Using CNNs to Automate Risk-Limiting Audits

Risk-limiting audits are an important process for ensuring the integrity of elections. However, existing voting systems do not provide sufficient details to facilitate scalable auditing. This paper introduces *CNNScan*, a system to design to aid transitive risk-limiting audits using deep neural nets.

Introduction

Ballot counting is a highly automated process that depends on the correctness of the code running on voting systems. However, large-scale computing systems should be inherently untrusted[1]. How then can the results of an election be trusted? The solution is post-election auditing. This paper explores the application of a form of post-election auditing known as *risk-limiting audits*.

To understand an election, it is important to understand its constituent ballots. A typical ballot consists of one or more contests. Each contest typically highlights one issue, such as "Who should be President?" or "Should this bond measure pass?". Each contest has multiple options. Voting systems take in multiple ballots and provide summaries about the result of each contest by counting the number of times each option was marked. Incorrectly marked contests are excluded from the final results.

A risk-limiting audit is a process for providing "statistical assurance that election outcomes are correct"[2]. Election outcomes are widely determined by computers scanning paper ballots, and processing them contest-by-contest before providing vote tallies. Risk-limiting audits work by iteratively evaluating random paper ballots and seeing if the voting system recorded the ballot's votes the same way as a human election official. The *risk-limit* is a parameter which determines how likely one is to declare the election results in error. When determining how many more ballots to sample, one computes a function of the risk-limit, how many ballots have already been seen, and how correct each ballot is. The statistical formula yields a number (n) of ballots you need to see before certifying the election outcome. If the total number of ballots reviewed exceeds n, then the audit stops and the audit certifies the election outcome. If n exceeds a threshold (e.g., 50% of ballots) then the risk-limiting audit is aborted, and a full hand recount is performed[3].

Most election systems do not provide per-ballot voting information. *Transitive* risk-limiting audits solve this shortcoming by introducing a new system, such as *CNNScan*, to reevaluate all

[1] https://people.csail.mit.edu/rivest/RivestWack-OnTheNotionOfSoftwareIndependenceInVotingSystems.pdf
[2] https://www.stat.berkeley.edu/users/stark/Preprints/gentle12.pdf
[3] In the case where the original system reported the incorrect outcome, the threshold will be exceeded, and a full hand-recount performed.

of the ballots and provide per-ballot voting information. The transitive audit is terminated and a full hand-recount performed if:

1. The number of ballots from the original system does not match the number evaluated by the second system.
2. The second system produces a different election outcome than the first system.

If none of these conditions occur, the risk-limiting audit proceeds as usual.

This process may seem obvious at first, but few systems exist in practice to perform transitive risk-limiting audits. *CNNScan* aims to make such audits easier to perform.

Project Description

In order to facilitate the automation of risk-limiting audits, several pieces of software need to be created. Firstly, a system must be devised for annotating blank ballot PDFs with information about contained contests. Using annotation information, blank ballot PDFs can be digitally marked to become marked ballots. The generated corpora of synthetically marked ballots can be used to train a system to provide per-ballot, per-contest voting information. If the evaluation system is well trained and the synthetic data is of high quality, the model should be portable to real marked ballots (i.e. those marked by humans as part of an election). By applying the contest evaluation system to real ballots, per-ballot per-contest voting information may be extracted, facilitating a risk-limiting audit.

