Peekaboom: A Game for Locating Objects in Images

By Luis von Ahn, Ruoran Liu, and Manuel Blum

ESP Game

- Only attaches labels to random images from the internet
- Does not specify location of image
  - Thus, insufficient for training computer vision algorithms

- Given label, can we identify where label actually is?

What is Peekaboom?

- 2 players:
  - Boom
    - Start with an image & related word
    - Gradually reveal 20 pixel radius circles of the image at a time
  - Peek
    - Start with blank screen
    - Try to guess the image label
- Every round, 4 images processed

Sample Boom Image

Sample Peek Image

Extra Features of Peekaboom

- Pass
- Hint
- Ping
- Bonus Round

Each one of these features can help provide examples for machine learning

Data that is collected for training

- Which pixels are necessary to guess the word?
- From hints: what is the relation between word and image?
- Data from pings: which pixels are inside the object?
- What are the most relevant aspects of the object?
- Elimination of poor image-word pairs

Dealing with single player games

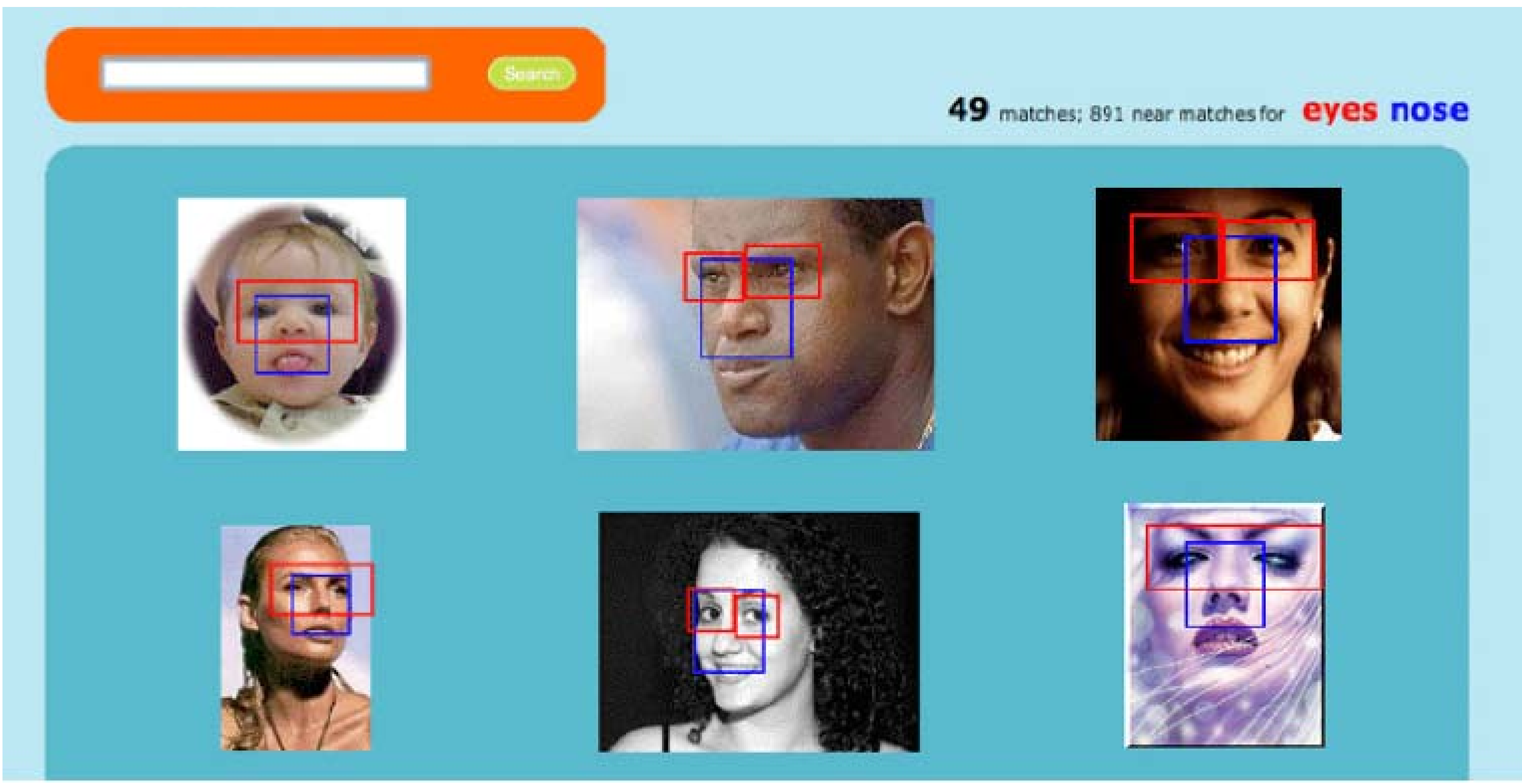
- Bot can easily simulate a Boom from previously saved human clicks
- Simulating Peek is much harder

