

RECOGNISING 100k+ MTG CARDS

And tidbits along the way

By Nathan Michlo for PyData 2025-04-14

My Background

- Cool work at Dragonfruit AI
 - Full-stack ML integration and architecture.
 - Self-checkout fraud detection.
 - Perspective & lens distortion correction
 - Custom ML models & training pipelines
 - Video streaming & WebRTC
- Studied at Wits
 - Representation learning, machine learning, computer vision



What is Magic The Gathering?

- Strategy trading card game where you build an optimised deck of cards and battle against opponents.
- Over 100,000 cards with different art in circulation.
- 1000s of new cards are released each year.



The Project

- "I want to point my phone at MTG cards and instantly and accurately get all information about them"

Black Lotus0



Mono Artifact

Adds 3 mana of any single color of your choice to your mana pool, then is discarded. Tapping this artifact can be played as an interrupt.

Illus. © Christopher Rush

Black Lotus0

Artifact

 Sacrifice this artifact: Add three mana of any one color.

Illustrated by Christopher Rush

Part of the Reserved List

NOT LEGAL

Standard

NOT LEGAL

Pioneer

NOT LEGAL

Modern

BANNED

Legacy

RESTRICT.

Vintage

BANNED

Commander

BANNED

Oathbreaker

NOT LEGAL

Alchemy

NOT LEGAL

Explorer

NOT LEGAL

Historic

NOT LEGAL

Brawl

NOT LEGAL

Timeless

NOT LEGAL

Pauper

NOT LEGAL

Penny

Canadian Highlander: 7 points

U

Unlimited Edition (2ED)
#233 · Rare · English · Nonfoil

EN

PRINTS

USD

EUR

TIX

★

Vintage Championship #2017NA

★

Magic Online Promos

★

Vintage Masters43.85

★

Vintage Championship #2003

★

Oversized 90's Promos

★

Intl. Collectors' Edition€1,858.92

★

Collectors' Edition\$2,699.72€1,831.63

★

Unlimited Edition€16,752.59

★

Limited Edition Beta€30,194.40

★

Limited Edition Alpha€12,880.60491.06

View all prints →

[1] <https://scryfall.com/card/2ed/233/black-lotus>

> ./demo





Total Value: \$0.02



Thriving Ibex 3⚡ \$0.02

Thriving Ibex 3⚡

Set: Kaladesh (old)

Type: Creature — Goat

Price: \$0.02

When this creature enters, you get ⚡⚡ (two energy counters). Whenever this creature attacks, you may pay ⚡⚡. If you do, put a +1/+1 counter on it.

Thriving Ibex

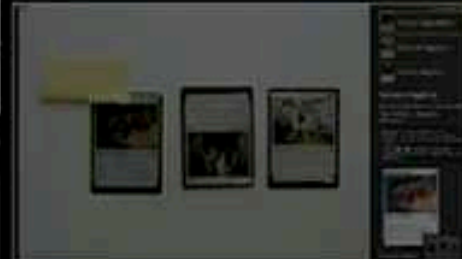


Creature — Goat

When Thriving Ibex enters the battlefield, you get ⚡⚡ (two energy counters). Whenever Thriving Ibex attacks, you may pay ⚡⚡. If you do, put a +1/+1 counter on it.

Textiles at the Inventors' Fair feature threads from every corner of Kaladesh.

Match Score: 0.97910976



And a video just in case... Because we all know how live demos go...

Why is This Hard?

- Over 100,000 different cards in circulation.
- Cards can be very similar, minor change of a small **set symbol**, or text.
- 1000s of new cards are released each year
 - Typical classification approaches don't work!



Lets Make It Harder!

- **Zero retraining** when new sets of cards are released. All we should have to do is update the DB.
- Must perform in **near-realtime**.
- "Why settle for 15 minutes of real-world data labelling, when I can burn a few weeks building an infinite **synthetic data generator**?"

... As the age-old adage goes ...



Examples of synthetic data

Q: So what's needed?

What's Needed - The Main Parts

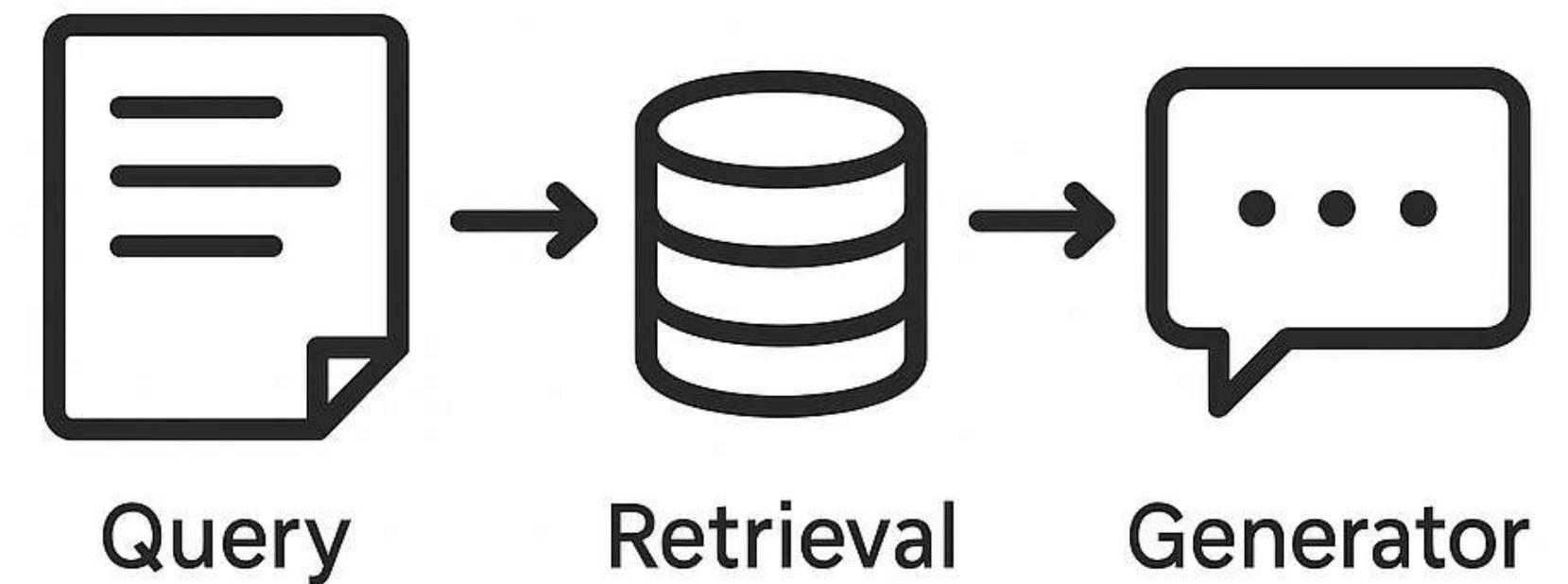
- **Data Collection**
- **Card Detection**
 - Object detection / Instance segmentation
 - Custom dataset
- **Card Embedding**
 - Embedding model
 - Custom dataset
 - Representation learning
- **Card Matching**
 - K Nearest Neighbours



Q: Related fields?

Why - Related Techniques and Projects

- Facial recognition
- Recommender systems
- RAG - retrieval augmented generation
- Object tracking & re-identification
- 2D/3D graphics & perspective correction
- Deploying models for edge devices
- Training models for AR / VR



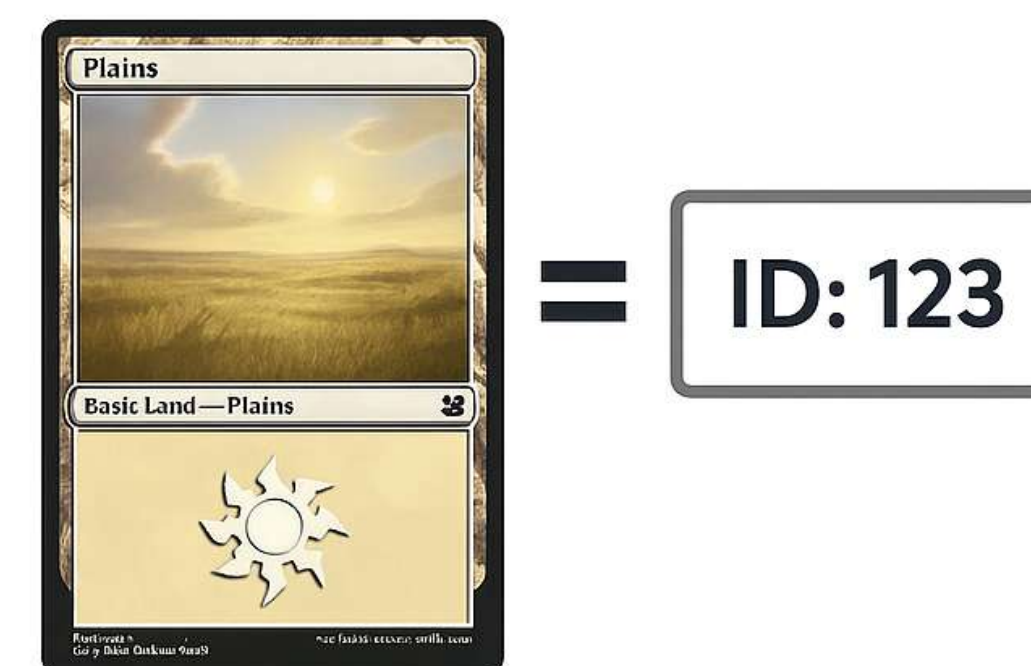
```
> ./lets-dive-in --top-down
```

1. Card Recognition

Searching For Cards By Fingerprinting

- A unique ID is given to each card.
 - its easy to look this up **if** we know it.
- What if we don't know the ID? But we have an image (or data)?
 - First **generate a "fingerprint"** or embedding of the image.
 - Compare with all known fingerprints **the most similar one** is the match
- Usually we do this with an **encoder neural network**