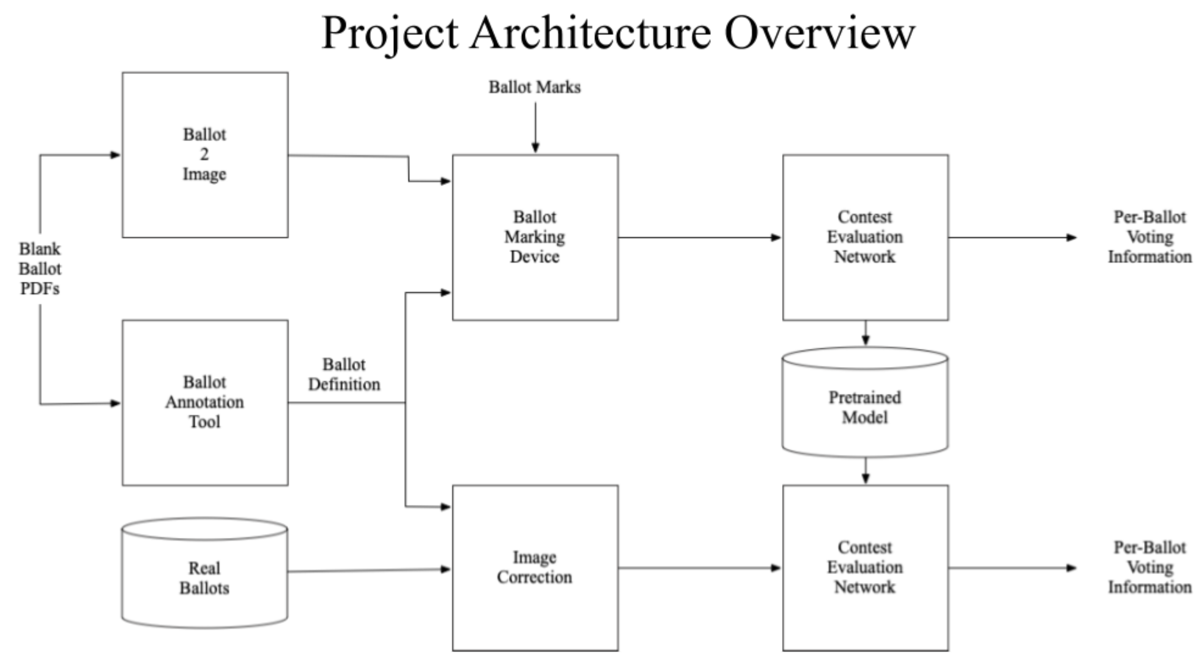


Image recognition, classification, and generation tasks are natural targets for deep neural networks. Projects, such as FRCNN[4], have had success finding bounding rectangles for objects, which is very similar to finding contests on a ballot. StyleGan[5] has success generating realistic

[4] See https://arxiv.org/abs/1506.01497

[5] See https://arxiv.org/abs/1812.04948

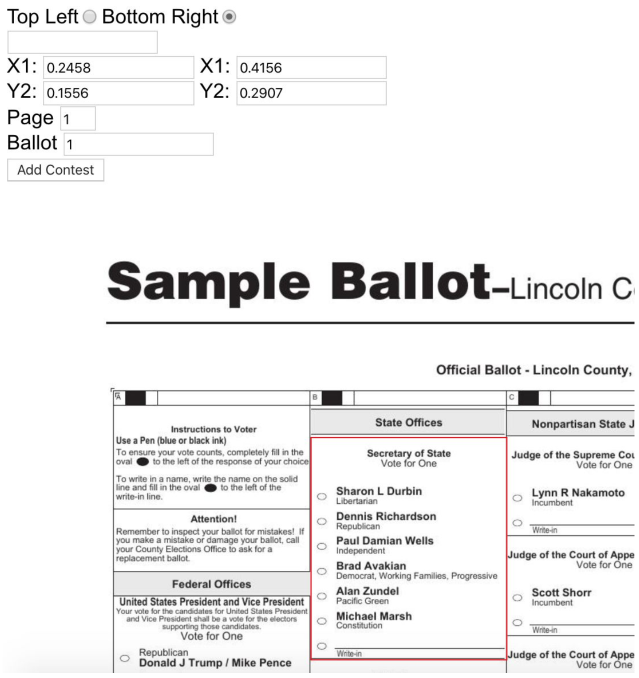
faces, so it seems possible to generate marked ballots using GANs. Lastly, optical mark recognition (OMR) is a similar task to optical character recognition (OCR), even if the specifics of the recognition task are different between OCR and ballot reading.

Methodology

This section presents the design and interactions of the three major neural network systems in the project. It presents the automatic ballot annotation system, which uses FRCNN[6]. Then, it discusses using a GAN as a step in generating synthetic marked ballots. Lastly, it evaluates the design goals and architecture of the CNN performing contest evaluation.

Ballot Annotation

Due to the lack of available marked ballots, a large portion of this project is dedicated to creating marked ballots. Given an unmarked sample ballot image, a marked ballot would have one mark per contest for each contest on the ballot. All sample ballots in our dataset used race-track shaped bubbles to indicate contest options. A ballot annotation contains information about the dimensions of the ballot, the relative locations of all contests, and the locations of options within each contest. Ballot annotations contain floating point numbers that represent the relative position of each contest and option in the image. In order to avoid manual pixel arithmetic, we created a simple web application (using Flask) that speeds up the process of describing bounding rectangles. These contest and option bound rectangles are saved to the disk for use in synthetic ballot generation.