Cheating and Collusion

- Anti-cheating mechanisms include:
  - Random matches with a matching interval when all players are matched at once
  - IP address checks
  - Blacklisting after consistent failure on "seed" images
    - By definition, bots cannot successfully play Peekaboom
  - Limited freedom to enter guesses

Object Bounding Boxes

- We can use Peekaboom results to establish tags for keywords in images
- Example below: eyes and nose



Results

- Is this an effective way to collect data?

  Yes!

- Game is enjoyable
  - Each person played average of 158.7 images
  - That's 72.96 minutes per person in one month!
    - User review: "This game is like crack. I've been Peekaboom-free for 32 hours…"

How Good is the Data?

- Bounding Boxes:
  - .754 overlap between Peekaboom-generated bounding box and bounding box generated by human volunteer
    - Note: Only looked at images with noun keywords
- Pings:
  - All the sample pings were in the object, as determined by human volunteers

Discussion

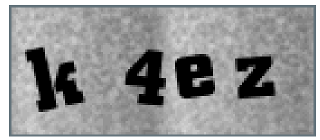
- What are some weaknesses of Peekaboom?
- Can you think of any other applications of Peekaboom?

CAPTCHA: Using Hard AI Problems for Security

By Luis von Ahn, Manuel Blum, Nicholas J. Hopper, John Langford

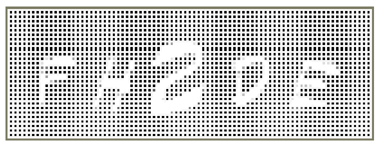
Examples of CAPTCHAs

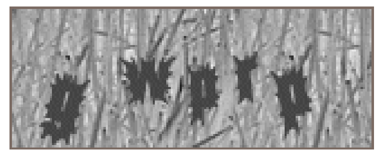
Configuration file

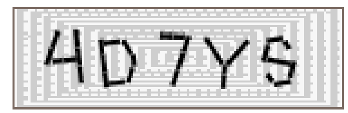


Configuration file

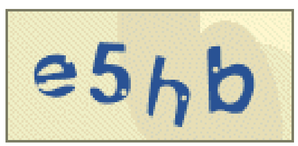
Configuration file


Configuration file


Configuration file

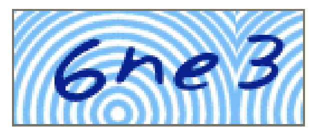


Configuration file

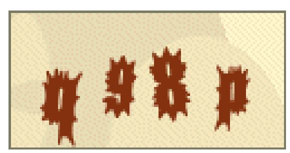


Configuration file

Configuration file


Configuration file


Configuration file

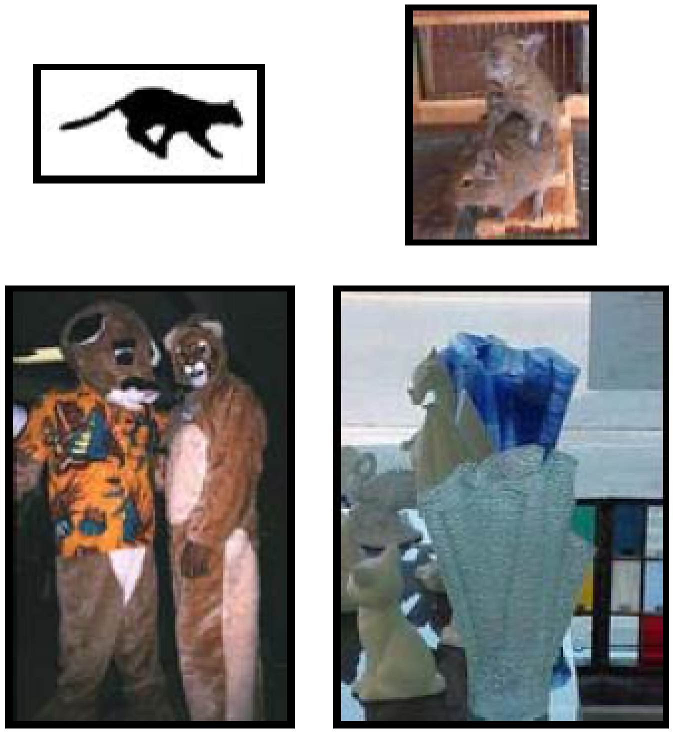

Configuration file


Configuration file

More Examples

More Examples

The CAPTCHA Project



Choose a word that relates to all the images.

TIP: You can type the first letter of a word and then use the down arrow to find it.

Submit

What is a CAPTCHA?

- A cryptographic protocol whose underlying hardness assumption is based on an AI problem
  - Certain problems, such as image / audio / voice recognition, are hard AI problems but easy problems for humans to solve
- So CAPTCHA is meant to prevent adversaries (aka bots) from performing a malicious task

What is a CAPTCHA?

- What kind of AI problems do we want to use?

1. Hard problems
   - Hard w.r.t. complexity
   - AI community agrees it's hard
2. Useful problems

What is a CAPTCHA?

- The fact that CAPTCHA is based on a hard AI problem gives us a win-win situation