ID / UUID

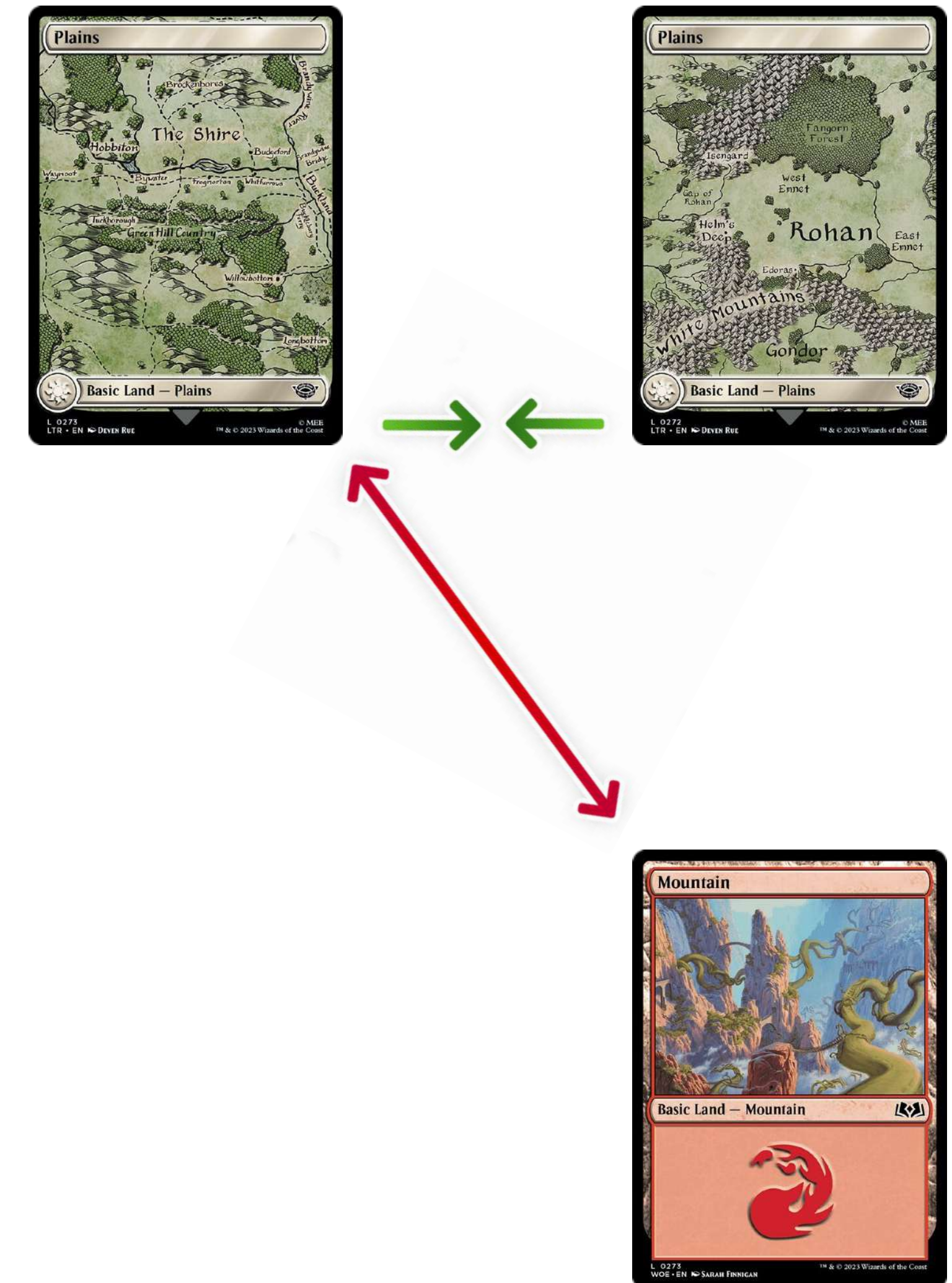


Fingerprint /
Embedding



What Makes Good Fingerprints/Embeddings

- Fingerprints/embeddings are vectors or fixed size lists of numbers.
- When comparing the embeddings from two different cards
 - **Dissimilar cards** should have **dissimilar embeddings**
 - **Similar cards** should have **similar embeddings**
- **Similarity is subjective**
 - In our case, lighting conditions and environmental factors shouldn't impact embeddings of cards



Efficient Vector Search

Assuming we have card "fingerprints"

- qdrant ("Quadrant") is a modern vector database and similarity search engine
 - Server written in rust with python client library
- Operates on "**points**" containing
 - **ID + payload + vector**
- Can search for nearest vectors WHILE also filtering on payloads
 - e.g. only return nearest "land" cards
- Easily run server with docker

```
from qdrant_client import QdrantClient, models

client = QdrantClient(url="http://localhost:6333")

client.upsert(
    collection_name="{collection_name}",
    points=[
        models.PointStruct(
            id="5c56c793-69f3-4fbf-87e6-c4bf54c28c26",
            payload={
                "color": "red",
            },
            vector=[0.9, 0.1, 0.1],
        ),
    ],
)
```

Insert Points

```
client.query_points(
    collection_name="{collection_name}",
    query=[0.2, 0.1, 0.9, 0.7], # <--- Dense vector
)
```

Find Nearest Points

Vector Search Use Cases

- Advanced Search / Image Search
- Re-identification
- Recommendation Systems
- Retrieval Augmented Generation
- Data Analysis
- AI Agents



The Real-World Vector Search Problem

- "Fingerprinting" a card needs to:
 - Produce the same embeddings under different conditions so that search is successful.
- Distortions include:
 - motion blurring, occlusion, pixelation, compression artefacts, lighting, bad warping, background images, etc
- e.g. for text, different phrasings of the same sentence mean the same thing.



Jumping ahead here. Real-world video of a card being detected in real-time as it or the camera moves.

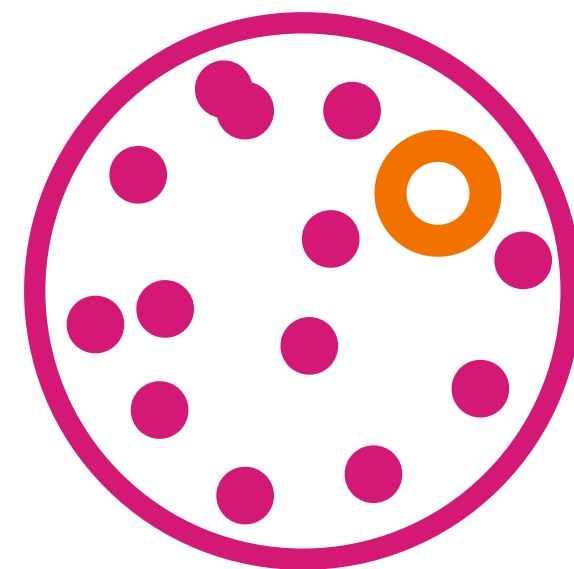
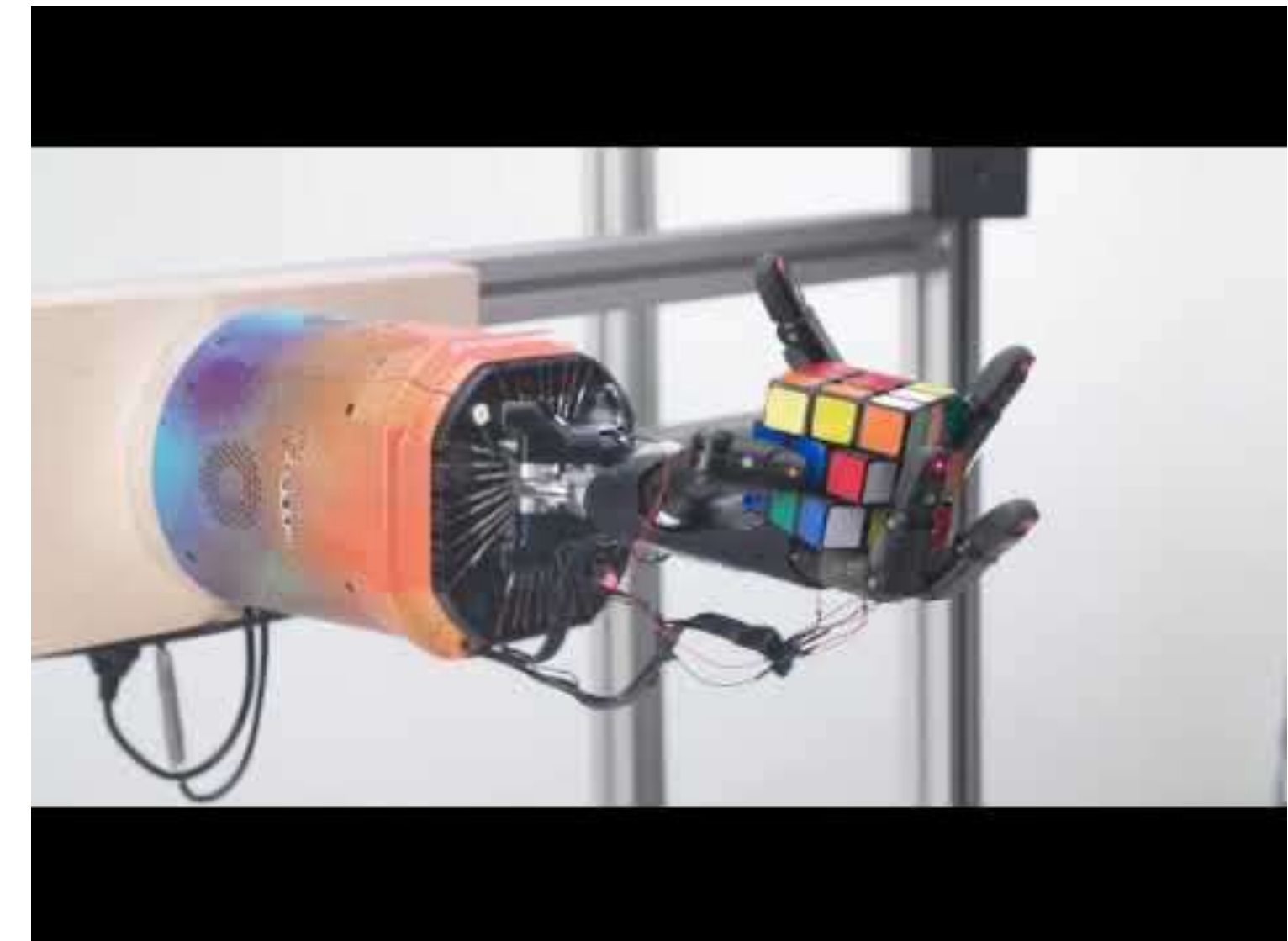
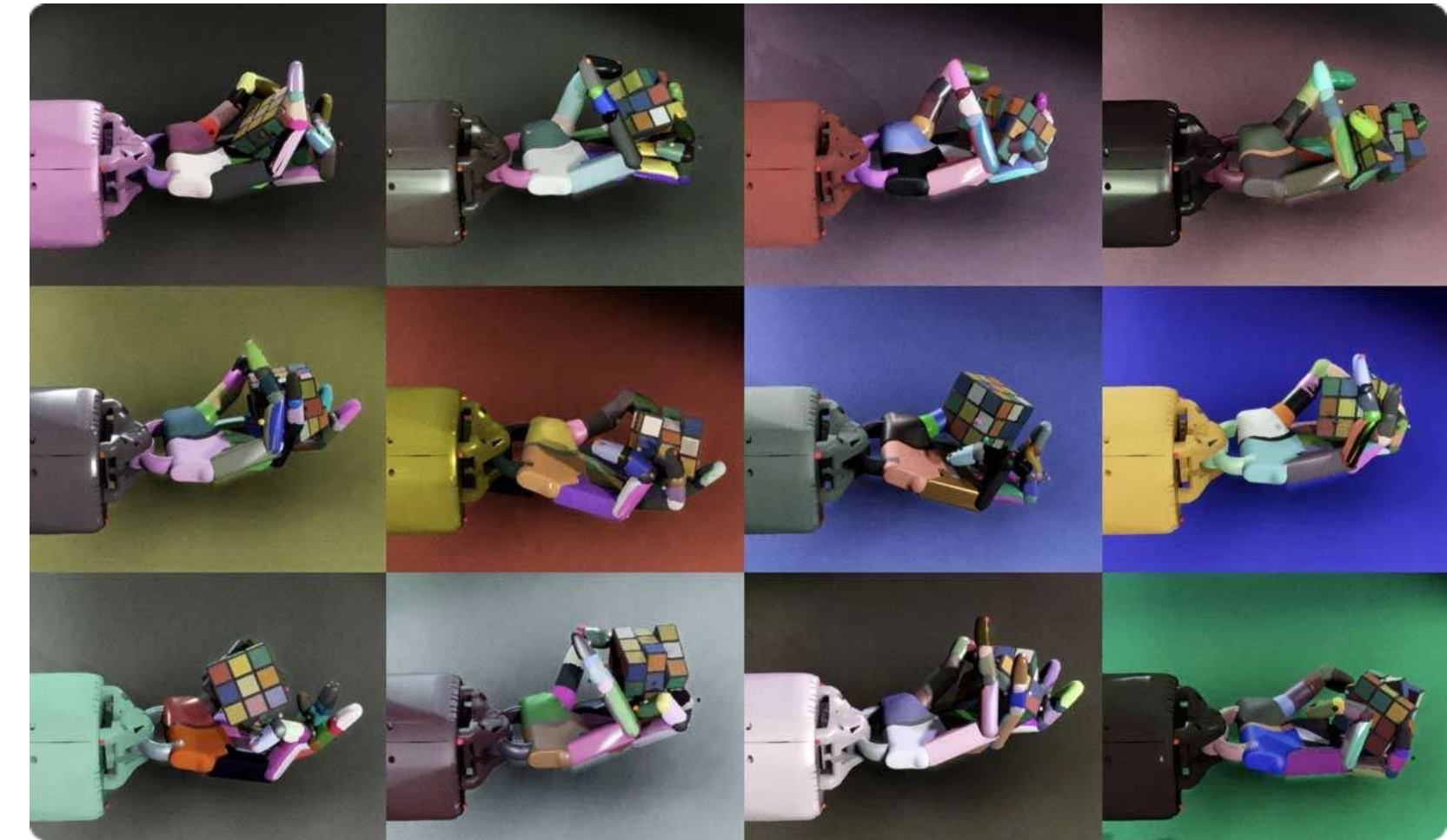
2. Card Embeddings

