# KYC - Know Your Cards
## A game card grader and classifier

Bianchi Christian (1984056), Mastrorilli Nicola (2015399),
Ramil Leonard Vincent (1984766), Sannino Siria (2001580)

La Sapienza University of Rome, Department of Informatics
https://github.com/u-siri-ous/KYC

**Abstract.** The program is designed to analyze pictures of Pokémon TCG 1st gen cards. It aims to classify the Pokémon, providing information regarding name, type, abilities, attacks, and any special features or edition indicators, and grade the card's condition based on the Beckett Grading Services, a.k.a. BGS.
The code is released publicly under GPL-3.0 License on GitHub.

**Keywords:** Pokémon · Neural Networks · OpenCV

## 1 Introduction

The project is structured in three parts, namely:

- Grader
- Classifier
- GUI

which are then implemented in a *main* file to coexist.
A strong variety of modules and techniques were used to achieve this objective, such as Keras for an easy neural network implementation, OpenCV for the wide range of functions it offers and Tkinter for a clean GUI.

## 2 Grader

The grader was implemented using the BGS grading scale and OpenCV. The usage of an automated tool was deemed necessary, as it is often hard to evaluate a card based solely on the naked eye of a newcomer. The condition value is crucial, as it indicates the card's state and influences its market price.

### 2.1 About BGS

Beckett Grading Services ensures a fair and comprehensive card evaluation based on four factors:

1. Centering
2. Corners
3. Edges
4. Surface

and a score from 1 to 10 in each of them accordingly, with half-points for in-between characteristics. The card will then be assigned a category according to the score in each factor.

| **Category** | **Grade** | **Front Centering** | **Back Centering** |
|---|---|---|---|
| Pristine | 10 | $\geq$ 50/50 both ways | $\geq$ 60/40 |
| Gem Mint | 9.5 | $\geq$ 50/50 one way, $\geq$ 55/45 other | $\geq$ 60/40 |
| Mint | 9 to 8 | $\geq$ 55/45 both ways | $\geq$ 70/30 |
| Near Mint | 8 to 7 | both ways between 60/40 and 65/35 | between 80/20 and 90/10 |
| Excellent Mint | 6 | $\geq$ 70/30 both ways | $\geq$ 95/5 |
| Excellent | 5 to 4 | $\geq$ 75/25 both ways | $\geq$ 95/5 |
| Very Good | 4 to 3 | both ways between 80/20 and 85/15 | $\geq$ 100/0 |
| Good | 2 | $\geq$ 90/10 both ways | 100/0 or offcut |
| Poor | 1 | 100/0 or offcut | 100/0 or offcut |

| **Corners** | **Edges** | **Surface** |
|---|---|---|
| Perfect | Perfect | Flawless color and gloss |
| Mint | Mint | Extremely minor flaws, scratchless |
| Mint | Mint | Minor spots and scratches |
| Sharp with minor imperfections | Relatively smooth | Minor speckling with occasional print spots |
| Fuzzy but not frayed | Moderate chipping | Minor discoloration and noticeable spots |
| Fuzzy with minor ding | Rough with no layering | Minor imperfections and discoloration |
| Notched with minor layering | Chipped with layering | Heavy print spots and moderate imperfections |
| Moderately layered | Severely chipped | Severe print spots and imperfections |
| Noticeably layered | Destructively chipped | Severe imperfections and creases |

### 2.2 Implementing a Grader using OpenCV

The grader uses image transformation techniques for each of the categories mentioned above.
In particular, the **surface** evaluation was obtained by averaging the three other values, as it strongly depends on the card's condition in general. Specifically, surface condition depends on flaws that are most likely not visible in a picture. The classifier is of use in cases where some info cannot be retained because of evident damages, such as an entire part of the card cut away, in which the grade would be low, but all info can still be displayed either way. For the scope of this project, the focus was on the front centering rather than the back centering.

The HSV colorspace was used to identify yellow through saturation and color values using thresholds, after which, the result was **binarized** to get a mask. This also helped in locating the card in the picture and cutting it out.

```
lower_yellow = np.array([20, 100, 100], dtype=np.uint8)
upper_yellow = np.array([30, 255, 255], dtype=np.uint8)

# Create a binary mask where pixels falling within the yellow
                                     color range become white (255
                                    ) and others black (0)
mask = cv.inRange(hsv_image, lower_yellow, upper_yellow)
```

Centering, Corners and Edges were evaluated by masking the card picture and analyzing the percentage of white pixels.

