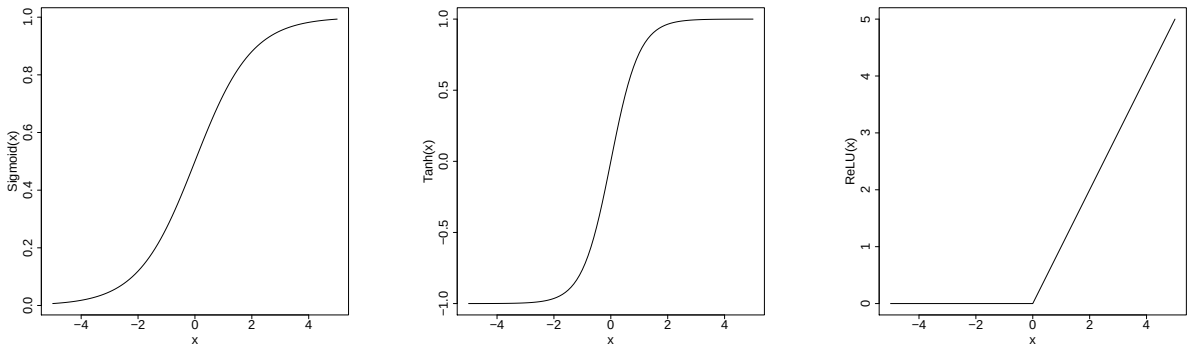


Since the resulting feature map is a linear transformation of the input matrix, adding more convolutional layers at this point would be redundant; the result could be obtained with a single linear product. Such feature maps are unlikely to produce smooth gradients, a necessary input for the learning phase described in Section 2.3. To address this problem, it is necessary to include an *activation layer*. This layer applies a non-linear transformation to the feature maps before being sent to the next convolution layer. The non-linearity property allows CNNs to stack multiple layers and extract more information from the image.

The specific transformation of the input depends on the *activation function* of the neuron. Figure

5 shows three examples of these functions. 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*, a linear transformation of *Sigmoid* that zero-centers the outputs and bounds them to the $[-1, 1]$ interval.[3] A limitation of both functions is that their output becomes flat near their 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).

Figure 5: Examples of Activation Functions



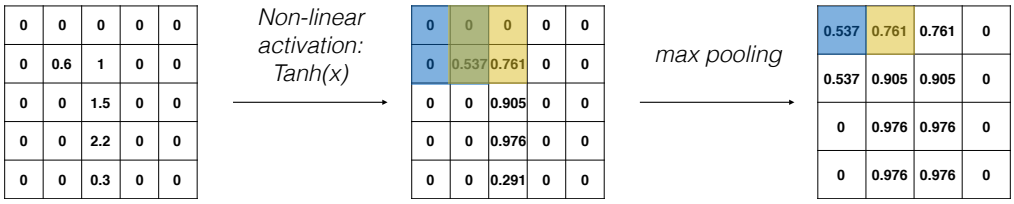
(a) $Sigmoid(x) = \frac{1}{1+e^{-x}}$ (b) $Tanh(x) = \frac{2}{1+e^{-2x}}$ (c) $ReLU(x) = \begin{cases} 0 & \text{if } x < 0, \\ x & \text{otherwise.} \end{cases}$

The following step reduces the dimensionality of the activation map by using a *pooling* layer. A pooling layer shrinks the size of the matrix while keeping the most important 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. In the example of Figure 6, 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 a feature.

The process is repeated for each of the filters in the first layer. The resultant feature maps then

[3]For illustration purposes, we use *Tahn* for our example in Figure 6.

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



become 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

The last stage of the CNN uses the elements extracted during the convolution to predict the label of the image. Such prediction is performed by a neural network, which consists of a set of inter-related neurons. Neurons here are also organized into layers. The input layer receives the features extracted during the convolution. The output layer produces the output values. Any layer in-between the input and output layers is called a hidden layer.

Similar to the feature extraction stage, each neuron receives an input, transforms it, and sends a signal to other neuron. The strength of the signal depends on the *weight* between the emitting and receiving neurons. Positive weights amplify the signals and highlight their contribution to the output. By contrast, negative weights weaken the signal to which they are attached.

When the information reaches the output layer, the nodes deliver the probability that the original input belongs to each specific label. If the classification task is only between two labels, the output layer estimates the probabilities using an inverse logit. Otherwise, the probabilities are estimated through a *softmax layer* layer, 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.

So far, the model has performed only a *forward propagation*—i.e., passing the input data sequentially through the hidden layers to the output layer to generate a prediction. However, for the model to learn what features of the image are more likely to belong to each label, it requires a process called *backpropagation* (Rumelhart, Hinton and Williams, 1988). In this case, the model goes back throughout the hidden layers towards the input layer. With every step away from the output layer, the model calibrates the weights between neurons to gradually minimize the errors in predictions (Schrodt, 2004; Lucas, 2018*a*). Therefore, the learning process for a CNN consists in reviewing a set of labeled examples multiple times to find the optimal combination of feature maps and weights that minimize the disparities between true and predicted labels.

Analytically, backpropagation searches for the local points in a high-dimensional space where the error derivative of the total error function is zero. To find those saddle points, the CNN modifies the weights between layers and checks for its resulting changes on the error. This estimation is called *gradient descent* and consists of repeatedly calculating the slope of each weight with respect to the loss function and modifying such weight to minimize the slope until finally reaching the bottom of the function.[4]

While these repeated estimations can be automatically calculated, we need to specify a few hyperparameters of the operation. First, it is necessary to define the number of *epochs*, or the number of times that all training examples pass through the network. Given the iterative nature of the gradient descent, the network is more likely to fit its weights as it has more opportunities to review the examples of the train data. It is expected that the train and validity loss i.e., the prediction errors in the train and validity data will decrease with the number of epochs. When the validity loss stops decreasing, we should stop the training process to avoid overfitting or allowing the model to "memorize" the training images without making generalizable predictions.[5]

Second, we need to define the *batch size*, which is the number of training images that need to pass through the network before tuning-up 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. However, 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,

[4] A more technical description of the backpropagation process is included in the Glossary.

[5] See Subsection 3.2 for a few recommendations on how to delay this issue during the training phase.

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. Though updating the model with every example decreases the risk of getting stuck on a flat region, it produces a very noisy signal. The sweet spot between both extremes splits the training data into *mini-batches*, allowing the model to update its parameters several times during an epoch. A common approach is to set the batch size to 32 (Bengio, 2012; Masters and Luschi, 2018). The number of mini-batches in our database multiplied by the number of epochs tells us the number of *iteration*, or how many times the gradient was updated during the training phase.