"Machine learning is making a magic box do its best to output what you want based on **what you want to give it as input.**"

- My 4ish second summary of machine learning

Using Synthetic Data In The Real World

- Can we generate synthetic data that lets models transfer to real-world?
 - A good example is OpenAI's method for training a real robot hand using random synthetic data.
- If there is enough **random variation**, hopefully the **real world** conditions are somewhere inside that.



Generating Synthetic Data - Images

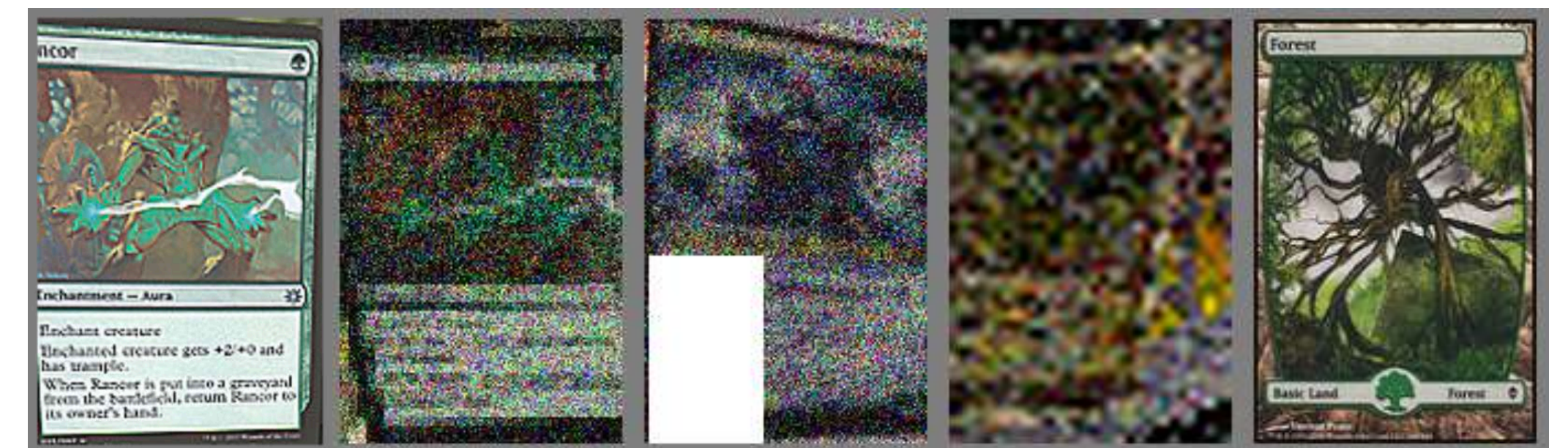
- Assuming a dataset of perfect undistorted MTG cards
- Randomly augment cards**
 - colour, tint, brightness, contrast, noise, JPEG compression, masking, downscaling, blurring, sharpness, etc.
 - Anything that might be present in a camera feed looking at a card.
- Perspective warp onto **random background images**, that are also augmented.

```
$ pip install albumentations  
$ pip install opencv-python
```

Jumping ahead here, but remember this is what we are trying to mimic



Perfect Inputs



Synthetic Inputs

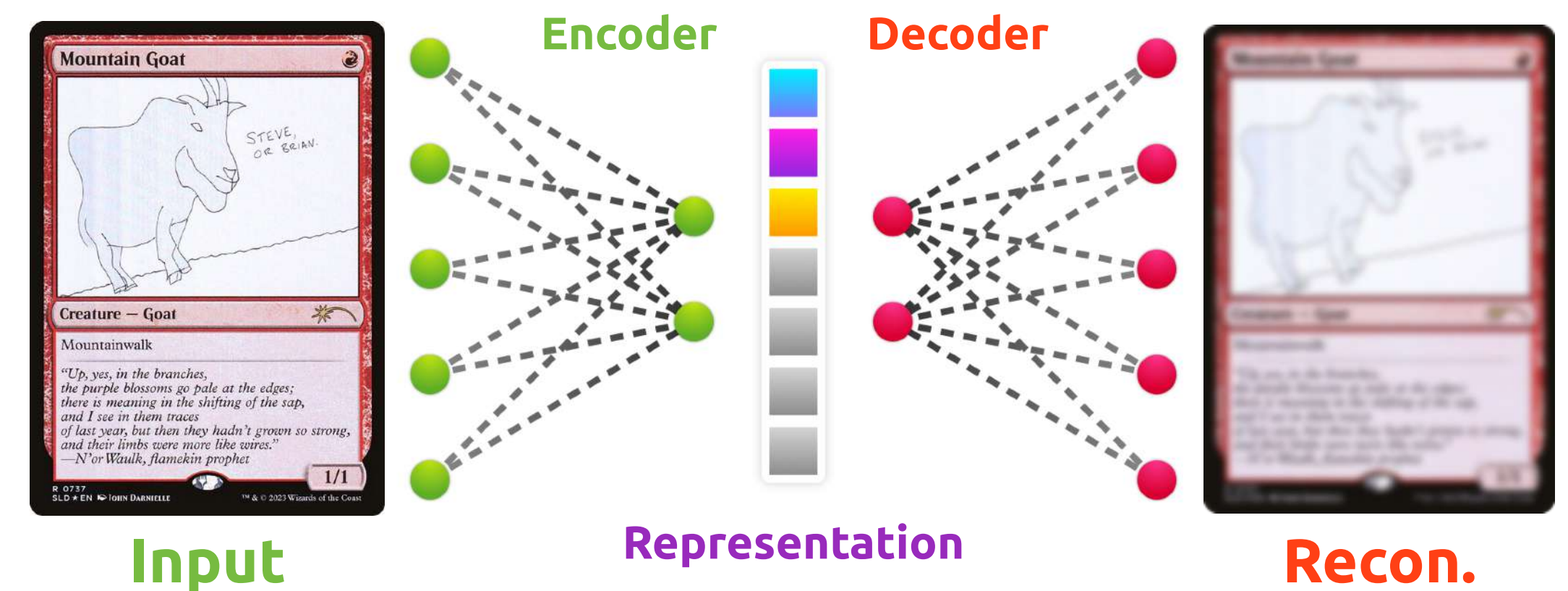
"Machine learning is making a magic box do its best to **output what you want** based on what you want to give it as input."

