**Methods for Card Recognition and Game Tracking in Digital Card Games**

**Vision-Based Recognition Approaches**

* **Template Matching & Perceptual Hashing:** Many projects identify cards by matching their images against a database of known card images. For example, a Hearthstone Twitch bot uses **perceptual hashing (pHash)** – it precomputes hashes of all card and hero images, then computes the hash of regions in the screen and finds matches by Hamming distance [github.com](https://github.com/wittenbe/Hearthstone-Image-Recognition#:~:text=Important%20for%20both%20recognition%20tasks,detection%20of%20a%20game%27s%20start). This method was sufficient to detect which cards are in hand or on board, and even identify the player’s hero, enabling automated deck tracking. In ambiguous cases, additional image features can be used – the same Hearthstone bot falls back on SURF feature matching to distinguish look-alikes (e.g. differentiating the coin for first vs second player, or the similar “Victory”/“Defeat” banners)[github.com](https://github.com/wittenbe/Hearthstone-Image-Recognition#:~:text=What%27s%20missing%20is%20the%20detection,both%20coins%20and%20both%20texts). A similar approach has been used for Magic: The Gathering (MTG) – one open-source MTG card detector segments card images and then computes a perceptual hash to identify the card by comparing to a pre-hashed library of card images[tmikonen.github.io](https://tmikonen.github.io/quantitatively/2020-01-01-magic-card-detector/#:~:text=The%20recognition%20of%20the%20card,hashing%20the%20reference%20set%20significantly). Template matching works well when cards are fully visible and a database of images is available, though it can be sensitive to image scaling or partial occlusion.
* **Optical Character Recognition (OCR) on Card Text:** Another strategy is to read the card’s name (or other text) from the screen and look it up. This is effective even if only part of the card is shown, as long as the title text is visible. For instance, **mtgscan** for Magic: The Gathering chooses to apply OCR to the card title in a photo or screenshot and then uses a dictionary of card names for matching[fortierq.github.io](https://fortierq.github.io/mtgscan-ocr-azure-flask-celery-socketio/#:~:text=Mtgscan%20is%20a%20project%20aiming,Optical%20Character%20Recognition)[fortierq.github.io](https://fortierq.github.io/mtgscan-ocr-azure-flask-celery-socketio/#:~:text=Image%20Example%20of%20stacked%20cards,card%20is%20only%20partially%20visible). In that project, Tesseract (an open-source OCR) was tried but performed poorly on MTG card fonts, so the developer switched to a cloud-based OCR (Azure OCR) which gave much better results[fortierq.github.io](https://fortierq.github.io/mtgscan-ocr-azure-flask-celery-socketio/#:~:text=Text%20recognition%20). The recognized text is then fuzzy-matched against the known card list (e.g. using approximate string matching to correct minor OCR errors)[fortierq.github.io](https://fortierq.github.io/mtgscan-ocr-azure-flask-celery-socketio/#:~:text=,in%20sideboard%20is%20totally%20wrong). This OCR approach handles cases where cards overlap or are partially visible (common when scanning a stacked collection or a hand of cards) since you only need a bit of the name to identify the card. Similar OCR-based card recognizers exist for other games – for example, **YuGiOCR** is a Yu-Gi-Oh! card detector that uses Tesseract OCR on the card name and set symbols to identify the card[github.com](https://github.com/Jaster111/YuGiOCR#:~:text=Jaster111%2FYuGiOCR%3A%20Simple%20YuGiOh%20card%20detector,lightning%20scenarios%20and%20positioning%20scenarios). OCR methods rely on clear text and might struggle if the in-game text is stylized or small, but they can be very flexible across large card sets.

Below is a **deep-dive on the two classic, non-DL approaches** that hobby projects still choose first because they’re cheap, light on training data, and easy to reason about. Where possible I cite public repos so you can read the source and copy patterns.