  - Either someone figures out a bot to solve CAPTCHAs (solving a hard, useful AI problem)
  - Or there is a way to tell humans from computers, which is useful for security

What is CAPTCHA?

- To be useful, CAPTCHA must be generated automatically
- The adversary (in our case, the bot) knows exactly how the CAPTCHA works
  - The code for reCAPTCHA is online!
    - www.captcha.net
- The only thing that the bot doesn't know is the random number the algorithm generates
- So CAPTCHA cannot base its security on a piece of code or a database

For More Security

- Establish a time limit for the user to respond to the CAPTCHA
- Repeated CAPTCHAs – so if a bot has a .1 probability of guessing the answer, it only has a .01 probability of guessing the answer twice

Some Notation

- An AI problem is represented by

$$P = (S, D, F)$$

- S = Set of problems
- D = Probability distribution over S
- f = function from S to $\{0,1\}$

- We require that a fraction α of humans can solve the problem with probability > δ
- P is $(\delta, \tau) - hard$ if there is no program to solve the problem with probability δ in at most time τ and the AI community agrees that it's hard

Definitions

- So an (α, β, η) CAPTCHA is a test that at least α of humans can solve with probability β
- $\exists$a $(\delta, \tau) - hard$ problem *P* where if some program *B* solves the CAPTCHA with at least η probability, reduction of program B to a program A is a (δ, τ) solution to *P*
- A win-win situation – either there is no program to solve the CAPTCHA or we solve the $(\delta, \tau) - hard$ problem *P*

Two AI Problem Families

- 2 families of AI problems that can be used for CAPTCHAs: both relate to images and image transformations

- Some notation before we begin:

$$I = \text{distribution on set of images} [I]$$

$$T = \text{distribution on set of image transformation} [T]$$

$$S_{I,T} = \{t(i) : t \in [T], i \in [I]\}$$

$$D_{I,T} = \text{distribution on } S_{I,T}$$

$$f_{I,T} = fn : S_{I,T} \to [I] \text{ s.t. } f_{I,T}(t(i)) = i$$

Two AI Problem Families

- Family ***P1***:

$$(S_{I,T}, D_{I,T}, f_{I,T})$$

  - Write a program that takes in t(i) as input and outputs image I

- Family ***P2***:

$$(S_{I,T}, D_{I,T}, g_{I,T,\lambda})$$

  - Write a program that takes in t(i) as input and outputs a label λ for image i

Hard Problems in P1 and P2

- We generally have a fairly small set of images to choose from, so the hardness from hard problems in families ***P1*** and ***P2*** come from the large number of transformations t
- Listing all possible transformations should be computationally infeasible