Top Left [✓ Bottom Right]

X1:	0.2458	X1:	0.4156
Y2:	0.1556	Y2:	0.2907

Page 1
Ballot 1
Add Contest

Sample Ballot–Lincoln C

Official Ballot - Lincoln County,

A

Instructions to Voter
Use a Pen (blue or black ink)
To ensure your vote counts, completely fill in the oval ● to the left of the response of your choice
To write in a name, write the name on the solid line and fill in the oval ● to the left of the write-in line.

Attention!
Remember to inspect your ballot for mistakes! If you make a mistake or damage your ballot, call your County Elections Office to ask for a replacement ballot.

Federal Offices
United States President and Vice President
Your vote for the candidates for United States President and Vice President shall be a vote for the electors supporting those candidates.
Vote for One
Republican
Donald J Trump / Mike Pence

B

State Offices
Secretary of State
Vote for One
Sharon L Durbin
Libertarian
Dennis Richardson
Republican
Paul Damian Wells
Independent
Brad Avakian
Democrat, Working Families, Progressive
Alan Zundel
Pacific Green
Michael Marsh
Constitution
Write-in

C

Nonpartisan State J
Judge of the Supreme Co
Vote for One
Lynn R Nakamoto
Incumbent
Write-in
Judge of the Court of Appe
Vote for One
Scott Shorr
Incumbent
Write-in
Judge of the Court of Appe
Vote for One

While the web application speeds ballot annotation, the process is still slow and tedious for a single person to create a diverse set of ballot annotations. Given enough annotated ballots, a neural net should be capable of learning to automatically annotate ballots. Faster-RCNN, a pre-trained PyTorch regional convolutional neural network[7], is designed to recognize objects within a picture. Since contests are objects within pictures, a tuned FCRNN will automate the annotation process and reduce human involvement in the process.

[6] https://arxiv.org/abs/1506.01497
[7] https://arxiv.org/abs/1703.06870

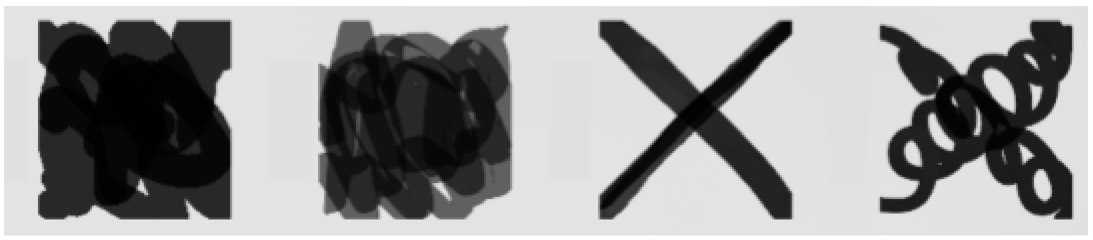
Synthetic Data Generation

A major issue encountered by this project is a lack of data. Election officials tend to closely guard marked ballots; there are not large, open corpora of annotated, marked ballots produced by real elections. If such corpora existed, the wide variety in ballot styles and configurations would likely limit the portability of training on one corpus and transferring to another. Therefore, a synthetic corpus of marked ballots must be created.

To produce a sufficiently-sized dataset to train neural nets, we need an automatic ballot marking system that uses existing blank sample ballots from local elections. Ballot marks are as varied as the people marking them; furthermore, a single person is expected to produce non-identical marks for different contests. In order to replicate this, the mark generation system must produce a wide variety of ballot marks

The object responsible for ensuring mark variety, the mark database, yields images that look like marks one may find on a ballot. While it would be preferable if all marks were unique with complex shading/transparency, the system may also be configured to produce geometric shapes, like colored squares or X's. To automate production of complex marks, we attempted to train a GAN on a custom mark dataset. Using the GAN, we have a near limitless source of unique marks to fill a mark database.

