

# Teaching Cozmo the wonderful world of Uno

Rebecca Manley & Nathan Glover

# How do we recognize cards?

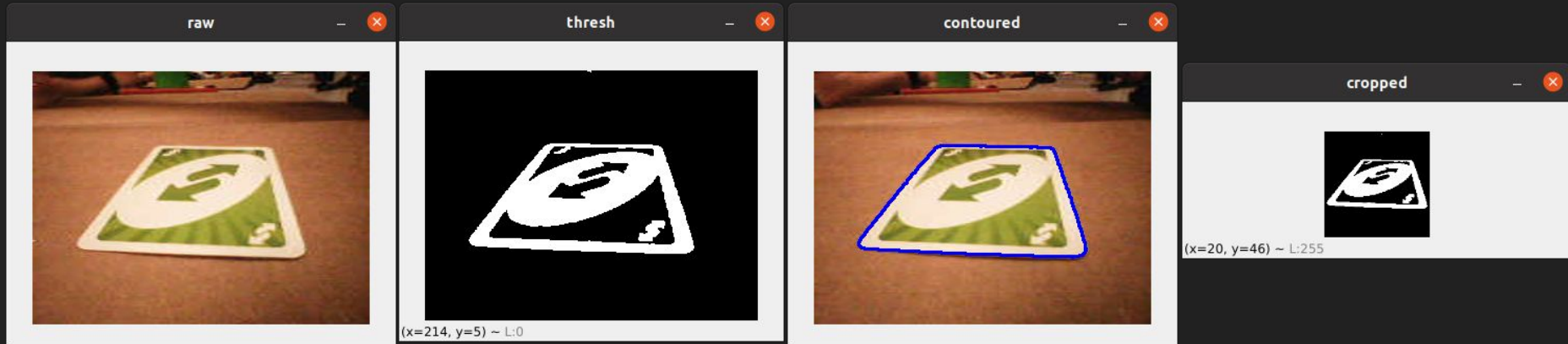
- Unfortunately, the internet was a bit lacking on cozmo height/angle training sets of uno cards
  - It was pretty lacking on uno cards in general, but we did find one birds-eye view sort of set. Not exactly the best for our situation
- So, we set out to train our own CNN, on our own card images

## Simplifying the problem

- We wanted to minimize the complexity of the network, so we made some simplifications to what we wanted to identify
  - 1: don't consider color (we can determine that separately)
  - 2: give it only black and white, no grayscale
  - 3: 6's and 9's are too similar, so we're starting with no-9's uno
- So at the end of the day, we had 12 types of cards: numbers 1-8, skip, reverse, and +2
  - Technically 13 types, but we handle wildcards separately. They are the only black cards, so once we identify the color we do not have to identify the "number"

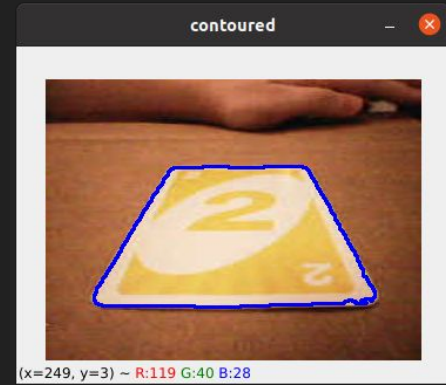
# Making a nice image (example: green)

We get a raw color image from Cozmo. Threshold to separate the white parts of the card from everything else, then locate where the card is with contours (biggest contour). Use the contour bounding box to crop it square, and we're good to go!



# The accursed yellow

That method worked *great* on red, blue, and green. Yellow... not so much.



(The card contour was still nice though, at least)

# Our Solution: Look at the color



To get around the low contrast on yellow, we stopped looking for light/dark and started looking for yellow/not yellow. Pictured below ('yellow mask') is the result of our color thresholding which spits out a 255 if the color is in the range we called 'yellow', and a 0 otherwise (inverted to match the other pictures). Once we had this nice and crisp, we combined it with the threshold from before to remove the background white.