## 1  Template-matching & Perceptual-hash pipelines

### 1-A What the pipeline looks like

| **Stage** | **Typical implementation** | **Notes** |
| --- | --- | --- |
| **1. Locate the card ROI** | Hard-coded crop, edge-finding, or a light object-detector (e.g. Haar, tiny YOLO) | Most arena-draft tools just crop the three pick slots by %-coordinates—no ML needed. |
| **2. Normalise** | Resize to a canonical w × h (e.g. 256 × 330) and convert to 8-bit gray | Removes resolution variance so hashes are comparable. |
| **3. Compute hash** | imagehash.phash(img) (64-bit DCT hash) or an aHash/dHash/wHash variant | A 64-bit perceptual hash captures global structure while ignoring small shifts or colour changes. |
| **4. Lookup** | Hamming-distance search in a pre-built table of {hash → card-id} | Naïve O(N) over 3 000–10 000 cards is ~50 µs; bigger sets use LSH or bucketing. |
| **5. Disambiguate edge-cases** | Keypoint matcher (SIFT/SURF/ORB) or histogram comparison | Helps when two cards share identical art framing (e.g. token vs non-token). |

**Concrete example:** wittenbe/Hearthstone-Image-Recognition precomputes a pHash for every full-art card, then scans arena screenshots and matches ROI hashes; if the Hamming distance of the best hit isn’t < 12, it falls back to SURF descriptor matching to resolve look-alikes. ([GitHub](https://github.com/wittenbe/Hearthstone-Image-Recognition?utm_source=chatgpt.com))

### 1-B Why it works for digital TCGs

* **Stable artwork & framing** – the client always renders the same 2-D art at fixed aspect ratios.
* **Limited cardinality** – even MTG’s 25 000 cards fit in < 500 kB of 64-bit hashes.
* **No training data** – once you have the PNGs (Hearthstone/MTGJSON dumps, etc.), you’re done.

### 1-C Implementation tips

| **Tip** | **Reason** |
| --- | --- |
| Hash **just the art box**, not the full frame. Mana gems and text shift between graphics settings and languages; the art is invariant. |  |
| Keep **multiple hashes per card** at different down-scales (e.g., 128² and 64²) to tolerate mip-mapping blur on stream captures. |  |
| If you need speed for N ≫ 5 000, bucket hashes by the first k bits (LSH) then compare inside the bucket—35× faster look-ups. |  |
| Store the hash table in Redis or as a NumPy array in shared memory; a single 64-bit integer fits in 8 bytes. |  |

### 1-D Strengths & limits

✅ Pros : near-zero CPU/GPU cost, deterministic, explainable.  
❌ Cons : fails on **partial occlusion** (hand overlays), needs re-hash when **new sets** drop, sensitive to colour-grading filters.

## 2  OCR-on-card-text pipelines

### 2-A Canonical workflow

[ROI crop of title area] → [pre-processing] → [OCR engine] → [fuzzy match] → card-id

1. **Crop** – get the title banner (or mana, attack/health numbers).
2. **Pre-process** –
   * grayscale
   * contrast-stretch / CLAHE
   * adaptive threshold (Sauvola)
   * slight dilation/erosion to close gaps in serif fonts.
3. **OCR** –
   * Tesseract with a custom whitelist: tessedit\_char\_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ'
   * PaddleOCR “en\_PP-OCRv3” for better small-font accuracy
   * cloud OCR (Azure Read, Google Vision) if you don’t want to tune local models.
4. **Fuzzy match** – RapidFuzz process.extractOne(ocr\_text, card\_names); keep hit if score > 80.

fortierq/mtgscan follows exactly that: it crops the card title, sends it to Azure OCR (better on MTG’s Beleren font than Tesseract), then fuzzy-matches against MTGJSON, achieving ~98 % title accuracy on smartphone photos. ([GitHub](https://github.com/fortierq/mtgscan?utm_source=chatgpt.com), [Quentin Fortier](https://fortierq.github.io/mtgscan-ocr-azure-flask-celery-socketio/?utm_source=chatgpt.com))  
alesapin/HearhRecognizer does the same with Tesseract but on Hearthstone. ([GitHub](https://github.com/alesapin/HearhRecognizer?utm_source=chatgpt.com))