Sample Marks

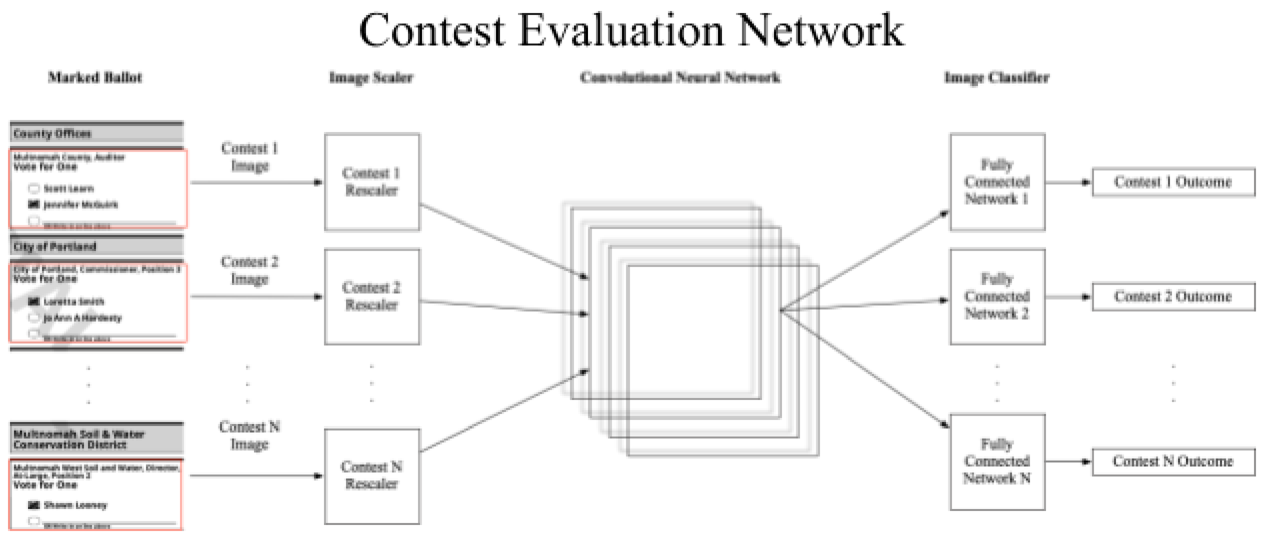


The mark generation system takes in a blank ballot PDFs, ballot annotations, and a mark database. To account for the fact that a single county may process multiple ballots simultaneously, the marking system is capable of handling arbitrary numbers of ballot types at once. By sampling marks from the database, the system generates an arbitrary number of marked ballots with random marks applied to random options for all contests on a ballot. The marked images as well as the "true" voting record for each contest is stored as a dataset for the neural network.

Contest Evaluation

The goal of the *contest evaluation network* (CEN) is to take in a marked ballot, split the ballot into a sequence of contests, and determine how the ballot voted for each contest.

One constraint of the CEN is that it must process multiple different styles of ballots. Counties have multiple different ballots types (e.g., ballots have different city contests) within a single election. Very often these ballot types are shuffled together. The CEN handles multiple different ballot types, but within each batch the ballots must be the same type[8]. This fulfills the requirement of the system, as a clever data loader may be used to properly collate ballots.



We tried multiple different network architectures for our contest evaluation network. These different architectures are selected via switches. The first switch enables downsampling. Without downsampling, all images are padded to be the size of the largest image. With downsampling, a combination of padding and pooling are used to convert all images to a fixed resolution. The second switch controls cropping tightness. It changes whether cropping selects either the smallest region containing all contest options or if cropping selects the entire contest region. The third switch---weight sharing---determines if the fully connected and output layers are shared between all ballots and all contests or if these layers are unique to each contest. These three flags provide eight different architectural combinations in addition to the typical hyperparameters, linear layer and convolutional layer configurations.