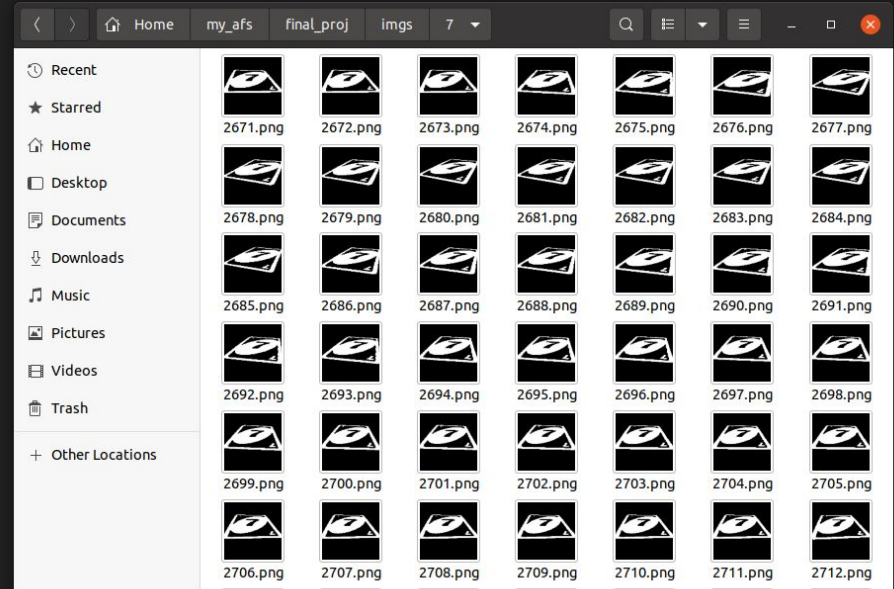
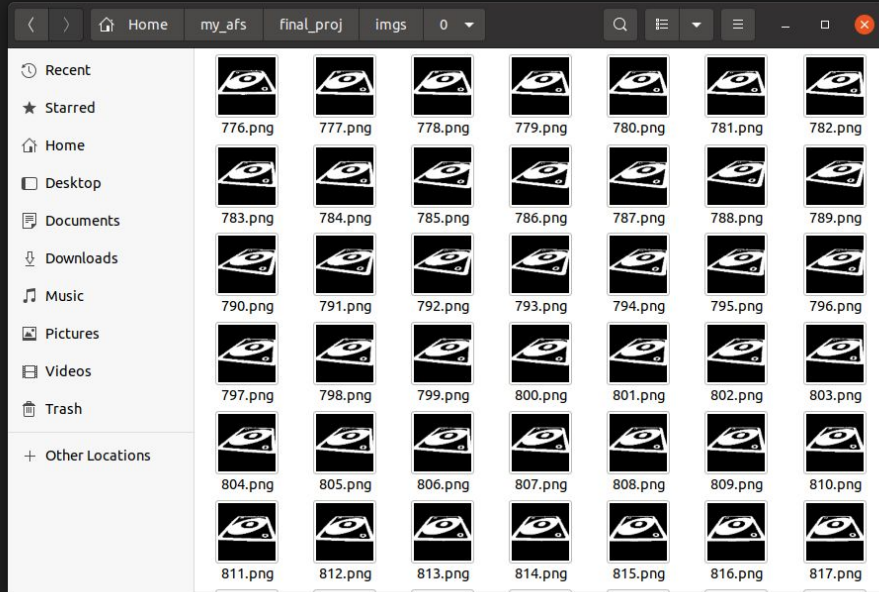


# Cool, we can see cards... Now what? Collecting data

We made an FSM to take a continuous video (processed as we just discussed) and save each frame according to what the card is (which is supplied via TM by the user). To vary the images we were saving, we:

- Have the Cozmo move slowly forward and backward. This changes the viewing angle we see the card from, and we want to be a bit resilient to how we are positioned
- Continuously turn the card. We also want resilience to the angle of the card
  - (This was accomplished by taping it to the end of a chopstick shoved through the background, and turning the chopstick)

# Data!



# Time for the machine to learn

The general network we've been trying out for this is a two layer convolutional network. We also tried a couple of different datasets, with varieties such as

- Spin the cards a full 360
- Assume the card is facing the Cozmo, and only wiggle by about +/- 15 degrees
- Include +2 cards
- Don't include +2 (since unfortunately the deck we have is a bit quirky)