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. By contrast, a very small learning rate is more likely to find a local minimum, but it will take a long time to converge. A good practice is to start with a large learning rate and gradually decrease it at every given number of iterations (Buduma, 2017). This strategy allows the gradient to start exploring across the entire hyper-parameter space and gradually make smaller jumps to approach the global minimum. There is a variety of optimizers that adaptively tune the learning rates for all parameters in the model.[6]

2.4 Transfer learning

Training the model from scratch can be computationally expensive and requires a large dataset to avoid overfitting. An alternative approach involves *transfer learning*, where an already trained model can be directly applied to a new task or used as the starting point for training a new model (Pan et al., 2010). Social scientists have taken advantage of the benefits of transfer learning. This technique then repurposes the information learned by a model for a different yet related goal. How much information we can transfer between models depends on the size of our pool of images, as well as on how similar our data and classification tasks are to those of the base model. If the researcher is lucky enough to find a CNN already trained on a dataset with similar content to

[6]For a very helpful comparison of the most common optimizers, see Karpathy, Andrej. 2019. "CS231n Convolutional Neural Networks for Visual Recognition." https://cs231n.github.io/neural-networks-3/ (March 30, 2019).

hers, and that also performs a similar classification task, she can retrain the model by "freezing" all but the last layers of the network. When we freeze a layer it becomes inactive; the new input images do not go through it. This process allows the new model not only to find the more generic features of the images in an easier way by using what it learned from the original training process, but also to focus its training on the more specific details of the new target categories. As the overlap between the datasets decreases, it is advised to extend the backpropagation to earlier layers. For example, Zhang and Pan (2019) used transfer learning to fine tune a CNN that allows them to identify collective action events. Their base model was a VGGNet, a CNN trained on a set of 1.2 million images classified into 1,000 categories (mostly common objects). Because those categories do not cover human faces or crowds, crucial elements of protests and demonstrations, the authors "froze" the first 12 layers of a VGGNet and fine-tuned and re-trained the last 4 with a set of images containing their target elements. After this, the new, modified net was able to find the probability that a given image depicts a protest. See Webb Williams, Casas and Wilkerson (Forthcoming) for a detailed explanation on the method, as well as the resources available when using this approach.

3 Recommendations and warnings

3.1 Software

There are multiple open-source machine learning packages that researchers can use to design and run a Convolutional Neural Network. Some of the most popular are `TensorFlow` (from Google), `Caffe` (from UC Berkeley), `CNTK` (from Microsoft), `PyTorch` (from Facebook), and `MXNet` (supported by AWS). `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 packages like `TensorFlow`.

The differences between the aforementioned packages are in terms of speed, energy efficiency, and accessibility. As a quick summary, `TensorFlow` is faster when running large-scale models, while `Caffe` is faster with small-scale ones (assuming they are both implemented on a CPU-based platform). Further, while `PyTorch` is more memory efficient than its counterparts, `MXNet` requires the least amount of computational energy. Thus, scholars should consider the size and complexity of their data and classification objectives, plus the hardware/computational resources they have at

hand. Zhang, Wang and Shi (2018) offer a great compilation of insights regarding the performance of these packages with respect to speed, memory and energy.

In particular, for simpler cases with smaller datasets, we opt to use `Keras` (see Application 1 below). It is not only designed to enable fast, small-scale experimentation, but given its popularity, there are also several resources and support materials online to guide its use. However, if the tasks are more demanding and larger datasets are under analysis, then `TensorFlow` and `PyTorch` are preferred alternatives (see Application 2 below). While the former offers great functionality and is particular useful in object detection tasks, the latter offers shorter training durations and easier ways of debugging.[7]

Beyond these packages, there are other tools that facilitate the design of the architecture of a CNN and also allow researchers to take advantage of pre-trained models. In particular, these might be interesting for deep learning beginners or scholars with less advanced programming skills. The options include but are not limited to *Amazon AWS Machine Learning Training*, *Google Cloud AutoML* or *Google Cloud Machine Learning Engine*. For a review of these platforms and their performance, see Webb Williams (2019).

3.2 Finding the training 'sweet spot'

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: a decreased ability of the model to make generalizable predictions outside the training data. We suggest four ways to prevent overfitting while training the model.

Grid search your model hyperparameters As mentioned several times above, the specific hyperparameters of the model depend on the data and classification goal. The challenge when looking for these specifications is that the values of a feature, say the number of layers in the model,

[7] Sayantini. 2019. "Keras vs TensorFlow vs PyTorch : Comparison of the Deep Learning Frameworks." `https://www.edureka.co/blog/keras-vs-tensorflow-vs-pytorch/` (September 22, 2019).

might depend on the values of other features, such as the number of filters in the layers or the learning rate. A way to approach this problem involves using a grid search. This technique runs the model in a for loop, compiling it with a different set of hyperparameters in each iteration. The results allows the researcher to compare the performance of multiple settings and to choose the configuration with the lowest loss.[8]

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, increasing the numbers of training examples forces the model to look for patterns that are more likely to appear in the test set.

Data augmentation This suggestion is helpful when we cannot collect new training examples. It produces random variations of the original training images by, for example, flipping, flopping, rotating, zooming out, or combining all of these alternatives (Chatfield et al., 2014). The augmented cases will force the model to pay less attention to the specific location and orientation of a feature on an image and instead grasp its relationship to other image features.

Dropout As its name suggests, this technique "drops out" a random set of neuron activations before being transferred to the next layer (Srivastava et al., 2014). 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.

3.3 Optimize your training set

A better learning depends not only on the number of times the model can learn from the examples, but also on the information it can extract from them. We suggest three things to keep in mind when choosing the amount and quantity of data that the model will learn from.

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 train-

[8] See Webb Williams, Casas and Wilkerson (Forthcoming) for a detailed discussion of how this technique works.

ing 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). As described above, images should be pre-processed to make sure they appear as similar as possible. In some cases, this step may require modifying and cropping irrelevant parts of the image.

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.