- A recap of my 4ish second summary of machine learning

**Q: Given distorted inputs of the same cards,
can we output matching fingerprints?**

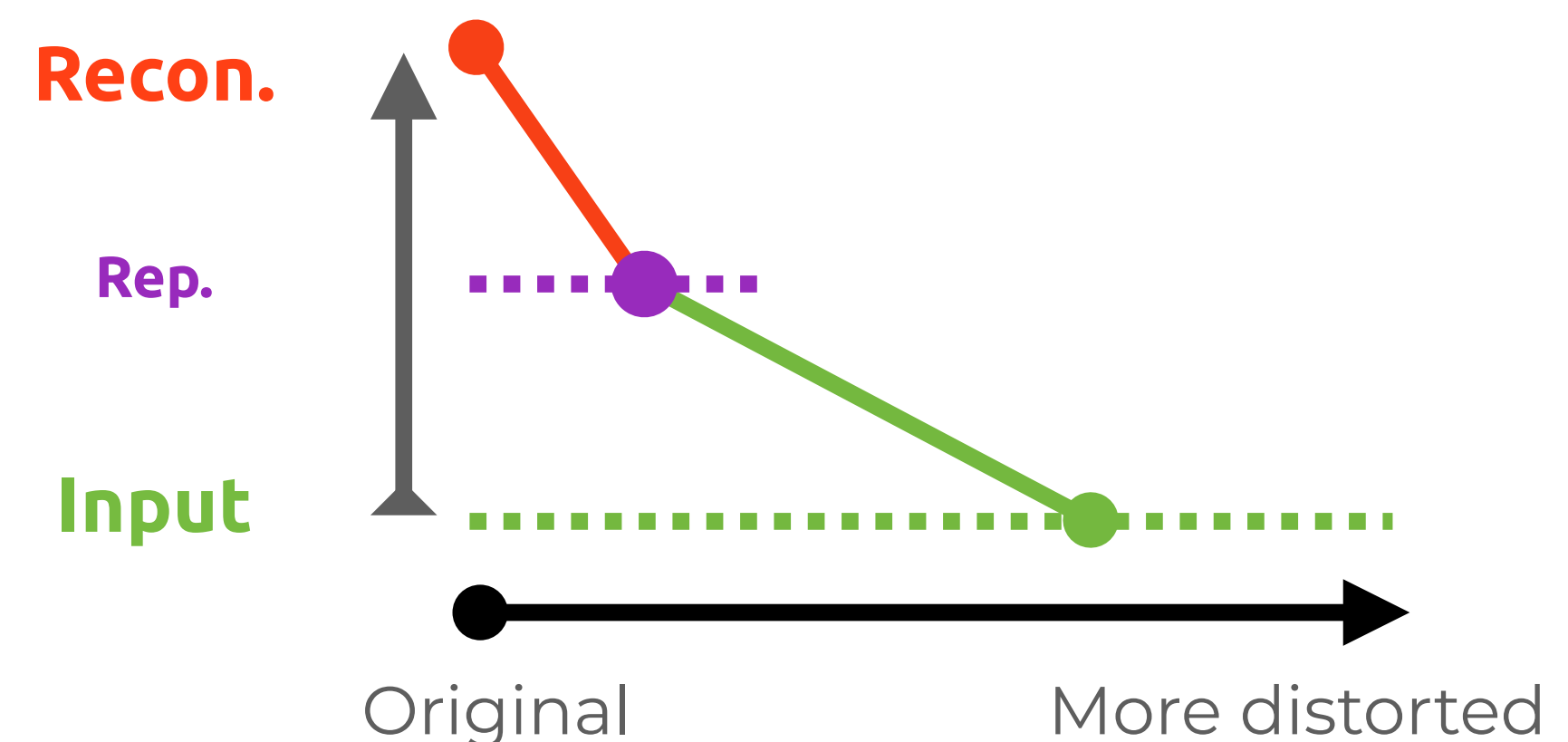
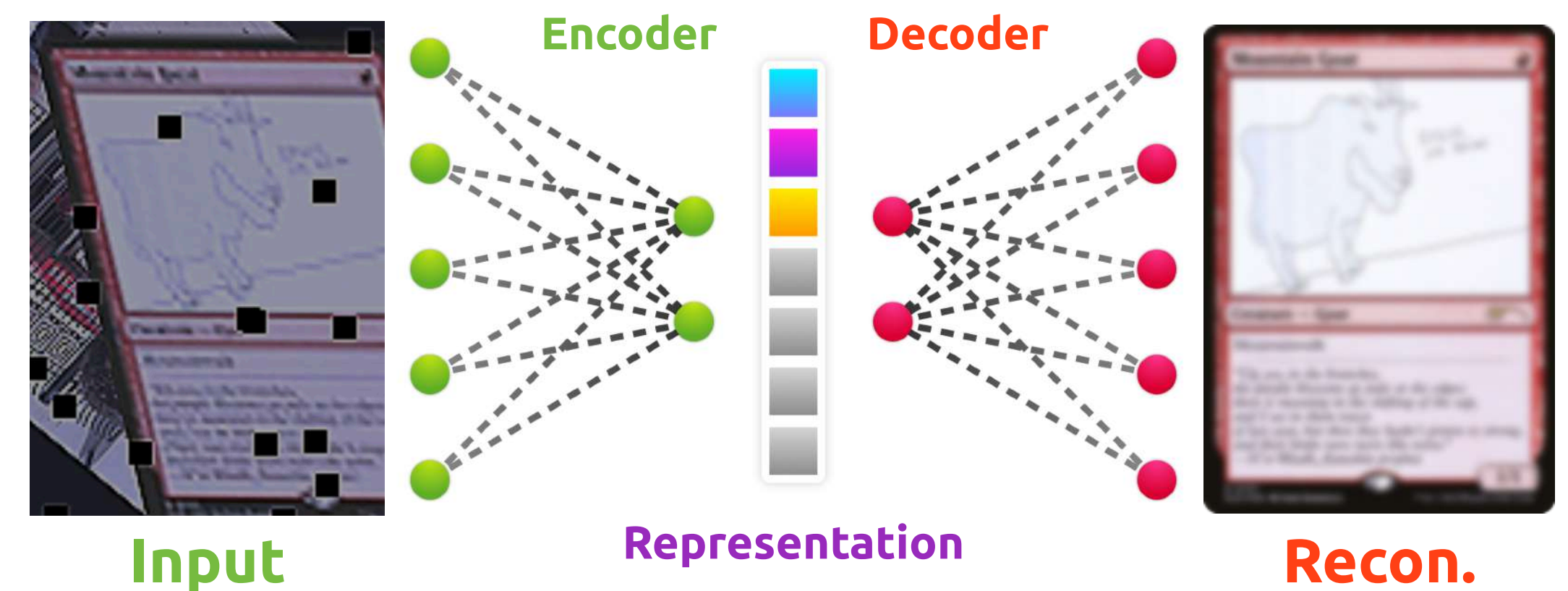
Representation Learning - Auto-Encoder

- **First** we need a way of **learning representations** to use with vector search.
- "letting machines discover the best way to describe data to solve a problem".
- Usually **encoding / compressing** high dimensional data into few dimensions.
- **Easiest way** is with an auto-encoder.
 - Output must be same as input, **BUT** constrained by bottleneck in middle.



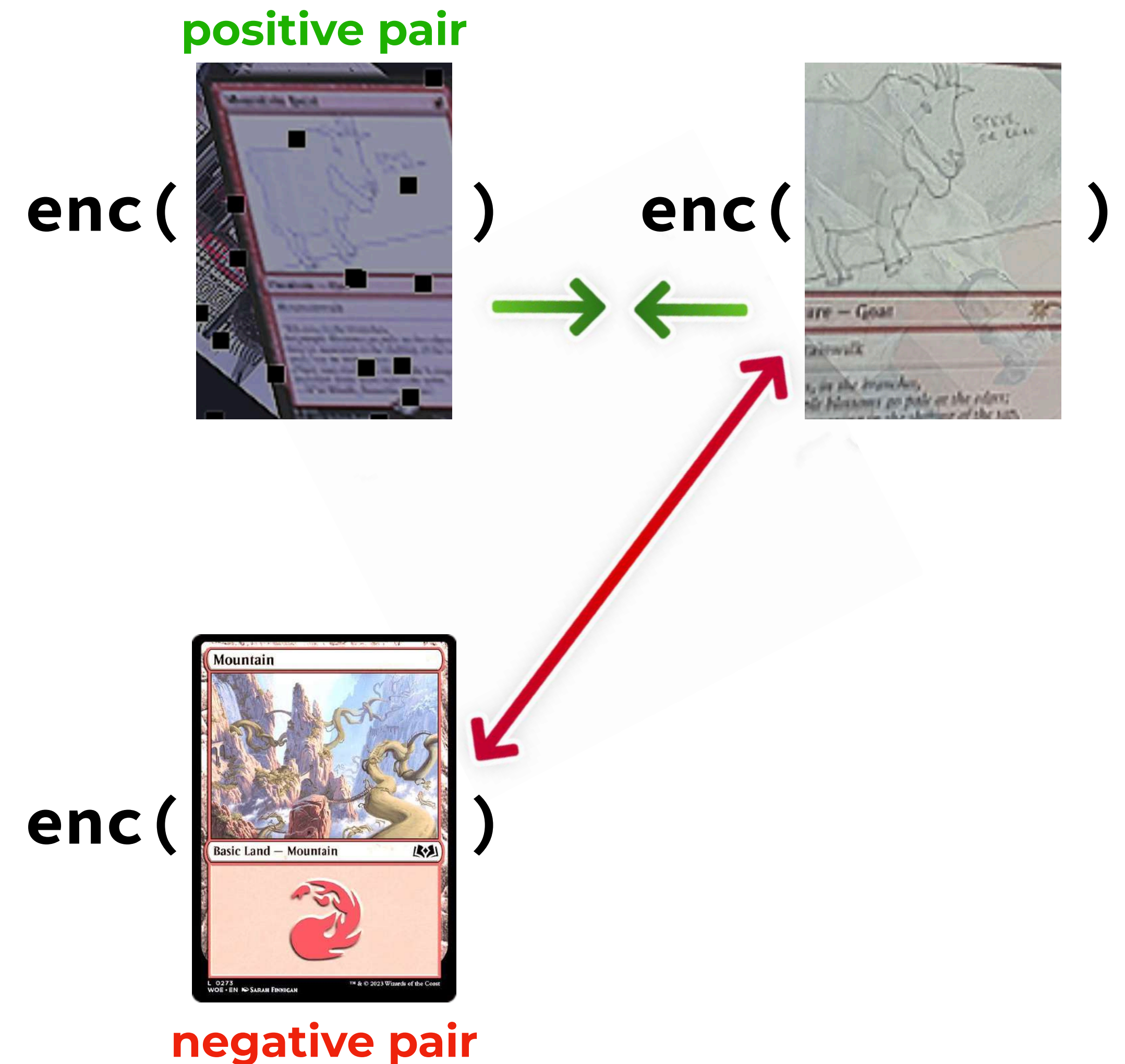
Representation Learning - Denoising AE

- **Problem:** previous example still didn't handle distorted inputs.
- Can use a **denoising auto-encoder**.
 - Make the model learn to output the perfect image given a distorted input.
 - Representation should then also be invariant to noise.
- **Doesn't work super well** in practice. No direct requirements for representations:
 - To be the same for the same cards.
 - To be close when cards are close.



Representation Learning - Contrastive

- Contrastive Learning:
 - **Pull embeddings** of similar pairs close **together**.
 - **Push embeddings** of dissimilar further **apart**.
- Can discard the decoder in most cases.
 - More efficient training.
 - But we can no longer generate pretty progress output images.



Encoder Model & Training

- Efficiently encode images and obtain embeddings
 - Used modified **ConvNeXt V2** (nano, 8ms)
 - Encoders are some of the most used neural networks. Many repos exist.
- Implemented with
 - **pytorch**
pytorch-metric-learning
- Loss chosen to be **CircleLoss** works very well for **facial recognition** without much tuning