### 2-B When OCR beats hashing

| **Scenario** | **Why OCR wins** |
| --- | --- |
| **Hand or stack overlay** hides half the art | You still see the title bar. |
| **New card art** leaks before PNGs are published | Text is present, but hash DB is outdated. |
| **Language variants** (e.g., EN / DE) | OCR picks up localized titles and you map via a multilingual DB. |

### 2-C Hard edges and mitigations

| **Challenge** | **Fix** |
| --- | --- |
| Stylised fonts (Hearthstone’s Milstone, MTG’s Beleren) | Train a Tesseract custom font or move to PaddleOCR / cloud OCR. |
| Low-DPI stream captures (720p Twitch) | Super-resolve the crop (Waifu2× or ESRGAN) before OCR—costs 10 ms on GPU. |
| Similar names (“Fireball” vs “Firebolt”) | Use **Levenshtein distance + mana cost + card set** as joint features in matching. |
| Text glow / drop-shadow | Pre-process with morphological opening; threshold on the green/blue glow channels separately. |

### 2-D Performance snapshot

* PaddleOCR-PP-OCRv3: **≈ 8 ms** on RTX 3060 for 256×64 crop
* Azure Read API v4: **200-300 ms** round-trip
* RapidFuzz fuzzy match over 20 000 names: **< 1 ms**

Batch a whole arena screenshot (three titles) and you’re well under 0.5 s even on CPU.

## Choosing between the two

| **Factor** | **Hash / template** | **OCR** |
| --- | --- | --- |
| **Setup time** | minutes (download art → hash) | hours (crop tuning, OCR whitelist) |
| **Handles occlusion** | poor | good (title needs ~20 px) |
| **Adds new set** | re-hash PNG dump | no change |
| **Non-Latin locales** | needs per-locale hash DB | needs additional OCR language model |
| **CPU footprint** | ~0.5 ms per card | 5-300 ms depending on OCR engine |

**Most resilient hobby tools mix both:** hash when the full art is visible (board, collection) and OCR when only the title is visible (hand, stack, discovery UI). A two-stage cascade keeps latency low while covering edge-cases.

### Quick reference implementations

| **Repo / Link** | **Game** | **Technique** | **Notes** |
| --- | --- | --- | --- |
| Hearthstone-Image-Recognition ([GitHub](https://github.com/wittenbe/Hearthstone-Image-Recognition?utm_source=chatgpt.com)) | Hearthstone | pHash + SURF fallback | Twitch chat bot, picks & end-game detection |
| hs\_card\_recognizer ([GitHub](https://github.com/zanardob/hearthstone-card-recognizer?utm_source=chatgpt.com)) | Hearthstone | pHash (art only) | Board-state extractor, Python + OpenCV |
| mtgscan ([GitHub](https://github.com/fortierq/mtgscan?utm_source=chatgpt.com), [Quentin Fortier](https://fortierq.github.io/mtgscan-ocr-azure-flask-celery-socketio/?utm_source=chatgpt.com)) | MTG | Azure OCR + fuzzy | Works on photos & Arena screenshots |
| HearhRecognizer ([GitHub](https://github.com/alesapin/HearhRecognizer?utm_source=chatgpt.com)) | Hearthstone | Tesseract OCR | C++/Qt demo, shows ROI crop code |
| magicscan ([GitHub](https://github.com/dctucker/magicscan?utm_source=chatgpt.com)) | MTG | OpenCV contour + Tesseract | Early prototype but clear preprocessing scripts |

Each repo is small (<2 k LoC) and illustrates real-world engineering trade-offs you can borrow directly for Hearthstone or any other digital TCG.