3.4 Validate and check the results

Similar to textual analysis, a key principle of machine-coded visual analysis is "validate, validate, validate" (Grimmer and Stewart, 2013, p. 269). Validating the results will reveal potential sources of errors and provide information about the model fit. An insightful way to improve the model is to review the misclassified images in the validity set. As we will show in the examples below, this is a helpful practice to find potential problems in the model. Another way to check the validity

of the model's predictions is to visualize the most relevant features driving the predictions (Zeiler and Fergus, 2014; Won, Steinert-Threlkeld and Joo, 2017). Similar to finding the most important coefficients in a regression, this exercise provides information about the the mechanisms behind the predictions. 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.

4 A warning note: The limits of CNNs

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 discuss some of the shortcomings of CNNs with respect to their scope, interpretability and validation.

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 below illustrate the usefulness of CNN to extract information from large pools of images in an efficient, fast, and reliable way. At the same time, these examples also acknowledge the limitations of CNNs when it comes to completing certain tasks. Below we mention some of the most important flaws for CNNs that researchers must keep in mind when solving complex tasks with this tool.

First, CNNs do not account for the orientation of objects in a picture. Ideally, we would like CNNs to identify an object regardless of its size or rotation. However, CNNs focus only on routing the pixel information throughout the layers without adding any information about the relative position of the extracted features. As a result, a CNN would identify a face when it finds features associated with an eye, a nose, and a mouth, ignoring whether the eyes are below the nose and above the mouth, for example. 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 such as 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 or that are subject to interpretation. 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. Other visual messages are filtered through cognitive biases, experiences and backgrounds of the person consuming them. For example, it is hard to reach high levels of accuracy when classifying images according to the emotions they trigger or evoke (Casas and Webb Williams, 2019).

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. The first application on the registration of vote counts shows that mistakes happen even when we deal with data with low complexity, such as numbers. However, assessing the errors and identifying their source is easier given the factual nature of the data. Thus, abstract concepts or latent traits offer a hard case not only for the actual classification process, but also for the post-classification analysis and validation of the results.

5 Applications

We illustrate the usefulness of a CNN with two relevant applications in political science. The first one codes handwritten vote results from election tallies. This example allows us to discuss the advantages and challenges of analyzing handwritten information. The second example focuses on classifying newspaper pictures according to their visual elements. In this case, a CNN can help us identify the different types of messages that an image can convey.

5.1 Coding Electoral Results from Vote Tallies

Our first application of the CNNs codes handwritten information. This type of information is a common data source to scholars, and it includes archival data, signatures, annotations, and vote counts. For example, Huff (2018) uses the military archives of Ireland to identify individuals who participated in the 1916 Easter Rising in that country by looking at handwritten entries in the applications for military pensions; Lladós et al. (2007) and Coüasnon, Camillerapp and Leplumey (2007) build datasets with personal records extracted from historical registers of births and marriages, and Taylor (2008) reviews personal notes and hand-written letters from African-American women to understand the factors that shaped their lives and legacies.

In our case, 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, not to mention to increase the transparency of it. In the case of Mexico, capturing vote results with a CNN approach may significantly decrease accidental errors when adding up the votes, which actually occurs in almost 40% of the tallies in the country (Challú, Seira and Simpser, 2018). Moreover, this technology can shorten the time between the closing of the polls and the announcement of the results, a period of distress for candidates and voters in elections across the world.[9] We thus propose this tool as a way to increase not only the efficiency of the vote count, but also citizens' trust in the impartiality of the process (Atkeson and Saunders, 2008; Bowler et al., 2015; Pastor, 1999).

We select this case as a running example, given the simplicity of the data. The handwritten numbers have one color channel and a low number of features and variations. These factors allow us to represent the pixel intensities on the matrix in a straightforward way and track them throughout the pre-processing and different layers of the CNN (as shown in Figures 1, 2, 4, and 6). Moreover, the outcome labels are not sensitive to subjective interpretation; that is, we expect that

[9]See, for example, (*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); *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).

an "8" would always be classified as such regardless of the coder (except in those cases where the particular handwriting style complicates the reading of a digit). The objective nature of the data allows us to evaluate the performance of the CNN in a rigorous way.

Figure 7 shows an example of one of the tallies under analysis.[10] The first step involves extracting the handwritten numbers from the tally. Because the alignment, definition 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 that will in turn allow us to extract the table with the vote counts and subsequently "cut" the individual numbers from it.[11]

Figure 7: Example of the image of a tally

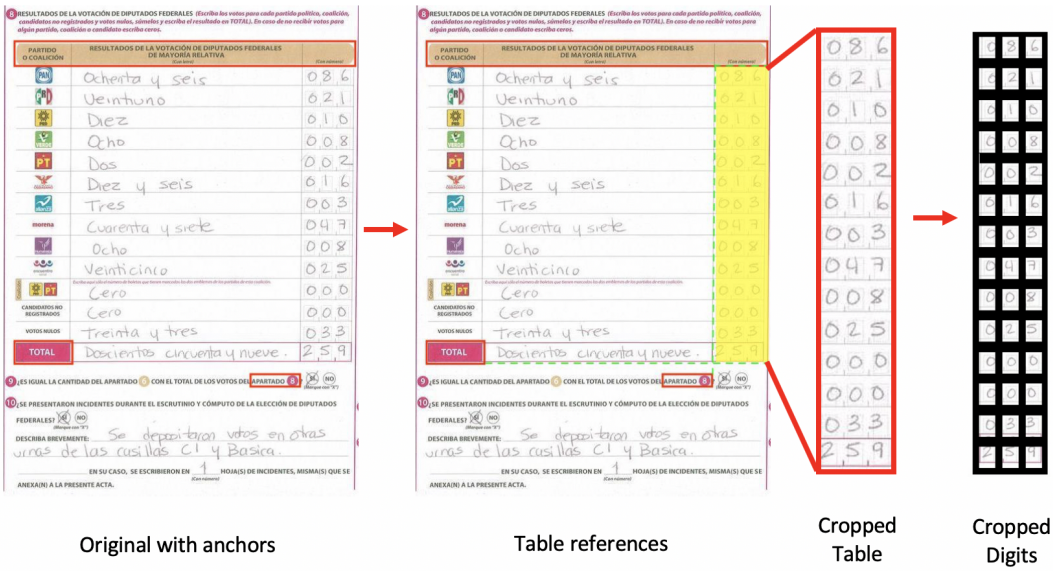


Figure 12 in the Appendix summarizes the architecture of the CNN that we use for the task of classifying each number. The network consists of two convolutional layers of 8 and 16 filters, respectively. Each convolutional layer is followed by a *ReLU* activation layer and a pooling layer. The convolutional outputs are sent to two fully connected layers and a terminal *softmax*.