Results

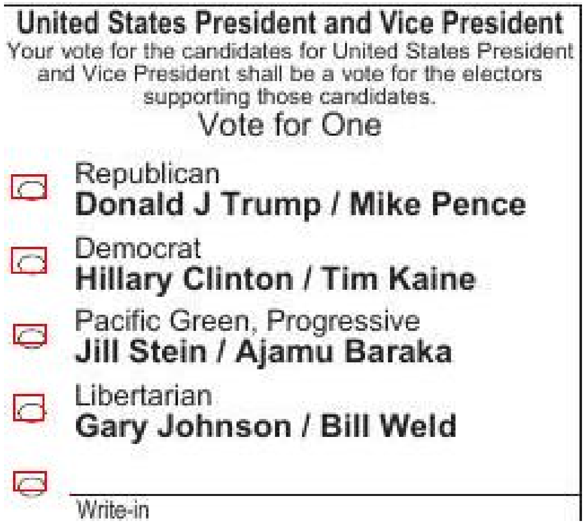
This section discusses the results of each of the three neural networks. First, it examines the qualitative accuracy of recognizing option bounding rectangles between annotated and unannotated ballots. Next, it explores the GAN's output quality issues as well as potential causes. Lastly, it details how different configurations of the contest evaluation network affect speed, memory foot-print, and accuracy.

[8] Different ballots have different image sizes and different numbers of contests. This is hard to parallelize.

Ballot Annotation

Locating contests and options within the image is an object detection task that requires annotating enough ballots so a neural network may identify its shape. Each ballots' contests contain up to seven options. As marking up the options within a contest is more tedious than finding contests on a ballot, option recognition was the first task to be automated. Faster-RCNN is powerful enough to train on three annotated contests of multiple different ballot styles and then accurately detect the bounding rectangles for every option in the remaining contents on that ballot. Collecting accuracy statistics for bounding rectangle locations would require annotating a large number of ballots, which is precisely the task this model was created to avoid. Therefore, results were evaluated qualitatively by saving evaluated contest images with red boxes drawn around detected options, and all detections are accurate to the human eye.

Automatic option annotation requires a small amount of labor to produce enough data to train a model well. Annotating twelve options between two or three contests for each option style is not terribly demanding. Automatic contest annotation is a more difficult task, and will require additional work to complete. Option bubbles are uniform in size and shape, they always appear in a straight line, they are evenly spaced out. Options take up a relatively small area of a contest, but contest boxes collectively fill the ballot page. Contests vary in shape and size, and there is often no space between contests. For these reasons, the object detection task for contest bounding rectangles is significantly more difficult and will require more data than we had time to produce for sufficient training.


United States President and Vice President
Your vote for the candidates for United States President and Vice President shall be a vote for the electors supporting those candidates.
Vote for One
Republican
Donald J Trump / Mike Pence
Democrat
Hillary Clinton / Tim Kaine
Pacific Green, Progressive
Jill Stein / Ajamu Baraka
Libertarian
Gary Johnson / Bill Weld
Write-in

Synthetic Data Generation

We tried two main approaches to synthetic mark generation: application of geometric shapes and a GAN. Ballots marked with geometric shapes are easily recognizable by the contest evaluation network. Despite the high accuracy, portability of these results to real ballots is questionable. Real people do not mark ballots with sub-pixel precision; so other marking methods, like GANs, are necessary for creating data that leads to portable results.

Using our mark corpus, we trained our GAN to produce other mark-like images. After extensive hyperparameter sweeping, few high-quality images were generated by the network. To debug the generator, we created an auto-encoder using the GAN's generator. This auto-encoder produced poor results like the GAN.

After examining the outputs of the network, it appears we devise a new method of mapping the output of the generator to a pixel value. Pixels perceived luminance (brightness) is a linear function of its components (predominantly the green channel). However, the output of our generator is run through a TanH layer to clamp values to [-1,1]. The TanH does *not* produce a linear distribution of values, especially as the desired outputs approach [-1, +1]. Most images produced with this function will have strong midtones and weak whites/blacks. However, our sample images' most common values are 0 and 255, that is, strong whites/blacks with weak midtones. Such an image requires a TanH output of [-1, +1] respectively. The neural network must generate -inf, +inf to generate these colors. Such a network is inherently unstable, explaining the poor image quality.

Contest Evaluation

Since the contest evaluation network contains several orthogonal configuration switches, each switch's impact on the performance of the system must be examined independently. For the following evaluation, a dataset of 100 marked ballots with 25 contests per ballot was used. All ballots were of the same type. Using a batch size of 10, a total of 2500 images over 10 batches were presented to the CEN. Each network was trained for 100 epochs. Ballots were marked with randomly colored boxes and X's; marks generated by the GAN are *excluded*. Ballots were rendered at 40 PPI. Unless otherwise specified, the following results used the following flags: no downsampling, loosely cropped images, and shared weights.