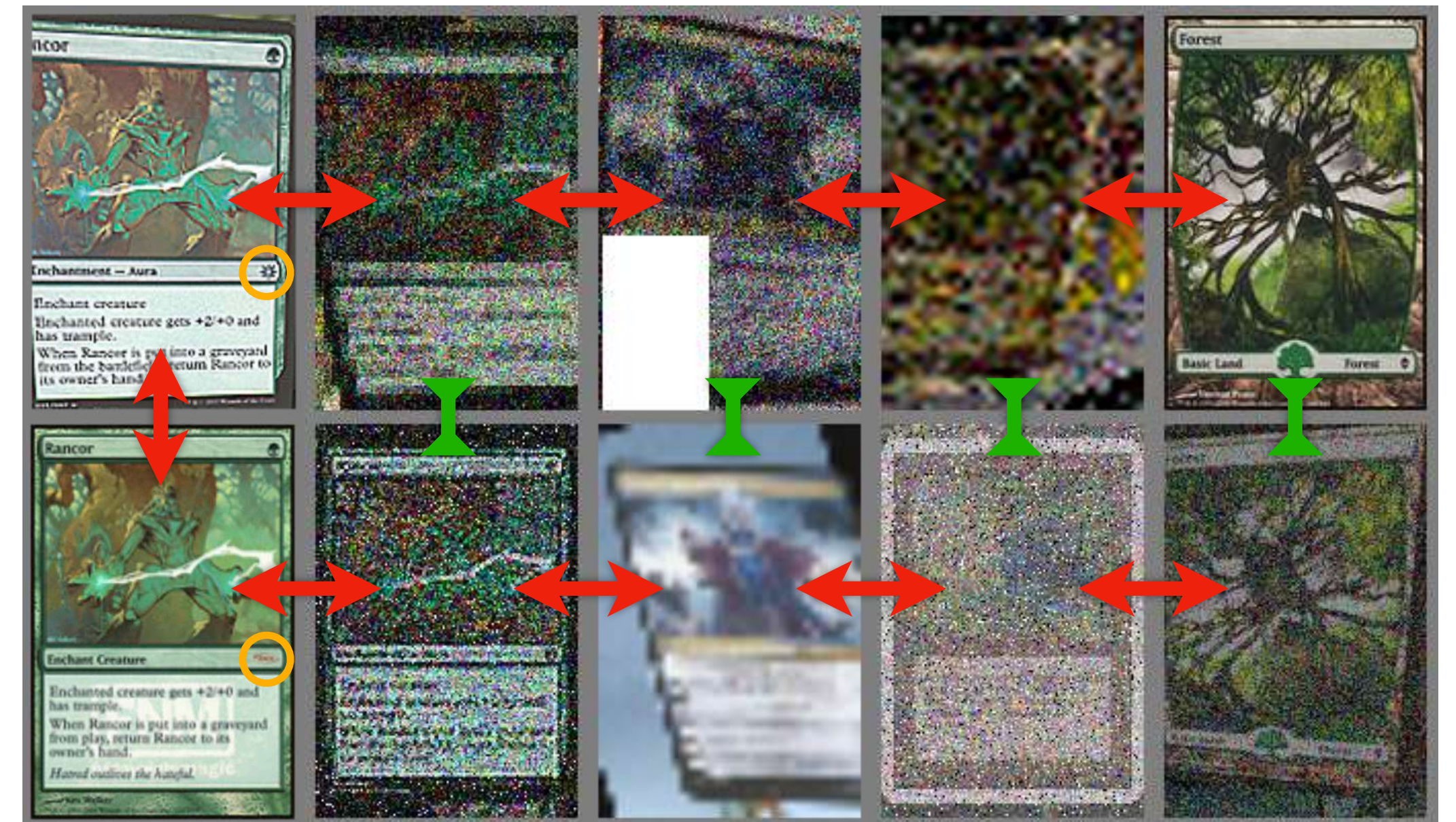
```
from pytorch_metric_learning.losses import CircleLoss

loss_fn = CircleLoss()
# pull card 0 closer to 1 and push it away from 2
loss = loss_fn(
    embeddings=[enc(card_0), enc(card_1), enc(card_2)],
    labels=[0, 0, 1],
)
# ... use loss to update encoder ...
```

\$ pip install pytorch-metric-learning
<https://github.com/KevinMusgrave/pytorch-metric-learning>

Encoder Training - Generating Batches

- Create batches
 - Augment each random image twice, resulting in **positive** + **negative** pairs.
 - Perfect for contrastive learning.
- **Problem**
 - With 100k cards, probability of batches with similar cards is too low. So model doesn't learn to differentiate.
 - We need to do **hard negative mining**, randomly swap out cards for cards with the same name.



Neural networks like cheating, or taking the easiest route to solve a problem. Don't truly understand the concept of a card.

Experiment Tracking With Weights & Biases

- Easily track experiments
 - Easily log any metrics or images and track performance over time
- Compare runs to find the best

```
import random
import wandb

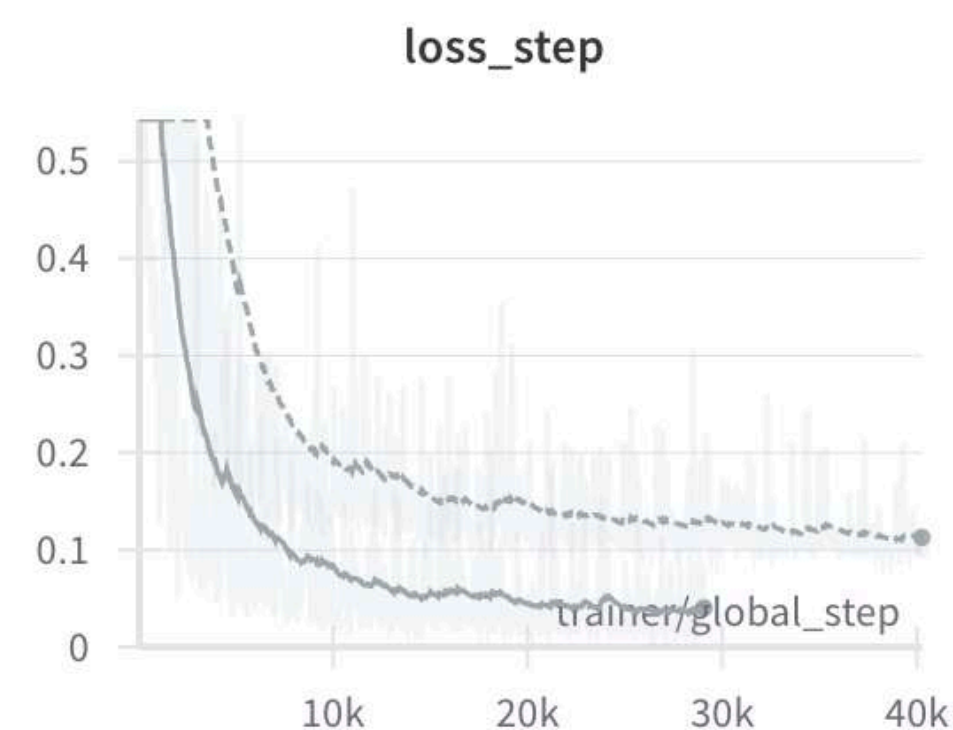
cfg = {"lr": 0.01, "epochs": 10}

# start experiment and track hyperparameters
run = wandb.init(project="my-awesome-project", config=cfg)

# dummy experiment
for epoch in range(cfg['epochs']):
    wandb.log({
        "loss": (cfg['epochs'] - epoch) * cfg['lr'] + random.random()
    })
```

`$ pip install wandb`

<https://github.com/wandb/wandb>



Step  807 ^ v

THIS IS YOUR MACHINE LEARNING SYSTEM?

YUP! YOU POUR THE DATA INTO THIS BIG
PILE OF LINEAR ALGEBRA, THEN COLLECT
THE ANSWERS ON THE OTHER SIDE.

WHAT IF THE ANSWERS ARE WRONG?

JUST STIR THE PILE UNTIL
THEY START LOOKING RIGHT.



The Real-World Embedding Problem

- How do we obtain the input image that is fed to the encoder from a camera feed?
- How do we **localise** onto individual cards, **extract** those, then feed them into the encoder in a **consistent** way?



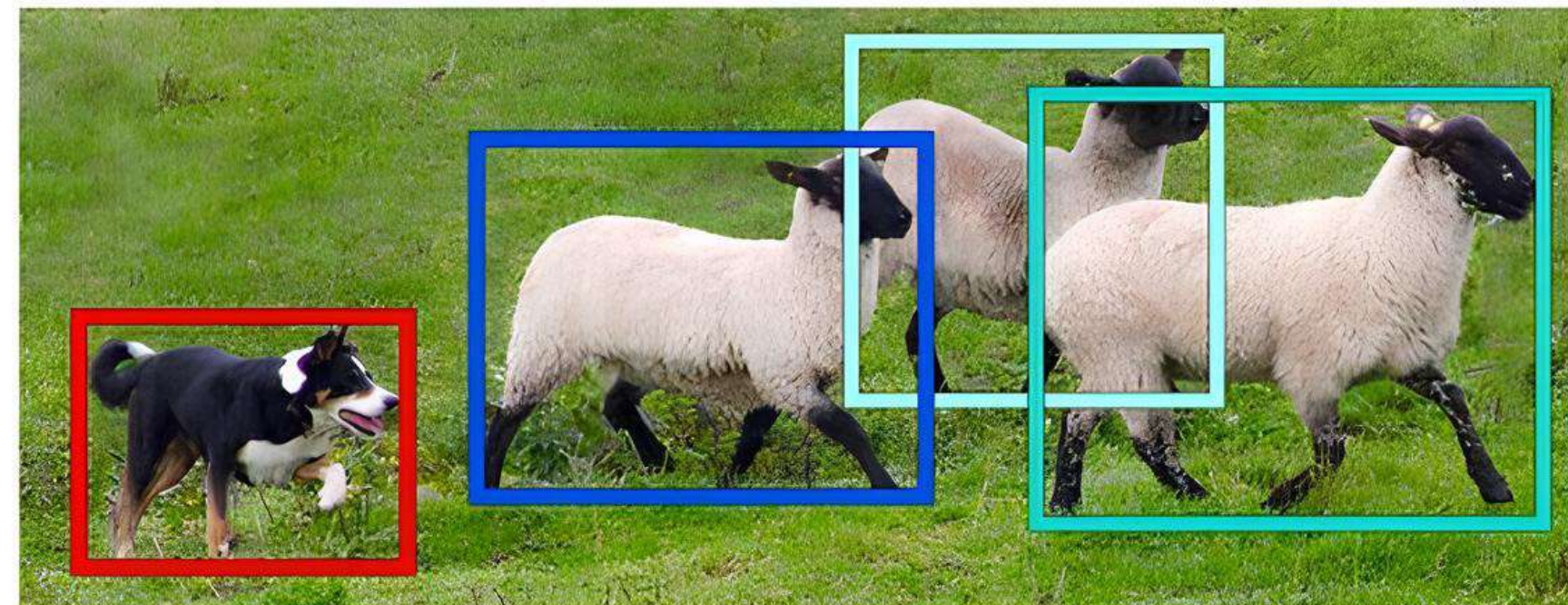
Remember this?

How do we turn **this**, into **these**



3. Card Detection

Q: Object detection... right?

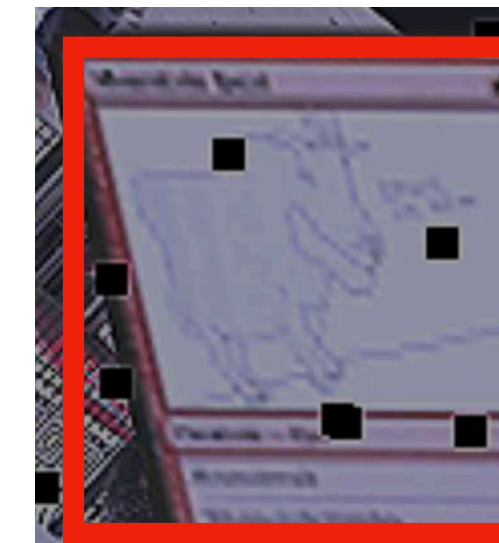


Object Detection

Our Card Detection Requirements

- Card recognition **needs the whole card** even if occluded
 - Aspect ratio should not change too much if contents extracted
- Should preferably have a perfect bounding box.
- Card recognition **needs oriented cards** to be effective
 - Could train the encoder to be invariant to rotation but then need a larger model.