We apply transfer learning using data from the Modified National Institute of Standards and Technology's (MNIST) database of handwritten digits (LeCun et al., 1989). This is one of the

[10] For the purposes of our example, we show only 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.

[11] We provide more information about this process in the Appendix.

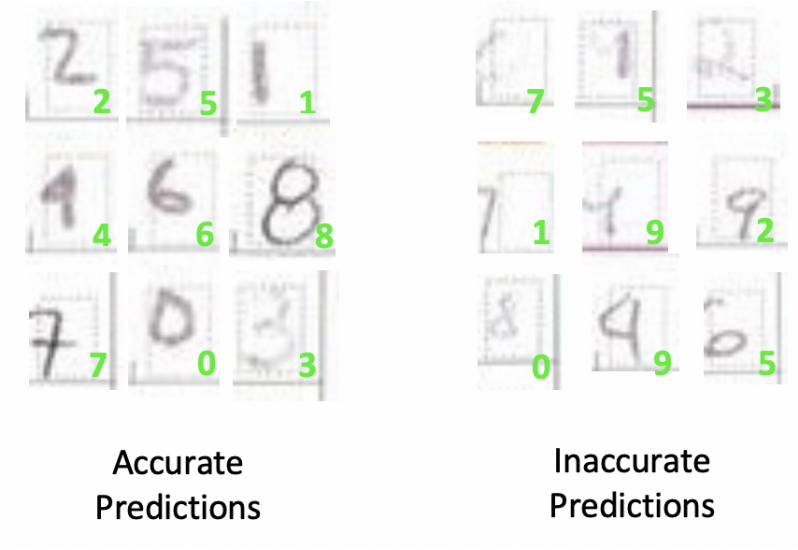
seminal databases on visual recognition, and it includes a training set of 70,000 examples of digits written by about 250 writers. The digits in the MNIST dataset are centered white digits on plain black backgrounds, without stains, blobs, or inconsistencies. We therefore train our model using the MNIST data and use the fitted model as a baseline for a second training process using our own database. We freeze the first convolutional layer, as explained in Section 2.4, and allow training for the rest of the components in the network. This re-training will allow our model to adapt the estimates (or weights) from the model trained on MNIST to our noisier data examples, which are surrounded by stains, guiding boxes and pencil marks. Our new training sample for the subsequent layers of the model consists of 26,271 labeled digits from our tallies, plus a validation sample of 2,616 digits.

The accuracy of this new model on the validation dataset with actual digits from the tallies is 96.46%. Figure 8 shows the classification that we perform of some of the digits in certain tallies with the CNN. While the CNN reaches accurate predictions for most of the numbers, it makes a few mistakes implausible to be human-based. 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 8 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. As explained in Section 3, these examples spotlight the importance of the post-classification process to understand the sources of errors, identify problematic cases, and conduct parameter tuning. Such a process requires humans actively finding out the sources of error in the model and correcting the most important mistakes. With this example we want to stress that using CNNs to classify images should not eliminate human intervention, but should rather limit it to the most crucial or controversial decisions.

To check the general validity of the model predictions, we calculate the parties' vote total in Mexico City's #15 Congressional district.[12] Since all the vote counts have three digits, including leading zeros, we estimate the "uncertainty" around a prediction with a weighted-average of the

[12]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.

Figure 8: Examples of digit predictions



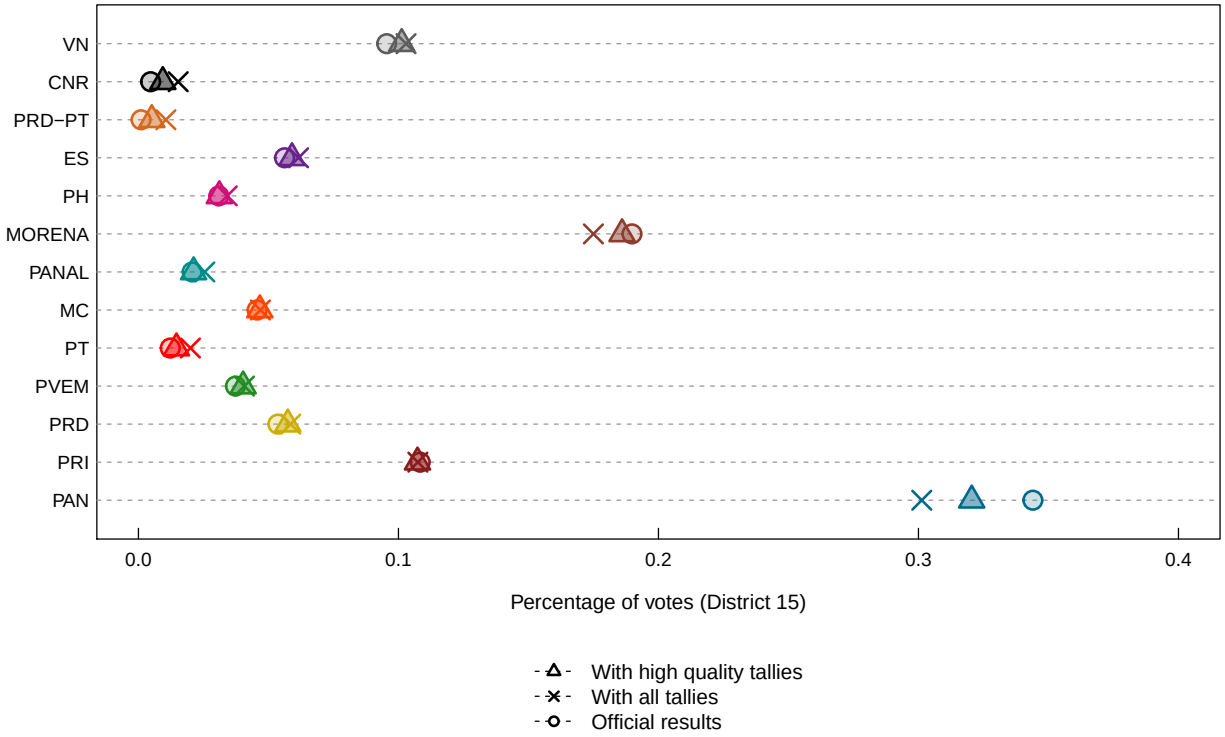
label probabilities for each digit in the vote count. This weighted mean assigns more importance to the errors that happen in the hundreds than in the tens or units. For example, if the model mislabels a "9" for a "4," the bias in the vote count would be larger when the vote count is "931" (underestimating the vote count by 500 votes) than when it is "139" (when the bias would be of only 5 votes).[13] Once we compute the weighted mean of all the vote counts per tally, we label those with a mean larger than 0.9 as "high quality."