The first switch, downsampling, has an extreme effect on the accuracy of the network. In networks without downsampling, the network converged on 95% accuracy within 100 epochs. With downsampling enabled, the network rarely exceeded 40% accuracy. The poor accuracy appears to be caused by feature distortion from the pooling layers. That is, the features being recognized changed sizes and positions in every different contest image, making the recognition task difficult. There is little overlap in the recognition task between different contests. For this reason, padding without downsampling is the preferred configuration. The table below summarizes the results.

Effect of Downsampling	With Downsampling	Without Downsampling
Accuracy at 50 epochs	38.48%	94.84%
Accuracy at 100 epochs	40.48%	97.72%

The second switch, cropping tightness, does not affect the accuracy of the network but significantly affects processing speed. Tightly cropped images have an order of magnitude fewer pixels than loosely cropped images, meaning less work is performed by the neural network. Tightly cropped images complete an epoch in five seconds while loosely cropped images take 18 seconds to complete a single training epoch. Tightly cropped images run the risk of missing details like crossing off the names of the candidates a voter voted against, but the assumption is that ballots that ignore directions will be relatively rare and are able to be handled by humans. However, a far larger neural network is required to achieve the same level of accuracy with the loosely cropped images. Therefore, the tightly-cropped configuration should be preferred. The table below summarizes the results.

Effect of Cropping	Tightly Cropped	Loosely Cropped
Accuracy at 50 epochs	94.84%	41.48%
Accuracy at 100 epochs	97.72%	41.64%

The third switch, weight sharing, determines if the fully connected and output layers are shared between all contests or if each contest has unique fully connected and output layers. Both configurations converged on 95% accuracy after 100 epochs. On an epoch-by-epoch basis, the unshared network consistently had higher accuracy than the shared configuration. However, The shared configuration significantly reduced memory overhead. Environments with high memory pressure, like machines with little VRAM, should prefer shared networks, while those with significant VRAM should prefer the reduced training time associated with unshared networks. The table below summarizes the results.

Effect of Weight Sharing	Shared Weights	Unshared weights
Accuracy at 50 epochs	94.84%	99.16%
Accuracy at 100 epochs	97.72%	99.68%

Next Steps

Future work for this project relies heavily on the availability of data, and our ability to automatically annotate ballots could be a heavy factor. We feel that we have proved the concept of successful object detection for annotating options, but object detection for contests was out of reach. While our model necessitated contest detection as part of automatic ballot annotation, it may not be a requirement in general. For example, instead of evaluating marks by contest, a model similar to ours could evaluate marks by ballot, requiring only a few option annotations per ballot style and a small amount of ballot-specific post-processing; a third similar model could infer the contest of each option based on distance from other options. Regardless of future improvements or alternatives, the current results of our object detection model suggest that automatic ballot annotation can be achieved with less requirements or more data.

To fix the issues encountered with the GAN for ballot mark generation, we will need to devise a new method of normalizing images and clamping outputs. Instead of using fixed values for the mean and standard deviation, we should compute a per-channel mean and standard deviation across our dataset and normalize with those values. Additionally, we will need to create some new output normalization function to the GAN that produces contrast curves similar to the distribution of the images we currently have.

One upcoming improvement to the contest evaluation network revolves around determining ballot types. At the moment, the CEN requires labels describing which ballot type the current marked ballot belongs to. This information is known with synthetic data, but is not provided with real marked ballots. Since we have had success in training a CNN to find contests on ballots as well as determine the outcomes of contests, it seems plausible to train a new CNN that can differentiate between multiple types of ballots. The output of this network would be used as ballot labels for the CEN, allowing the existing network to function as before.

With all these improvements, we hope to have a comprehensive system that can help automate the tedious process of performing a risk-limiting audit.

Conclusion

Our project aims to create large numbers of synthetic ballots and to train a system to evaluate the choices made on a ballot. To ease the creation of synthetic ballots, we created ballot annotation tools and designed neural networks to automate annotation. Using the annotations, we attempted to train a GAN to generate marks that look like the marks humans place on ballots. The synthetic ballots are fed through a neural network that scores each contest to provide a per-contest, per-ballot record of votes. By providing highly detailed election data, our project should facilitate risk limiting audits, and increase confidence in election outcomes.