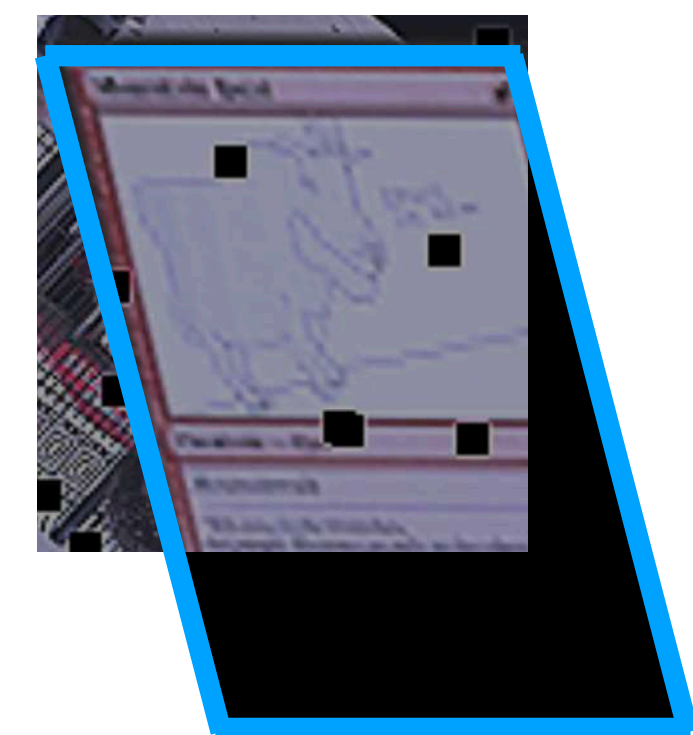
Worst



Better



Even Better

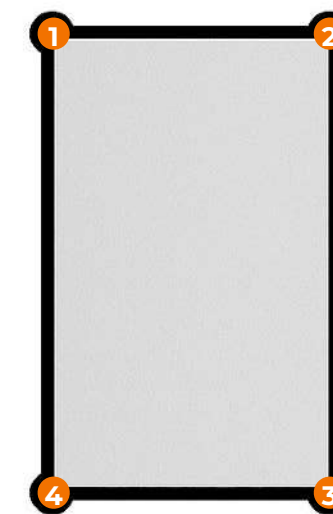
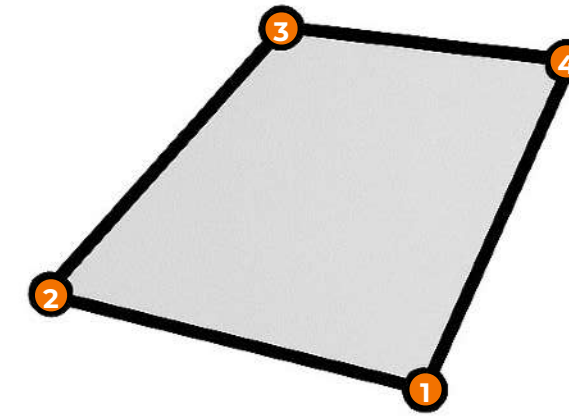


Best



Warping Images

- Perspective transform
 - Straight lines remain straight
- Calculate using:
 - 4 points on the source image
 - 4 points on the destination
 - `m = cv2.getPerspectiveTransform(src, dst)`
- Apply using
 - `cv2.perspectiveTransform(pts, m)`
 - `cv2.warpPerspective(img, m, (w, h))`



`$ pip install opencv-python`
<https://github.com/opencv/opencv-python>

Card Orientation - Approaches

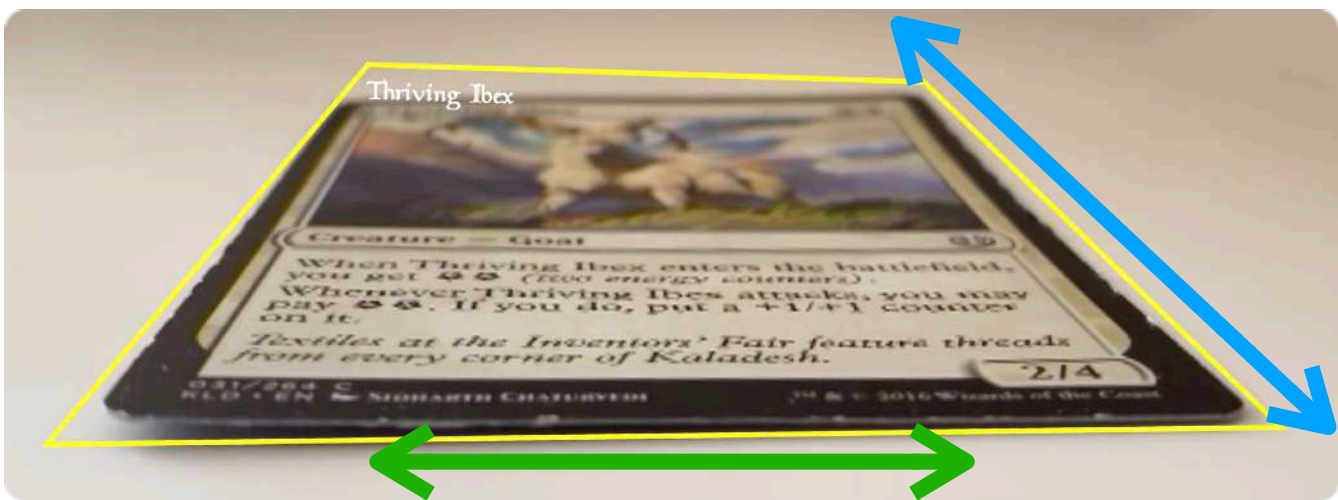
What doesn't work well

A. Just Embed Everything



Too compute heavy.
Not exact.

B. Aspect Ratio



Width < Height

Aspect can easily
be wrong if viewed
at a slight angle

Doesn't handle perspective.
Ultralytics does minimal OBB
instead of truly oriented.

C. Oriented Bounding Boxes



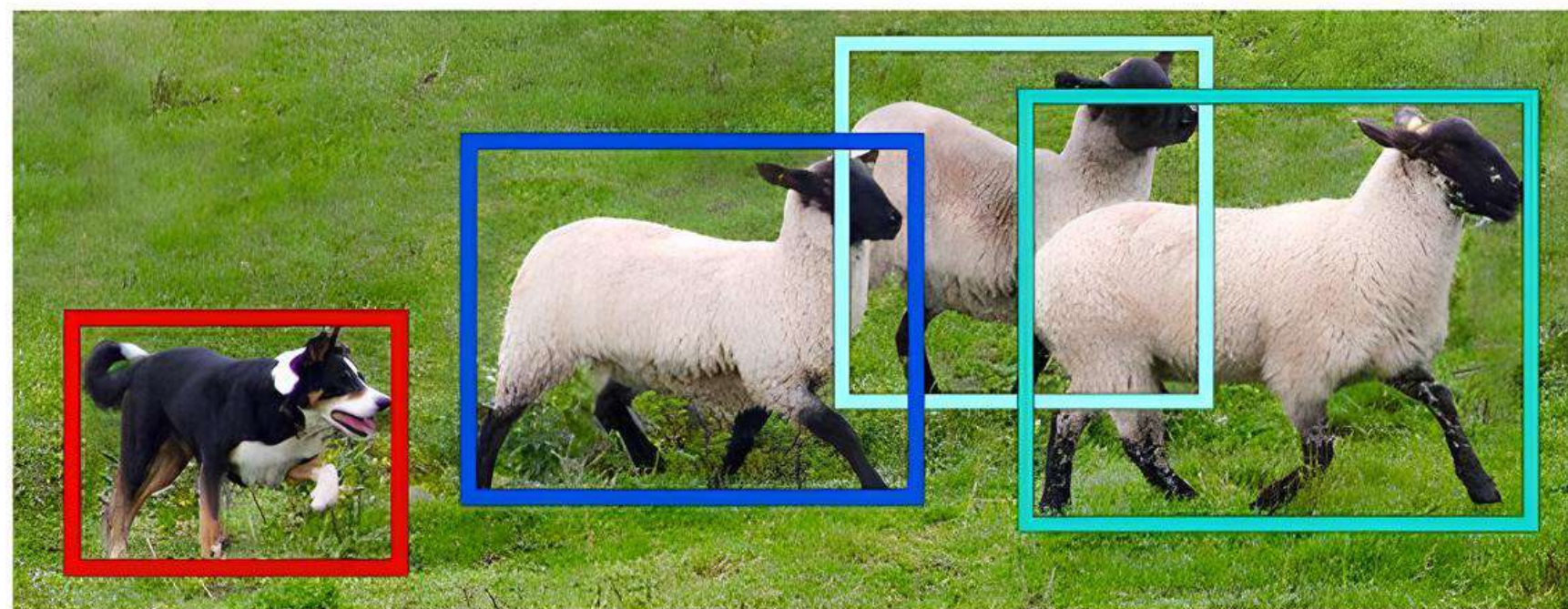
Centre X + Width + Rotation
Centre Y + Height

D. Inner boxes

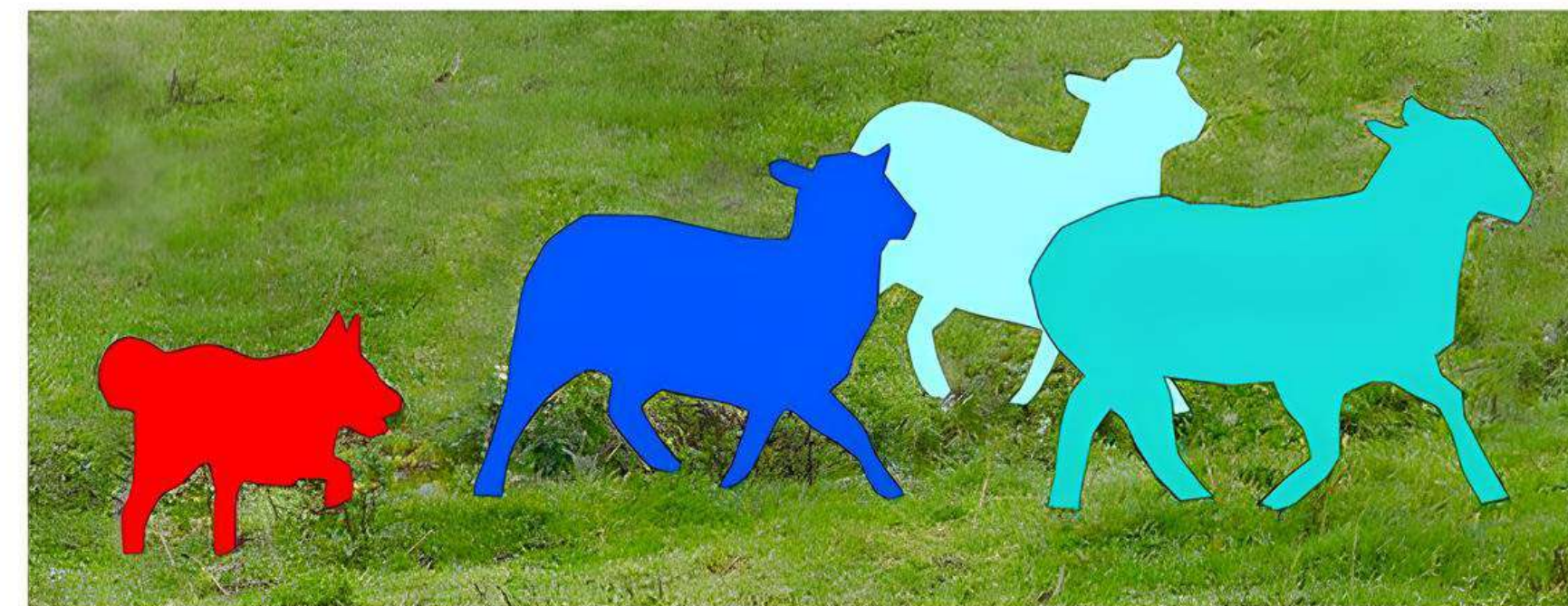


Difficult to do
association reliably
with main bbox

Q: What if we used instance segmentation instead?