Figure 9 compares the estimated vote shares for each party (crosses, for all tallies; triangles for "high quality" tallies) with those reported by the electoral authority (circles).[14]. As the plot shows, the CNN recovers proportions similar to the official ones, 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 accurate identification illustrates the applicability of CNNs and their power for data collection tasks.

[13]The weights for the probabilities for the hundred, ten and unit positions are 0.5, 0.35 and 0.15, respectively. Notice that these weights are simply giving more importance to some digits than others when computing the uncertainty of a full vote count. They should not be confused with the "weights" of the features that the CNN uses to optimize predictions and that we discussed above.

[14]In the Appendix we also present a comparison between predicted and observed vote counts at the tally level for a finer level of analysis.

Figure 9: Vote proportions by party in District 15: Official vs. Predicted



5.2 Classifying images from newspapers: Visual framing of a mass shooting

Our second example applies a CNN to classify the different ways in which news outlets frame political events. Media content impacts people's attitudes and behavior by setting the tone of an event or highlighting particular elements of a story (Iyengar, 1994; Valentino, Hutchings and White, 2002; Brader, 2005; Mutz, 2007; Dunaway, 2008; Abrajano and Singh, 2009; Abrajano, Hajnal and Hassell, 2017). In a world with increasing ideological polarization and the means to rapidly spread information, the use of CNNs to analyze media framing can help us manage the overwhelming amount of visual information available nowadays (Won, Steinert-Threlkeld and Joo, 2017; Torres, 2018; Farris and Silber Mohamed, 2018; Casas and Webb Williams, 2019). Further, automated methods like CNNs help us diminishing concerns about cognitive and ideological biases of coders (Vallone, Ross and Lepper, 1985; Ito et al., 1998; Ali et al., 2010). These biases are likely to be impactful given the nature of political data and prevalent even in those cases involving a small number of images.

Our example focuses on the ways in which newspapers frame visual information about mass

shootings and gun-related violence. In particular, we analyze the front pages of newspapers published on August 4, 2019, the day after a mass shooting in El Paso, Texas, where a gunman opened fire in a Walmart store. The event was featured in multiple newspapers across the nation, many of them reproducing the material from wire news sources like the *Associated Press*. While the content of the text could be quite similar across newspapers, we found more variance in the type of images published. As Figure 10 shows, some of the featured images include victims, relatives and employees. Others, in contrast, focus on law enforcement authorities and, in particular, the heavily armed special forces that responded to the attack.

Figure 10: Examples of newspaper covers from August 4, 2019

(a) *Sun Sentinel* (Ft. Lauderdale, FL)

(b) *Star Tribune* (Minneapolis, MN)

The actors on which a news article centers shape the framing that affects individuals' opinions on the issue. For example, Brader, Valentino and Suhay (2008) find that news about the costs of immigration trigger stronger opposition when Latino immigrants are featured instead of European immigrants. In the case of mass shootings, the spotlight on either the victims or heavily armed police helps to highlight the different angles of a traumatic event, and can also be used to evoke different emotions (Marcus, Neuman and MacKuen, 2000). On the one hand, pictures of the

victims communicate the consequences of gun violence by featuring its emotional consequences and focusing on sadness or charity. On the other hand, displays of heavily armed police highlight the criminal aspect of the attack and the actions taken to solve it, in this case through the use of weapons, and have the potential of triggering fear (Dardis et al., 2008; Boydstun, 2013; Boydstun and Glazier, 2013). We therefore estimate the proportion of newspapers featuring heavily armed police in the articles about the mass shooting in El Paso and suggest a few political variables that could be used for further analysis.

5.2.1 Designing and implementing a CNN: identifying pictures with heavily armed police

The images under analysis come from 450 front pages of local U.S. newspapers published on August 4, 2019. The newspaper covers are compiled by *Newseum*.[15] We find that 64.8% (292) of the newspapers featured at least one piece related to the shooting. Among these newspapers, 62.7% (183) included at least one picture illustrating the event. News pieces feature between one and four images, but most of them include only one. All of the pictures collected from the newspaper covers form our *test* dataset.

To classify the content of the images, we use a CNN with transfer learning from Won, Steinert-Threlkeld and Joo's (2017) model. These authors compiled the *UCLA Image Protest Dataset*, a novel database of 40,764 images from Twitter, from which they identify pictures of protests. The authors customize a CNN to annotate relevant objects and aspects of protests in the pictures, such as the level of violence and elements like "signs" or "police." These labels were generated by their customized CNN.[16]

We adapt the authors' model to our specific classification goal in two ways. First, we refined some of the original labels: instead of 17 outcomes, we are only interested in identifying whether a picture features *heavily armed* or *militarized* police. Second, as the example in Figure 10 shows,

[15]The decision to feature an event on the front page also provides information about what a certain publication considers important. Thus, focusing on front pages allows us to take that dimension into account. Further, under the assumption that even individuals who do not generally read newspapers are potentially exposed to front pages in public places like kiosks or stores, we consider that the content on the front page has a potentially larger effect on attitude formation.

[16]The architecture of this CNN is based on a 50-layer ResNet (He et al., 2016). This CNN is trained on more than a million images from the ImageNet database and can classify images into 1,000 categories. Thus, this pre-trained network has "learned" rich feature representations of a large number of images. Won, Steinert-Threlkeld and Joo (2017) made slight modifications to the architecture. More specifically, they changed the output layer of the network to their outcomes of interest (17 categories) and re-trained the last few layers with their own data also containing "negative" examples of protests.

some pictures feature Walmart employees wearing dark blue vests. This particular outfit shares features with the uniforms of regular police that can be found in the Won, Steinert-Threlkeld and Joo (2017) dataset. Since we anticipated misclassification errors from that source (e.g., Walmart employees classified as policemen), we re-trained Won, Steinert-Threlkeld and Joo's (2017) model with a dataset composed of 229 images from *Google images* and *Getty* with the tags "post-shooting victims," "heavily armed police," "special teams," and "Walmart employees." From these 229 images, we selected 152 to be part of the *training* data and 77 for the *validity* data. It is important to highlight that none of the images in these two datasets includes our target pictures from the newspaper covers.