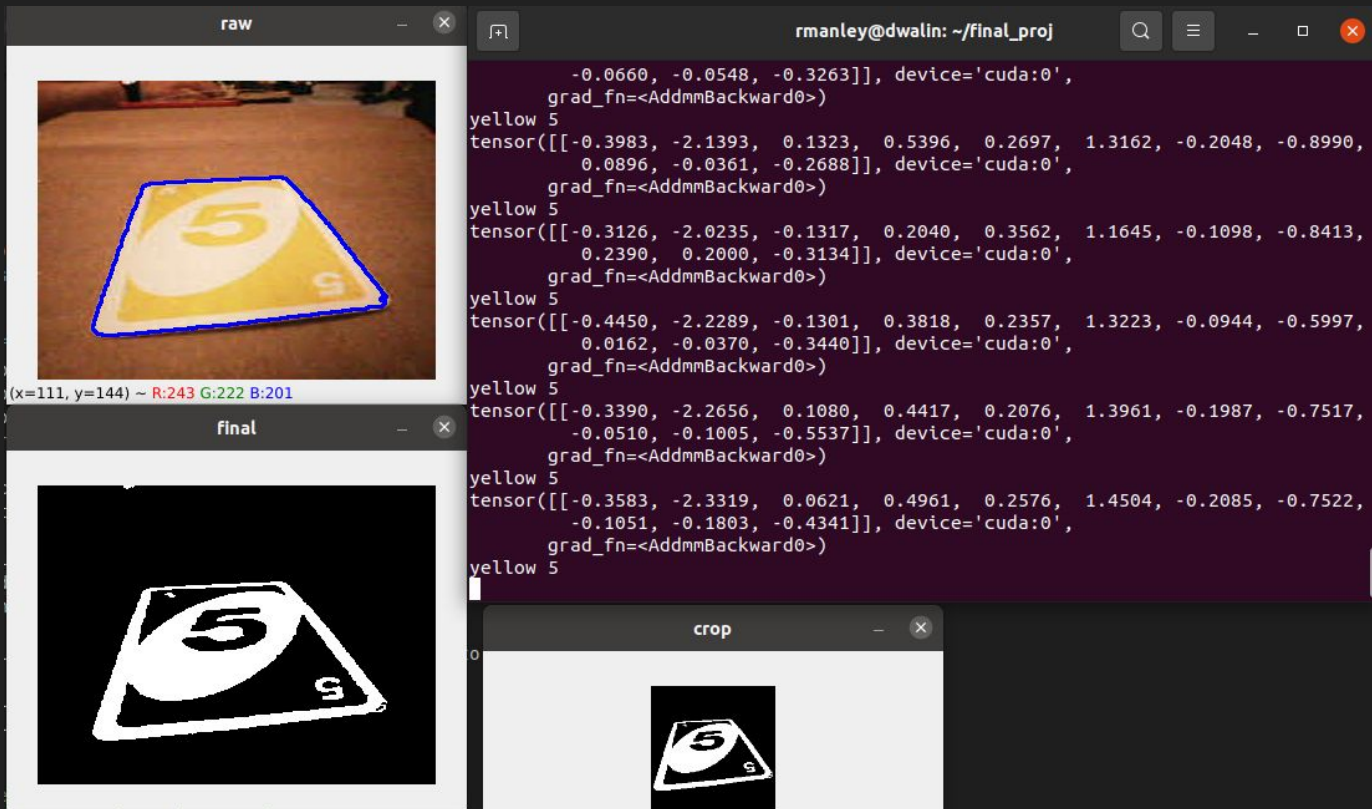
```
rmanley@dwalin: ~/final_proj
blue skip
Quit (core dumped)
rmanley@dwalin:~/final_proj$ python3 training.py
device is cuda:0
{0: '0', 1: '1', 2: '2', 3: '3', 4: '4', 5: '5', 6: '6', 7: '7', 8: '8', 9: 'reverse', 10: 'skip'}
Loading...
Loading complete.
 0 loss = 9860.29   time = 8.19 s   correct = 14.19 %
 1 loss = 9087.43   time = 1.90 s   correct = 27.89 %
 2 loss = 8471.69   time = 1.88 s   correct = 39.33 %
 3 loss = 7936.35   time = 1.88 s   correct = 47.70 %
 4 loss = 7439.74   time = 1.88 s   correct = 53.83 %
 5 loss = 7007.27   time = 1.92 s   correct = 58.44 %
 6 loss = 6598.08   time = 1.92 s   correct = 64.38 %
 7 loss = 6232.71   time = 1.89 s   correct = 67.80 %
 8 loss = 5876.54   time = 1.92 s   correct = 72.32 %
 9 loss = 5569.75   time = 1.94 s   correct = 75.12 %
10 loss = 5276.28   time = 1.90 s   correct = 77.58 %
11 loss = 5021.83   time = 1.89 s   correct = 80.17 %
12 loss = 4766.58   time = 1.89 s   correct = 81.94 %
13 loss = 4546.54   time = 1.89 s   correct = 83.37 %
14 loss = 4336.70   time = 1.89 s   correct = 85.07 %
15 loss = 4154.41   time = 1.89 s   correct = 87.13 %
16 loss = 3981.89   time = 1.90 s   correct = 87.97 %
17 loss = 3819.72   time = 1.89 s   correct = 88.90 %
18 loss = 3664.73   time = 1.89 s   correct = 90.17 %
^Quit (core dumped)
rmanley@dwalin:~/final_proj$
```