Object Detection



Instance Segmentation

Card Orientation - Instance Segmentation

A hacky way to orient the card: cut a bit out the side



1. Segment
YOLO



2. Close
dilate, erode



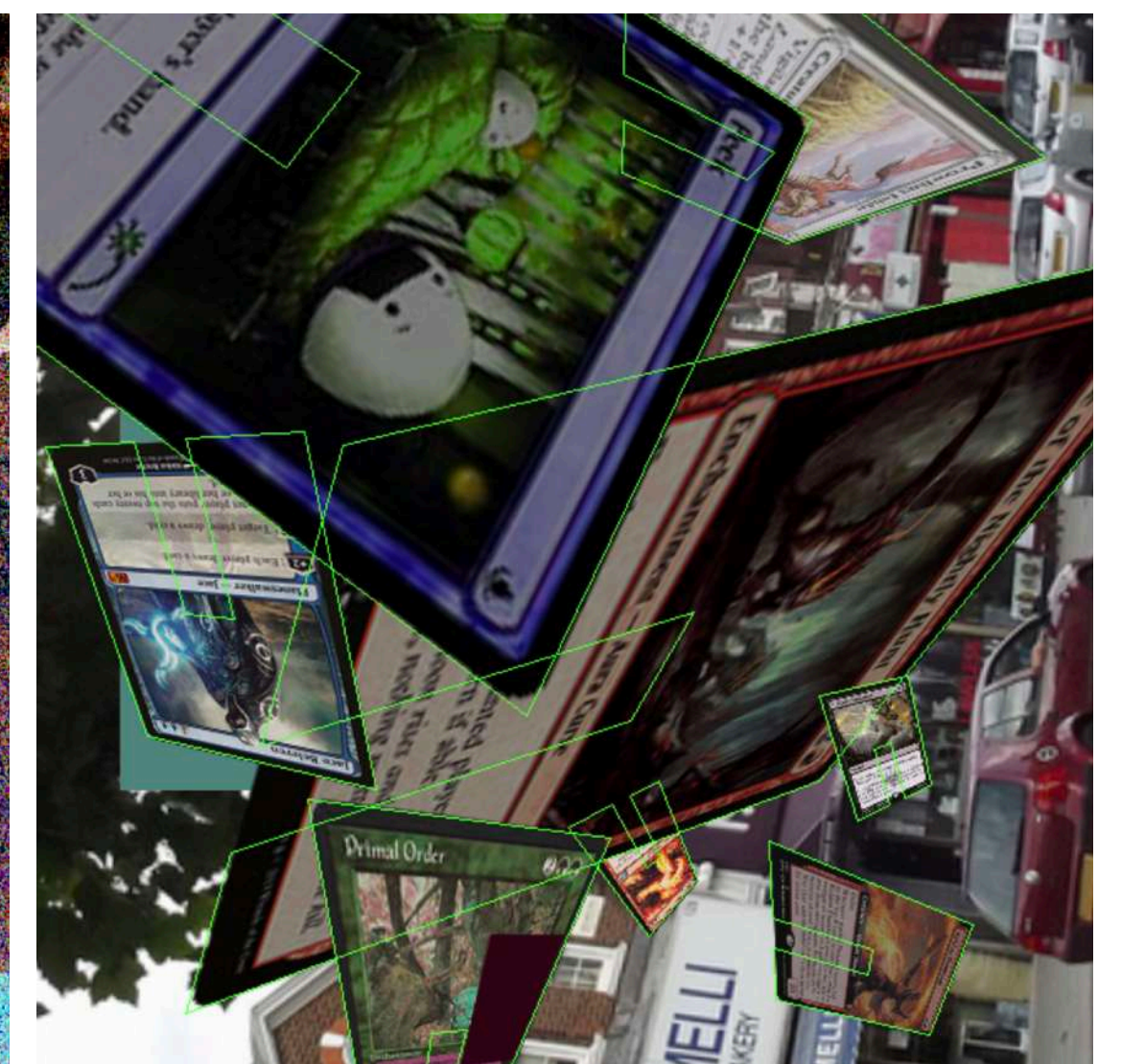
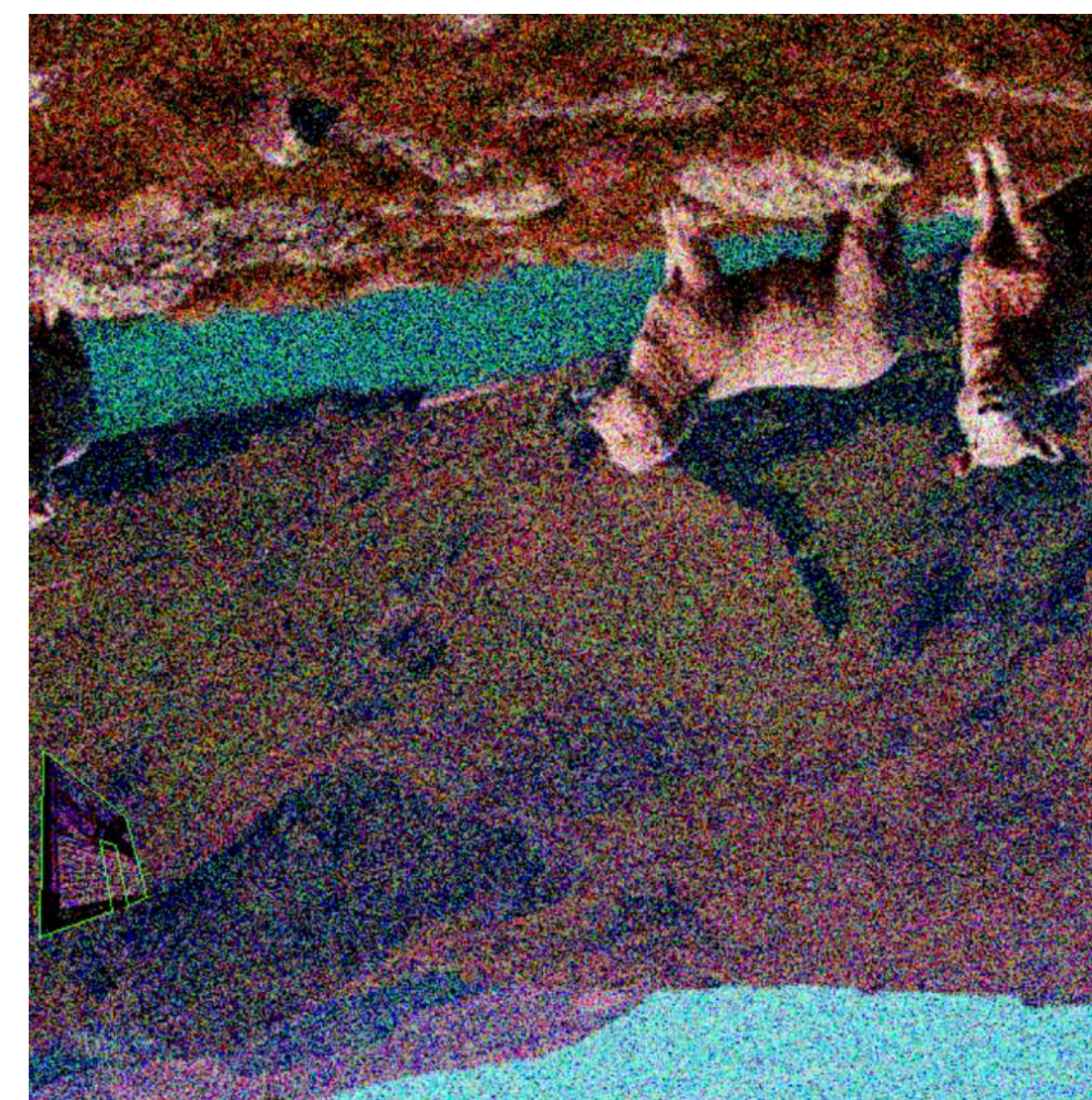
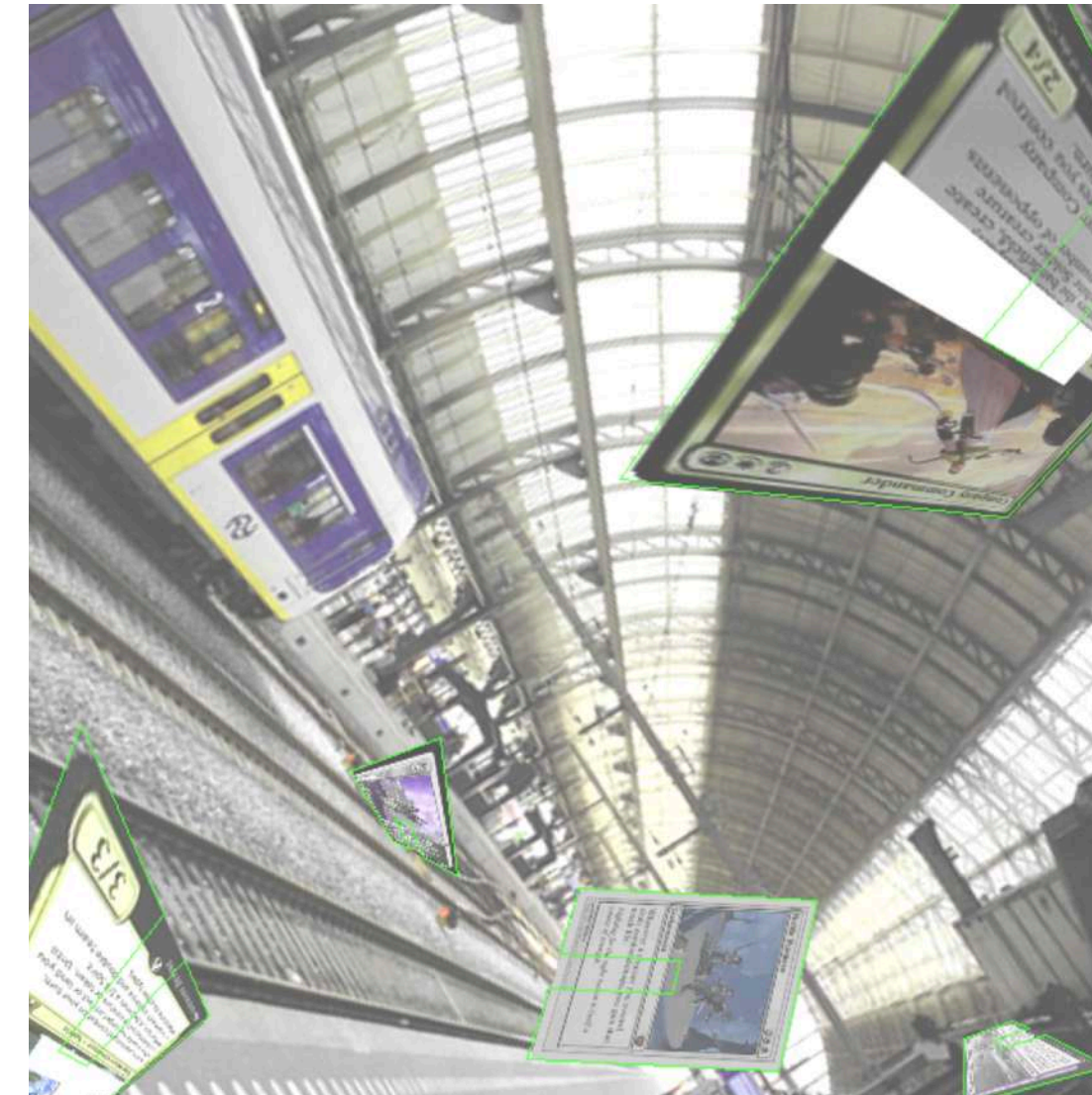
3. Simplify
`cv2.approxPolyN`