From the validity set of 77 images, our model correctly classifies 92% of its elements.[17] Once applied to our testing dataset with images from the newspaper front pages, our model took less than 30 seconds to classify them.[18] After this exercise, we find that 17.3% of the pictures related to the shooting feature heavily armed police.

Understanding the factors that may influence the prevalence of pictures with militarized police is outside the scope of this paper. Nevertheless, we present a few bivariate relationships that may suggest further analyses in the future. Our variables include the newspaper's ideological slant (Gentzkow and Shapiro, 2010), the number of gun shows per 100,000 citizens in 2018,[19] and the gun death rate and strength of gun-control laws in the state.[20] Our outcome of interest is the proportion of pictures featuring heavily armed police and our unit of analysis is the state.

Figure 11 presents four scatter plots illustrating the relationship between the four variables enumerated above and the proportion of pictures published in a state's newspapers that feature heavily armed police. The upper left panel illustrates the relationship between gun popularity and proportion of images with heavily armed police. We add labels to those points with proportions of images different from zero. The x-axis shows the number of gun shows per 100,000 individuals that happened in each state in 2018. A simple correlation of these two variables of -0.168 suggests that, on average, as the number of gun shows increases, the proportion of images with militarized police decreases. The top right panel explores the relationship between strength of gun control

[17] The precision is 1.0, recall is .838, and the F-1 score is 0.91

[18] After manual exploration for evaluation purposes, we found only six misclassified images in the full sample.

[19] The list was compiled by the website http://www.gunshows-usa.com/2018-gun-knife-show-listings/.

[20] Giffords Law Center. "Annual Gun Law Scorecard." https://lawcenter.giffords.org/scorecard/. Accessed: September 17, 2019.

and images with police. The x-axis has a ranking of the strength of gun laws as defined by the Giffords Law Center, where 1 is the most restrictive state, and 50 is the least restrictive. Thus, the correlation of 0.075 and visual inspection suggest a positive relationship between the two factors: less restrictive states tend to have higher proportions of images showing police with guns. We find a similar positive relationship between the gun-related death rate and the proportion of images with militarized police in the bottom left panel (correlation=0.023) of the figure. The last panel shows a positive relationship between the ideological slant and the proportion of images with police. In other words, as the mean of the ideological slant of the newspapers in a state becomes more conservative, the proportion of images showing weapons and militarized police increases.

It is interesting to notice that although some of these relationships seem to be meaningful by visual inspection, the correlation coefficients we compute are not reliably different from zero. This result can be attributed to data constraints. Beyond the sample size, the data under analysis has an excess of zeros both structurally and as a regular count. In other words, a 0 ("No police") in the outcome variable is a result of either no display of heavily armed police in a picture (a regular zero count), or the lack of picture on the main cover of the newspaper (structural zero). However, a pool of images from not only the front page but the full body of the newspaper decreases the latter, helps to increase the sample size, and allows the researcher to conduct a more rigorous analysis of the topic.

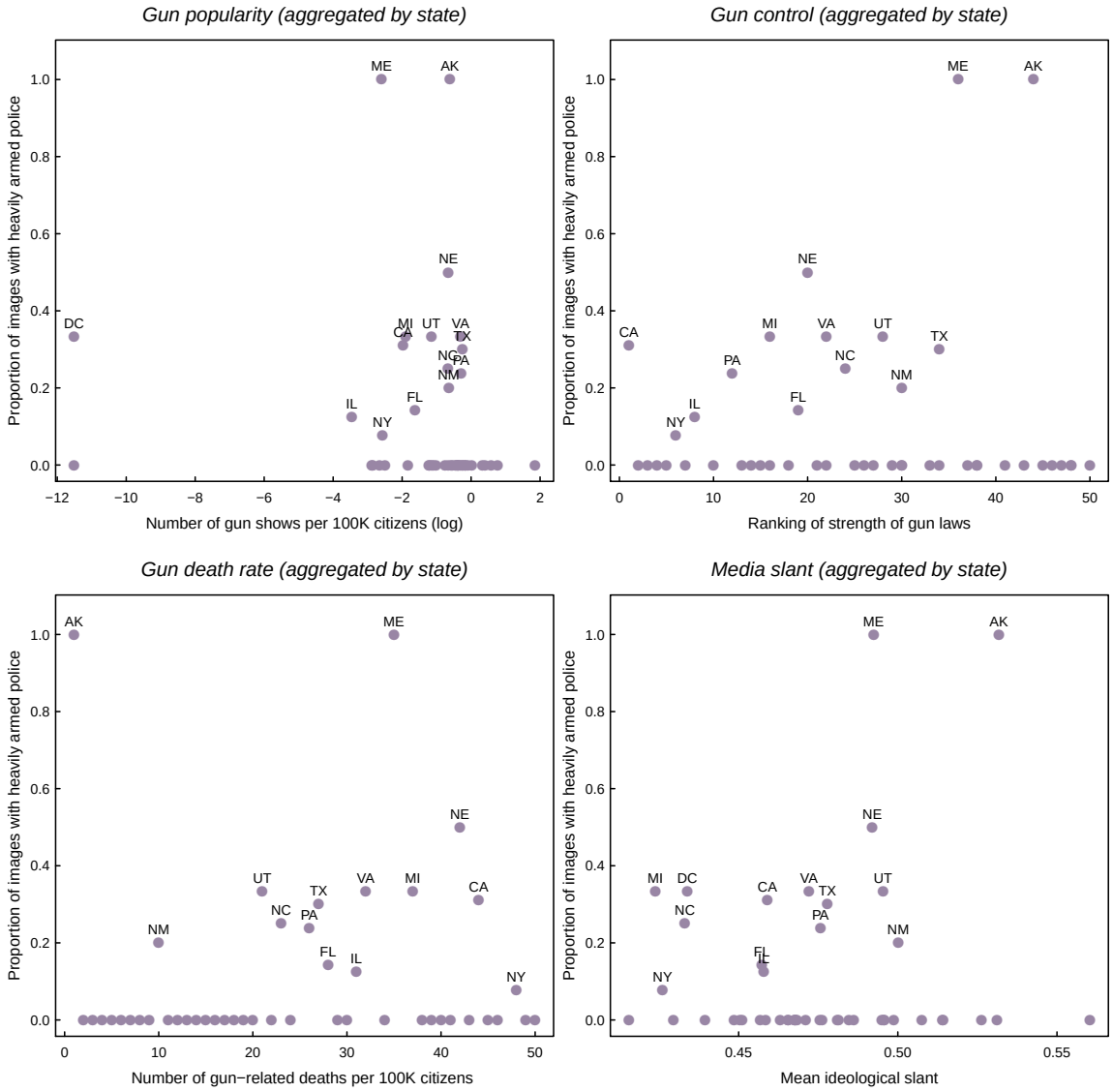
We present these basic findings to illustrate the potential application and impact of the classifications conducted by the CNN and the multiple research avenues that easy and accessible visual classification opens for researchers.