# Training went fine, but performance is... uhh...

- Most cards are very finicky
  - With enough minor movements/tweaks, we can get most cards to reliably identify
  - But only in the “perfect” position and angle
- A few were actually pretty consistent, which was nice to see
  - especially reverses and skips
  - but also 7’s sometimes
- (Note: color detection is being done via the sort of color thresholding we talked about earlier. We’ve specified ranges for each color, and assume that whichever range has the highest response is the color of the card. This is pretty reliable as long as the video doesn’t have a ton of stuff in the background)

# Here is a 5 in its ideal position



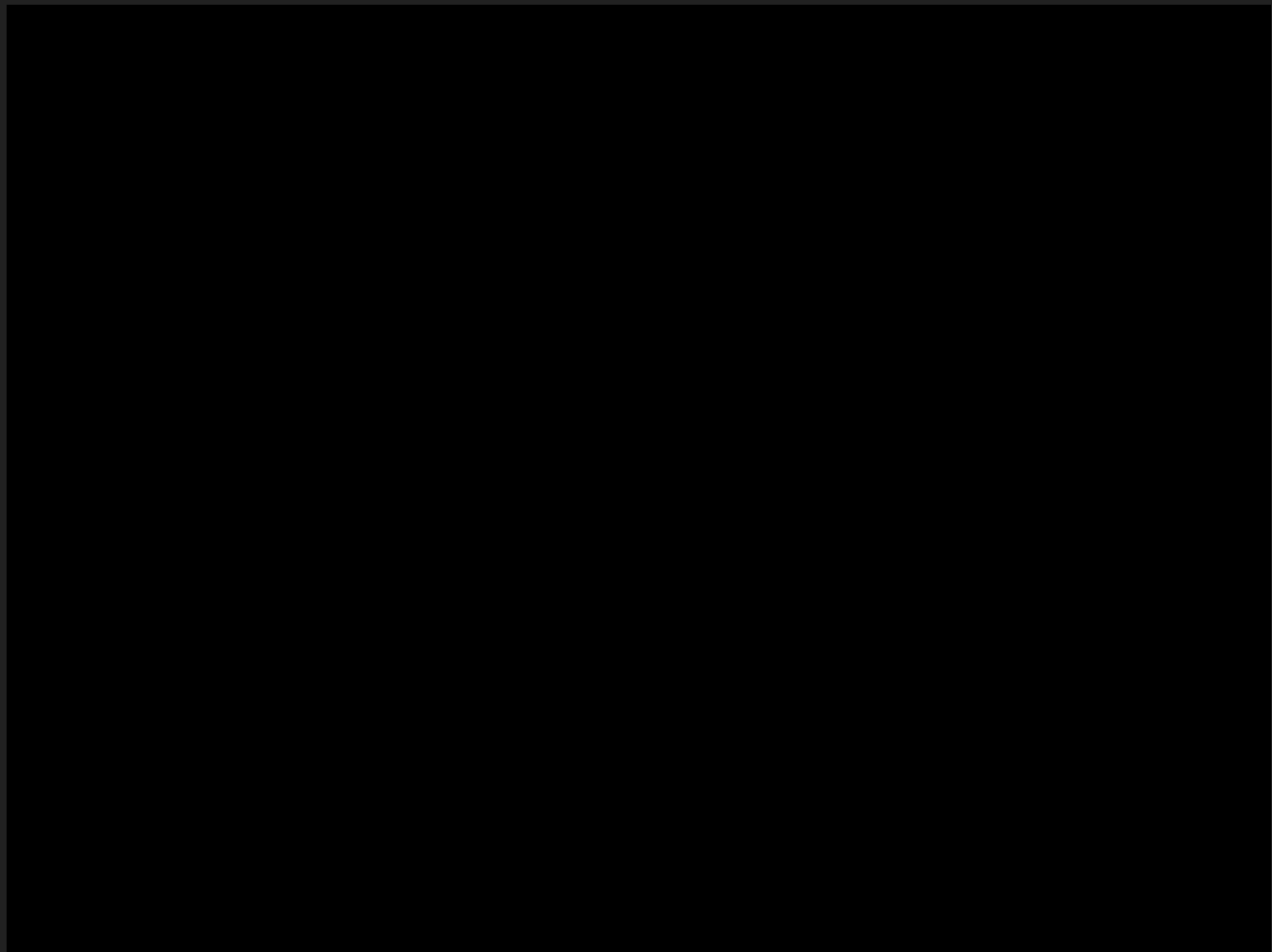
The image displays a sequence of steps in a digit recognition process:

- raw**: Original image of a baseball with a blue bounding box around the number 5. Coordinates: (x=111, y=144) ~ R:243 G:222 B:201.
- final**: The digit 5 is isolated in white on a black background.
- crop**: A smaller version of the isolated digit 5.

The terminal window shows the output of a neural network, including the number 5 and several 3x3 weight tensors:

```
yellow 5
tensor([[ -0.0660, -0.0548, -0.3263]], device='cuda:0',
        grad_fn=<AddmmBackward0>)
yellow 5
tensor([[ -0.3983, -2.1393,  0.1323,  0.5396,  0.2697,  1.3162, -0.2048, -0.8990,
          0.0896, -0.0361, -0.2688]], device='cuda:0',
        grad_fn=<AddmmBackward0>)
yellow 5
tensor([[ -0.3126, -2.0235, -0.1317,  0.2040,  0.3562,  1.1645, -0.1098, -0.8413,
          0.2390,  0.2000, -0.3134]], device='cuda:0',
        grad_fn=<AddmmBackward0>)
yellow 5
tensor([[ -0.4450, -2.2289, -0.1301,  0.3818,  0.2357,  1.3223, -0.0944, -0.5997,
          0.0162, -0.0370, -0.3440]], device='cuda:0',
        grad_fn=<AddmmBackward0>)
yellow 5
tensor([[ -0.3390, -2.2656,  0.1080,  0.4417,  0.2076,  1.3961, -0.1987, -0.7517,
          -0.0510, -0.1005, -0.5537]], device='cuda:0',
        grad_fn=<AddmmBackward0>)
yellow 5
tensor([[ -0.3583, -2.3319,  0.0621,  0.4961,  0.2576,  1.4504, -0.2085, -0.7522,
          -0.1051, -0.1803, -0.4341]], device='cuda:0',
        grad_fn=<AddmmBackward0>)
yellow 5
```