Two Families of CAPTCHAs

- MATCHA
  - Programs that can solve MATCHA can solve anything in ***PI***
  - How it works:
    - Choose a transformation t
    - Flip an unbiased fair coin
    - Heads => pick image k and set i=k, j=k
    - Tails => pick images i,j s.t. i != j
    - Send (i, t(j)) and ask the user (the prover) to state whether i = j
    - If incorrect, then reject. If correct and i != j, play another round. If correct and i = j, accept

Two Families of CAPTCHAs

- Matcha is not very useful, since if you always return true, you have a 50% probability of being correct
- Nevertheless, if a program is success enough at MATCHA we can solve AI problems in ***P1***

Lemma 1. *Any program that has success greater than η over $M = (\mathcal{I}, \mathcal{T}, \tau)$ can be used to $(\delta, \tau|[\mathcal{I}]|)$-solve $\mathcal{P}1_{\mathcal{I},\mathcal{T}}$, where*

$$\delta \geq \frac{\eta}{1 + 2|[\mathcal{I}]|(1-\eta)}.$$

- Proof idea: From program B that is successful enough over MATCHA, construct a program that can solve the corresponding problem in ***P1***

Two Families of CAPTCHAs

- PIX
  - Being successful against PIX means that we can solve AI problems in ***P2***
  - How it works:
    - PIX program picks a random image i and transformation t
    - Sends the user (or bot) (t(i), L) where L is a set of possible labels
    - User sends back a label. If label = λ(i) where λ is the correct label, accept

Two Families of CAPTCHAs

- PIX is actually used in Gmail, Hotmail, Yahoo! etc to identify bots
- To summarize, we have 2 theorems to define how problems in ***P1*** and ***P2*** are translated to CAPTCHAs

(M is MATCHA and X is PIX)

Theorem 1. *If $\mathcal{P}1_{\mathcal{I},\mathcal{T}}$ is $(\delta, \tau|[\mathcal{I}]|)$-hard and $M = (\mathcal{I}, \mathcal{T}, \tau)$ is (α, β)-human executable, then M is a* $(\alpha, \beta, \frac{(2-|[\mathcal{I}]|)\delta}{2\delta-|[\mathcal{I}]|})$-CAPTCHA.

Theorem 2. *If $\mathcal{P}2_{\mathcal{I},\mathcal{T},\lambda}$ is (δ, τ)-hard and $X = (\mathcal{I}, \mathcal{T}, L, \lambda, \tau)$ is (α, β)-human executable, then X is a* (α, β, δ)-CAPTCHA.

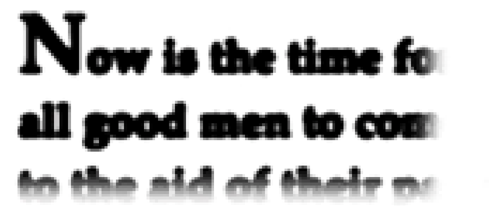
Application: Robot Steganography

- Alice wants to send an image to Bob and hide some secret information in the image
- Eve wants to transform the image to remove this information. However, Eve's transformation cannot significantly impact human interpretation of the images
- The authors create a construction of a stegosystem where Alice can transform the message (key) into an image, which is indistinguishable from regular images and is robust against transformations by Eve
  - Formally, the stegosystem is steganographically secret and robust

reCAPTCHA

- reCAPTCHA is a project to use CAPTCHAs to help digitize old books
  - Currently, Optical Character Recognition is not perfect
  - The words that the OCR is not sure about are presented in a CAPTCHA for users to solve
- The CAPTCHA will present 2 words – one known one generated and one from a digitized book
- If the user correctly identifies the word that is known, the program assumes the word from the digitized book was entered correctly as well

1 We start with a scanned book containing thousands of words.



2 Next we extract a word that cannot be read by OCR. These words come in a variety of fonts, and are inherently distorted by the age of the book or the quality of the scan.

3 To add extra security, the word is then distorted even more, using random lines and warps.

4 A CAPTCHA is generated with two distorted word images from books.

Discussion

- What are some other applications of CAPTCHA?
- Besides image/audio recognition, are there other hard AI problems that can become CAPTCHAs?
- What are some vulnerabilities of CAPTCHAs as they are implemented now?