6 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

Figure 11: Factors associated with depictions of heavily armed police



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 (Lucas, 2018*b*), recording the activities of crowds in a protest (Won, Steinert-Threlkeld and Joo, 2017; Zhang and Pan, 2019), counting the number of people waiting to vote in polling stations (Stein et al., Forthcoming), as-

sessing media bias in the photographic coverage of candidates (Neumann, 2019), detecting the demographic composition of a neighborhood using its visual features (Wilcox-Archuleta, 2019), analyzing race interactions using videos (Dietrich and Sands, 2019), and others. The extraction of information from images and visual content invites a wider variety of questions.

The present article also illustrated a couple of the many benefits of CNNs for data collection purposes and image classification: data collection and content analysis for the study of visual framing. With these applications we intend to not only illustrate the benefits and substantive impact of CNNs for social scientists, but also to present some of the challenges and practical issues that researchers should consider when dealing with visual 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

Abrajano, Marisa A, Zoltan Hajnal and Hans JG Hassell. 2017. "Media Framing and Partisan Identity: The Case of Immigration Coverage and White Macropartisanship." *Journal of Race, Ethnicity and Politics* 2(1):5–34.

Abrajano, Marisa and Simran Singh. 2009. "Examining the link between issue attitudes and news source: The case of Latinos and immigration reform." *Political Behavior* 31(1):1–30.

Ali, Omar, Ilias Flaounas, Tijl De Bie, Nick Mosdell, Justin Lewis and Nello Cristianini. 2010. Automating News Content Analysis: An Application to Gender Bias and Readability. In *JMLR W&CP: Workshop on Applications of Pattern Analysis*. pp. 36–43.

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.

Boydstun, Amber E. 2013. *Making the news: Politics, the media, and agenda setting*. Chicago, IL: University of Chicago Press.

Boydstun, Amber E and Rebecca A Glazier. 2013. "A Two-Tiered Method for Identifying Trends in Media Framing of Policy Issues: The Case of the War on Terror." *Policy Studies Journal* 41(4):706–735.

Brader, Ted. 2005. "Striking a responsive chord: How political ads motivate and persuade voters by appealing to emotions." *American Journal of Political Science* 49(2):388–405.

Brader, Ted, Nicholas A Valentino and Elizabeth Suhay. 2008. "What triggers public opposition to immigration? Anxiety, group cues, and immigration threat." *American Journal of Political Science* 52(4):959–978.

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. 2019. "Images that matter: Online protests and the mobilizing role of pictures." *Political Research Quarterly* 72(2):360–375.

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* .

Coüasnon, Bertrand, Jean Camillerapp and Ivan Leplumey. 2007. "Access by content to handwritten archive documents: generic document recognition method and platform for annotations." *International Journal of Document Analysis and Recognition (IJDAR)* 9(2-4):223–242.

Dardis, Frank E, Frank R Baumgartner, Amber E Boydstun, Suzanna De Boef and Fuyuan Shen. 2008. "Media framing of capital punishment and its impact on individuals' cognitive responses." *Mass Communication & Society* 11(2):115–140.

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.

Dietrich, Bryce and Melissa Sands. 2019. "Seeing Racial Avoidance on City Streets." 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.

Dunaway, Johanna. 2008. "Markets, ownership, and the quality of campaign news coverage." *The Journal of Politics* 70(4):1193–1202.

Elkan, Charles. 2012. "Evaluating Classifiers." Working Paper.

Farris, Emily M and Heather Silber Mohamed. 2018. "Picturing immigration: how the media criminalizes immigrants." *Politics, Groups, and Identities* 6(4):814–824.

Gentzkow, Matthew and Jesse M. Shapiro. 2010. "What Drives Media Slant? Evidence From U.S. Daily Newspapers." *Econometrica* 78(1):35–71.

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

Grimmer, Justin and Brandon Stewart. 2013. "Text ad Data: The Promise and Pitfalls of Automatic Content Analysis Methods for Political Texts." *Political Analysis* 21(3):267–297.

Hastie, Trevor, Robert Tibshirani and Jerome Friedman. 2009. *The elements of statistical learning*. New York: Springer.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 770–778.

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.

Ito, Tiffany A., Jeff T. Larsen, N. Kyle Smith and John T. Cacioppo. 1998. "Negative information weighs more heavily on the brain: The negativity bias in evaluative categorizations." *Journal of Personality and Social Psychology* 75(4):887–900.

Iyengar, Shanto. 1994. *Is anyone responsible?: How television frames political issues*. University of Chicago Press.

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.

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

Lladós, Josep, Partha Pratim-Roy, José A Rodríguez and Gemma Sánchez. 2007. Word spotting in archive documents using shape contexts. In *Iberian Conference on Pattern Recognition and Image Analysis*. Springer pp. 290–297.

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

Lucas, Christopher. 2018*b*. "A Supervised Method for Automated Classification of Political Video." 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* .

Marcus, George E, W Russell Neuman and Michael MacKuen. 2000. *Affective intelligence and political judgment*. Chicago, IL: University of Chicago Press.

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.

Mutz, Diana C. 2007. "Effects of "in-your-face" television discourse on perceptions of a legitimate opposition." *American Political Science Review* 101(4):621–635.

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.

Neumann, Markus. 2019. "Fair and Balanced? News Media Bias in the Photographic Coverage of the 2016 U.S. Presidential Election." Working paper.

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.

Pan, Sinno Jialin, Qiang Yang, Wei Fan and Sinno Jialin Pan. 2010. "A survey on transfer learning." *IEEE Transactions on Knowledge and Data Engineering* .

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.