Video  
demo of  
some  
cards



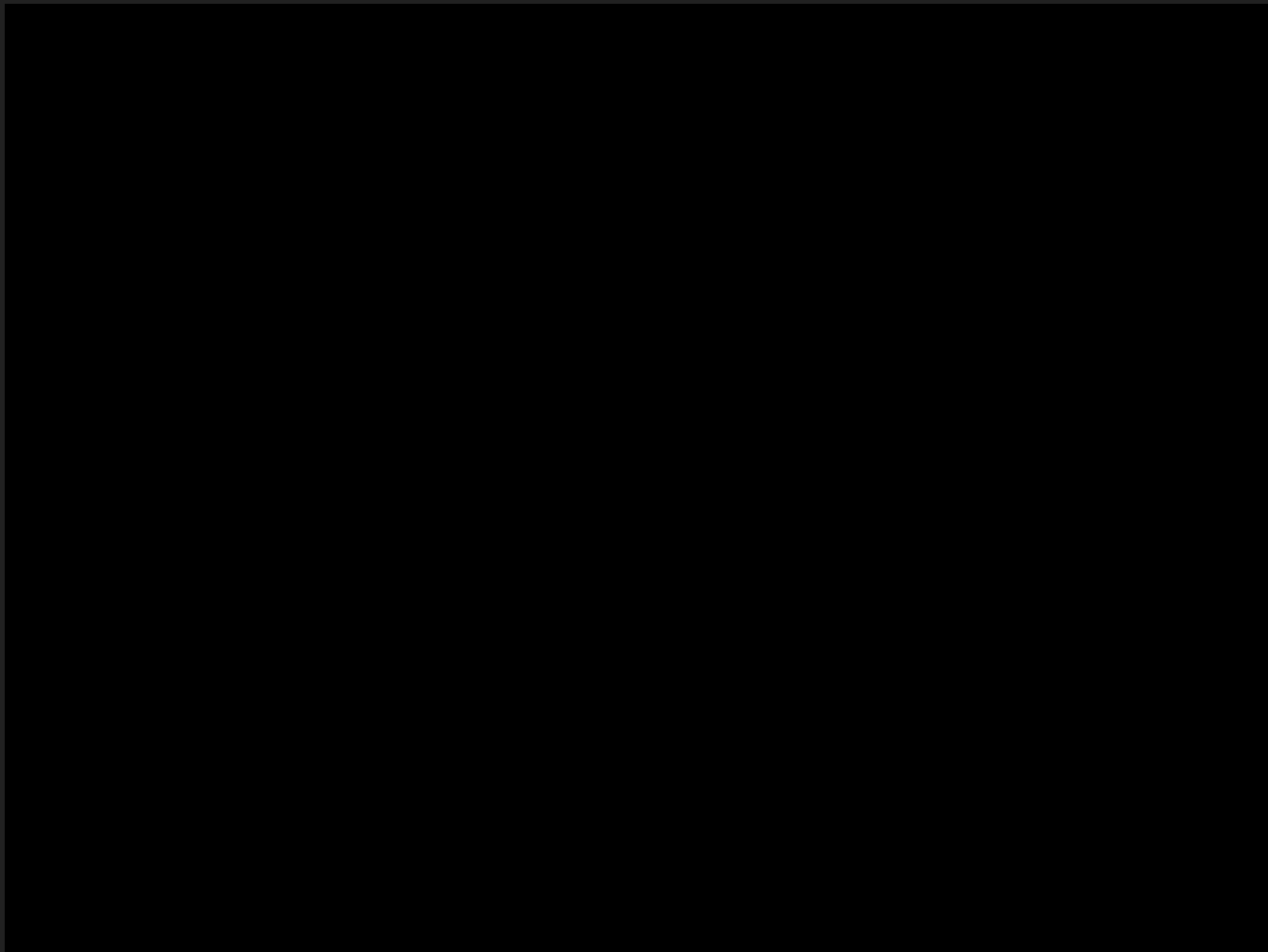
# Why is it like this?

- Our best guess is that due to our sorta limited training set (~6k images) and/or the way we're training it, we're getting a decent amount of overfitting
- And/or maybe that something about our parameters isn't letting catch enough detail or is letting it focus on too much detail, etc
- We've played around, but this is more or less the best we've gotten
  - more/less weight decay
  - more/less kernels
  - more/less layers
  - You get the idea
  
- Future work: continue trying to improve the network, and/or try a simpler method of number determination (like shape match)

# Questionable card recognition aside, can we actually play the game?

- We are still in the process of teaching the Cozmo the physical mechanics of being able to play the game
- At time of writing this, we have a semi reliable system to manipulate (pick up and drop) the cards
  - Using a lift attachment created out of some wire and some tape, Cozmo can lower and raise his lift to pick up a card
  - In order to drop it, we need something to hold it down as Cozmo raises his lift back up
    - We tried a wire strung across the background, but that didn't provide a lot of force
    - Now we're trying a cardboard frame, which seems to allow for more reliable drop off.
  - It is a fine balancing act between being too sticky, and not sticky enough

# Lift Demo



# Future work: Putting together the pieces

- It is still our overall vision/dream/etc that Cozmo may one day play a game of Uno with minimal human intervention
- In order to accomplish this we imagine a world where:
  - Cozmo has a fixed number of card slots, each marked with an Aruco. The aruco will allow Cozmo to line up fairly reliably with any card in his hand
  - Cozmo has a little house/shield around his 'hand', for privacy
  - The central draw pile is face up, under a tent. An aruco will mark the entrance
  - The opponent will need to take the card from Cozmo when he wants to 'play' it and set it down
  - Basically arucos everywhere and a well specified world map, so Cozmo can most neatly navigate the 'arena'