4. Orient
extend line out
from centres

Generating Synthetic Data

- Similar to embedding images, but now we **warp multiple images** onto a single background
 - Use **rejection sampling** to make sure cards don't overlap too much.
- Augment background, cards, and final combined images. Similar augments as before.
- Also need augment segmentation mask along with original card.



A `$ pip install alumentations`
<https://github.com/alumentations-team/alumentations>

Training Instance Segmentation

- Use Ultralytics yolov11
 - nano model, 8ms inference
- Data format is class ID followed by segmentation mask coordinates
- **Ultralytics problems**
 - Not ideal, doesn't allow coordinates that are out of bounds

```
from ultralytics import YOLO

# Load a pretrained YOLO11 segment model
model = YOLO("yolo11n-seg.pt")

# Train the model
results = model.train(data="path/to/your_dataset.yaml", epochs=100, imgsz=640)
```



```
0 0.8052840365561655 0.9551909360731555 0.7926695257620324 0.8555264448850087 0.71060104370
0 0.731794165967751 0.34959054011677926 0.615232086012237 0.42587809589152414 0.71699738502
0 -0.049709883155035856 0.8199917821523446 0.2517199302895455 0.8189194703415223 0.4220900
0 1.0484192940890293 -0.018184238822183747 0.6481082436440145 -0.15103024356150838 0.709
0 -0.030593335420855045 0.6530556554068088 0.6021757166514334 0.473334741156766 0.2319232
0 0.4800438377199594 0.8330537688485876 0.8185038586840729 0.4725312293725312 0.37644214630
```

The Final Problem

- Where do we get all source data to generate the synthetic data?
- With 100,000 cards, collecting this manually would...
 - Take 11 days at a generous 10s per card.
 - Cost over \$335,943 (R6.4 million) if we don't borrow them



4. Data Collection

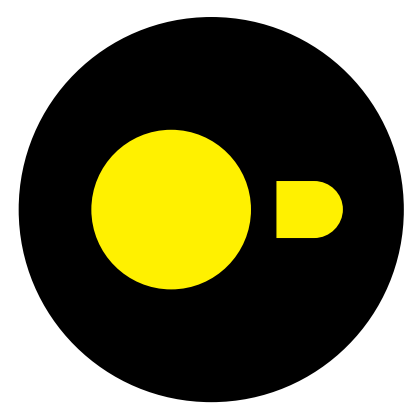
Scraping Card Information

- Luckily community maintained services like scryfall.com exist, providing a free API for accessing card information
 - **Bulk information**
 - api.scryfall.com/bulk-data
 - **.json** is **472 MB** as of 2025-04-13
 - Contains **info** and **image urls** of all cards.



Extracting Card Info

- python `json` module is slow + would need to implement custom handling
- `duckdb` (from prior talk) proved to be super useful & extremely fast
 - Query json file using SQL-like syntax
 - ~1.5s to select info from 450mb json



DuckDB

```
query = f"""
SELECT
    t.id,
    t.oracle_id,
    t.name,
    t.set AS set_code,
    t.set_name,
    img.image_uri,
    '{img_type}' AS img_type,
    '{bulk_type}' AS bulk_type
FROM read_json('{path_bulk}') t
CROSS JOIN LATERAL (
    VALUES
        (t.image_uris.{img_type}),
        (t.card_faces[1].image_uris.{img_type}),
        (t.card_faces[2].image_uris.{img_type})
    ) AS img(image_uri)
WHERE img.image_uri IS NOT NULL
ORDER BY t.id;
"""

ddb = duckdb.connect(self.__path_cards)
ddb.execute(query)
```

Are we even devs if we're not in dark mode?
Also, if in prod use prepared statements not f-strings

Downloading Images

- 100,000 cards at 1 per second will take 27 hours
- Multi-threading is needed
 - Helper library **doorway** makes this easy with http proxy support.

Although Scryfall doesn't have rate limits for images.

```
from doorway.x import ProxyDownloader

# collect a default list of proxies and cache them
downloader = ProxyDownloader()

# download multiple files in parallel while
# rotating http proxies and retrying if any fail
failed = downloader.download_threaded([
    ("https://example.com/file1.txt", "my/file1.txt"),
    ("https://example.com/file2.txt", "my/file2.txt"),
    ("https://example.com/file3.txt", "my/file3.txt"),
])
```

\$ pip install doorway

<https://github.com/nmichlo/doorway>

please check the Ts & Cs of websites before use



MTG Dataset

- There is a fair bit of additional code needed to turn this into a dataset
 - Caching
 - Iteration
 - Image loading
 - Image standardisation
- Can use `mtgdata` which does all of this for you.
- Dataset is an object that can be iterated over to get images. Can wrap for ML libs.

```
from mtgdata import (
    ScryfallDataset,
    ScryfallImageType,
    ScryfallBulkType,
)

dataset = ScryfallDataset(
    img_type=ScryfallImageType.small,
    bulk_type=ScryfallBulkType.default_cards,
    download_mode='ondemand',
)

for card in dataset:
    im: "PIL.Image.Image" = card.dl_and_open_im_resized()
```

`$ pip install mtgdata`

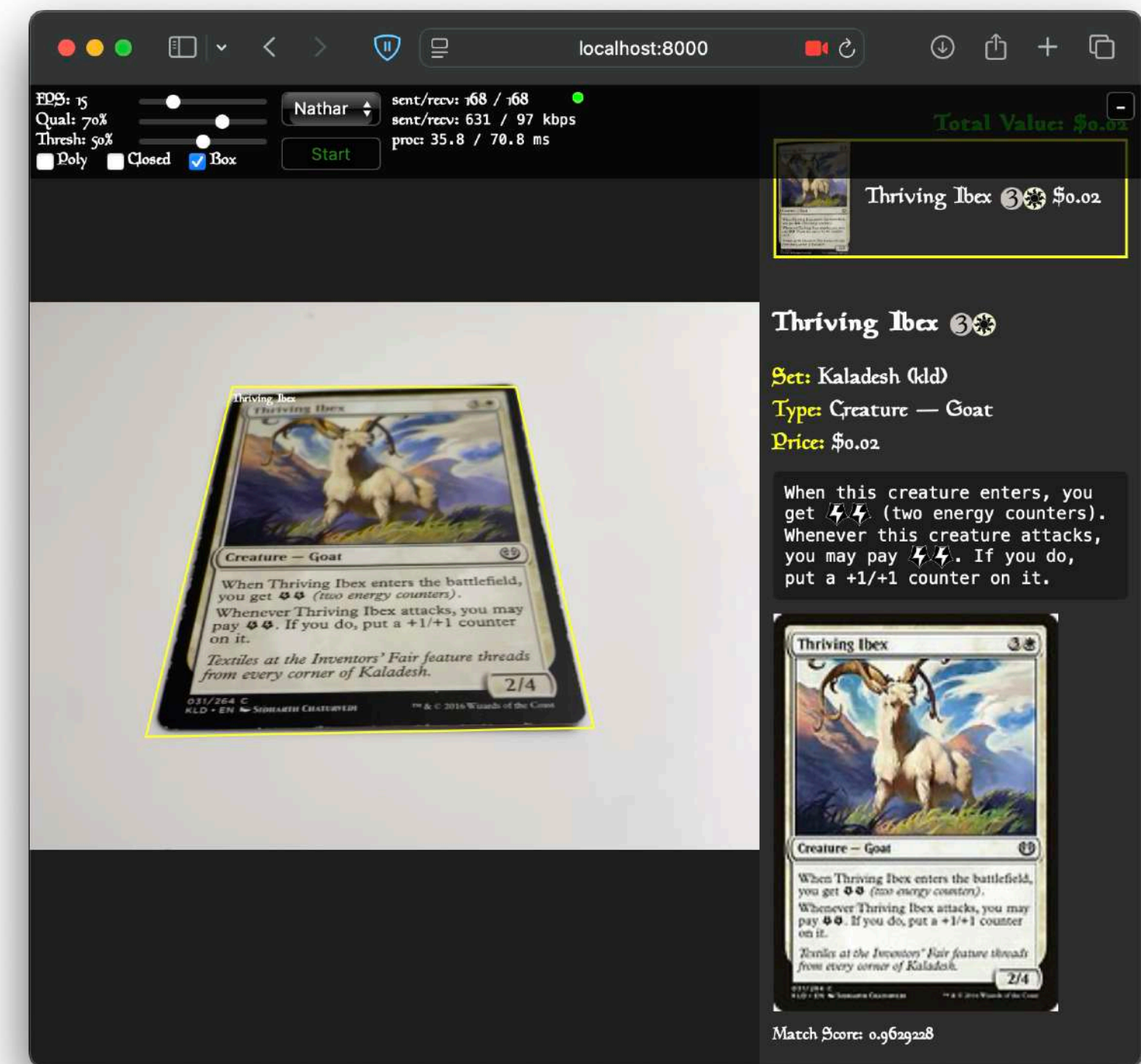
<https://github.com/nmichlo/mtg-dataset>



5. Demo Code

Code for Detector, Encoder, Synthetic Data

- `mtg-vision` contains all the code for this talk, including:
 - **Embedding**
 - Synthetic Data + Model + Export
 - **Instance segmentation**
 - Synthetic Data + Model + Export
 - **Client + Server**
 - Web-socket server with detection, tracking and card matching.
 - Lightweight frontend for demo



<https://github.com/nmichlo/mtg-vision>



> exit

> ./demo