Stein, Robert M., Christopher Mann, Charles Stewart, III, Zachary Birenbaum, Anson FungJed Greenberg, Farhan Kawsar, Gayle Alberda, R. Michael Alvarez, Lonna Atkeson, Emily Beaulieu, Nathaniel A. Birkhead, Frederick J. Boehmke, Joshua Boston, Barry C. Burden, Francisco Cantu, Rachael Cobb, David Darmofal, Thomas C. Ellington, Terri Susan Fine, Charles J. Finocchiaro, Michael D. Gilbert, Victor Haynes, Brian Janssen, David Kimball, Charles Kromkowski, Elena Llaudet, Kenneth R. Mayer, Matthew R. Miles, David Miller, Lindsay Nielson, Yu Ouyang, Costas Panagopoulos, Andrew Reeves, Min Hee Seo, Haley Simmons, Corwin Smidt, Farrah M. Stone, Rachel VanSickle-Ward, Jennifer Nicoll Victor, Abby Wood and Julie Wronski. Forthcoming. "Waiting to Vote in the 2016 Presidential Election: Evidence from a Multi-county Study." *Political Research Quaterly* .

Taylor, Ula. 2008. "Women in the documents: Thoughts on uncovering the personal, political, and professional." *Journal of Women's History* 20(1):187–196.

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

Valentino, Nicholas A, Vincent L Hutchings and Ismail K White. 2002. "Cues that matter: How political ads prime racial attitudes during campaigns." *American Political Science Review* 96(1):75–90.

Vallone, Robert P., Lee Ross and Mark R. Lepper. 1985. "The hostile media phenomenon: Biased perception and perceptions of media bias in coverage of the Beirut massacre." *Journal of Personality and Social Psychology* 49(3):577–585.

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

Webb Williams, Nora, Andreu Casas and John D. Wilkerson. Forthcoming. *Images as Data for Social Science Research: An Introduction to Convolutional Neural Nets for Image Classification*. Cambridge University Press.

Wilcox-Archuleta, Bryan. 2019. "Measuring Neighborhood Level Ethnic Visibility: Evidence from Street View Images." 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.

Wuhs, Steven T. 2014. The Partido Acción Nacional as a Right Party. In *The Resilience of the Latin American Right*, ed. Juan Pablo Luna and Cristóbal Rovira Kaltwasser. Baltimore: John Hopkins University Press pp. 219–241.

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

Zhang, Han and Jennifer Pan. 2019. "CASM: A Deep-Learning Approach for Identifying Collective Action Events with Text and Image Data from Social Media." *Sociological Methodology* 49(1):1–57.

Zhang, Xingzhou, Yifan Wang and Weisong Shi. 2018. pcamp: Performance comparison of machine learning packages on the edges. In {*USENIX*} *Workshop on Hot Topics in Edge Computing*.

7 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. 9

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. 12

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. 17

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

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. 9

filter size Product of height and width, in pixels, of a matrix representing a filter. 8

filter stride Number of pixels that a filter slides through an image. 8

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. 7

forward propagation The process in which the input data moves in the forward direction through the network. Each hidden layer accepts the input data, processes it as per the activation function and passes to the successive layer. . 12

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 $x^{'} = x - \epsilon \Delta x f(x)$. 12

hyperparameters Hyperparameters are the variables exogenous to the model that determine the network structure and how it will be trained. The values of the hyperparameters are set before the training begins and do not depend on the data.. 5

iteration It is the time in which a batch of images passes forward and backward through the network. . 13

k-fold cross validation This method directly estimates the average generalization error of the model. It first assigns the images into a given number of subsets. It picks one subset to check the accuracy of the model after it is trained with the rest of the images, repeating this process for each subset. Training starts from scratch in every iteration, so what the learned in one iteration is not transferred to a new one. . 7

layer depth Number of filters used in a layer. 8

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 ϵ in the gradient descent function (see gradient descent). 13

mini-batches Subsets of the images in the training dataset used in the batch-normalization process. 13

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. 10

receptive field The area where a given filter, or neuron, is positioned to execute a convolution. 8

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. 10

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. 10

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. 10

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). 13

weight The unknown parameter of the neural network that seek to improve the fit between the model and the data.. 11

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

7.1 Backpropagation

Suppose that a neuron j in the last layer provides a classification outcome y_j.[21] 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} :

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

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

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

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

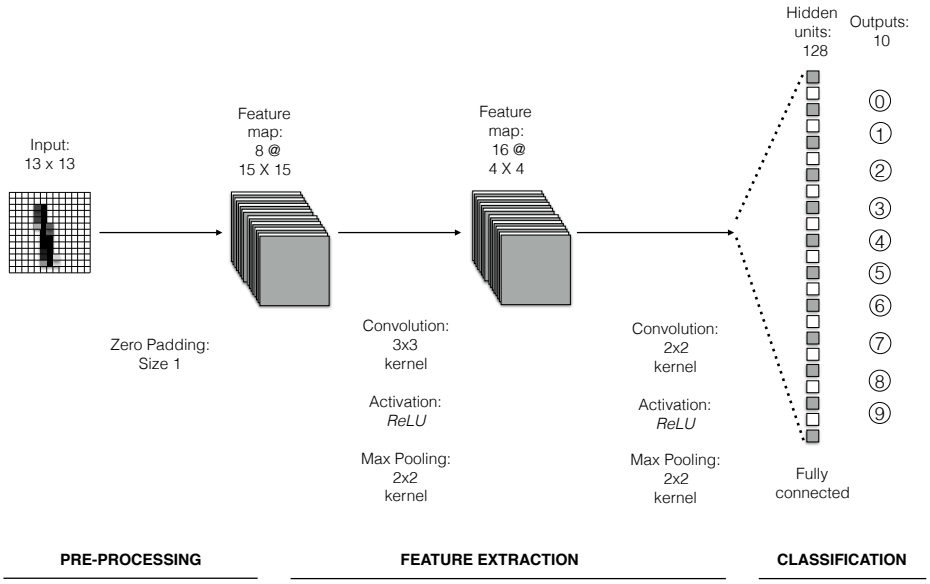
The partial derivative in Equation 5 allows the model to gradually modify its weights after reviewing a set k of examples from the database K:

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

7.2 Application 1: Coding electoral results from tallies

7.2.1 Network architecture for digit detection

Figure 12: Network Architecture



Notes: Figure 12 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.

7.2.2 Extracting digits from tallies

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.[22]

[22]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.

7.2.3 Vote counts per party in District 15: predicted vs. observed

Figure 13 presents the comparison of detected and real vote counts in the tallies of the district. 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. The triangles in Figure 13 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.

Figure 13: Number of votes registered in tallies: Official vs. Predicted