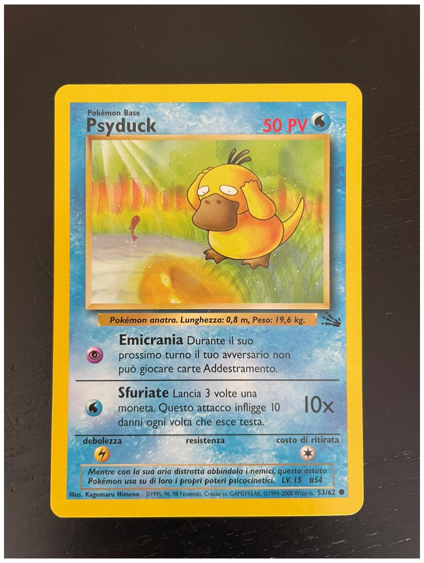


**Fig. 1.** Original image

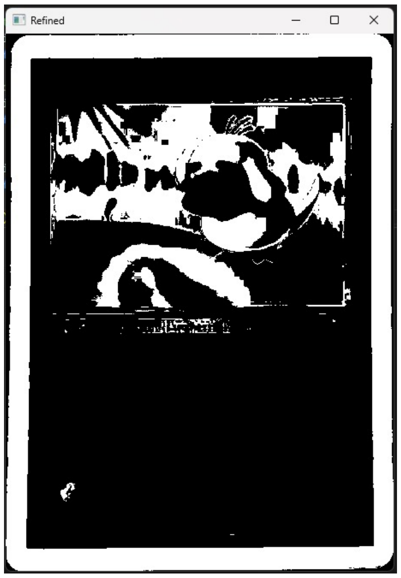
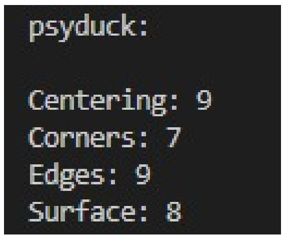


**Fig. 2.** Cropped and masked image


```
psyduck:

Centering: 9
Corners: 7
Edges: 9
Surface: 8
```

**Fig. 3.** Marks for Psyduck

## 3 Classifier

The classifier was implemented using Keras, with the aid of Jupyter Notebook. This dataset was taken from Kaggle and adapted to the scope of the project, as some 1st Pokémons were missing.
In particular, there are around 25 to 50 images for each Pokemon, all with the Pokemon in the centre. Most (if not all) images have relatively high quality (correct labels, centred).

### 3.1 Pre-Processing Data: Splitting and Data Augmentation

The dataset was split using two techniques, namely:

- Sampling 15 images from the training set and using them in the validation set, as seen in this article.
- The 80/20 technique

Data augmentation was implemented to increase the used dataset's size by applying various image transformations, such as rescaling, shear range, horizontal flip and zooming.

### 3.2 CNN Models

Two CNNs were created to correctly classify the Pokémon, specifically:

1. The first CNN has two Convolutional layers:
   - 64 filters, (5, 5) stride
   - 128 filters, (3, 3) stride

   followed by two (2, 2) Max Pooling layers to sharpen the image for the second one and a Flatten layer followed by 151 SoftMax layers to output class. This model uses a 64x64x3 input shape, a ReLU (hidden layers) and SoftMax (output layer) as activation functions, an Adam optimizer and the Categorical Cross Entropy loss function.

2. The second CNN is similar, it has three Convolutional layers with the third layer being a duplicate of the second; and an input size of 128x128x3.

Early stopping was implemented in the training stage to prevent data fitting problems. If the current epoch's loss function is greater than the previous epoch's, training is stopped and the model is saved without generating the .h5 file.

### 3.3 Training and hyperparameters

Hyperparameters values for the CNNs were:

1. First
   - 30 epochs
   - Batch size 32
   - Learning rate .001
2. Second
   - 20 epochs
   - Batch size 32
   - Learning rate .001

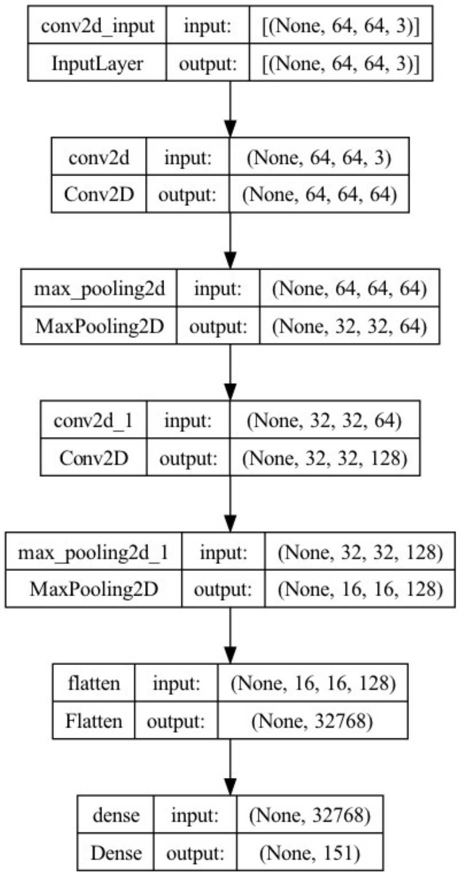

conv2d_input | InputLayer | input: [(None, 64, 64, 3)] | output: [(None, 64, 64, 3)]
conv2d | Conv2D | input: (None, 64, 64, 3) | output: (None, 64, 64, 64)
max_pooling2d | MaxPooling2D | input: (None, 64, 64, 64) | output: (None, 32, 32, 64)
conv2d_1 | Conv2D | input: (None, 32, 32, 64) | output: (None, 32, 32, 128)
max_pooling2d_1 | MaxPooling2D | input: (None, 32, 32, 128) | output: (None, 16, 16, 128)
flatten | Flatten | input: (None, 16, 16, 128) | output: (None, 32768)
dense | Dense | input: (None, 32768) | output: (None, 151)

**Fig. 4.** CNN 1

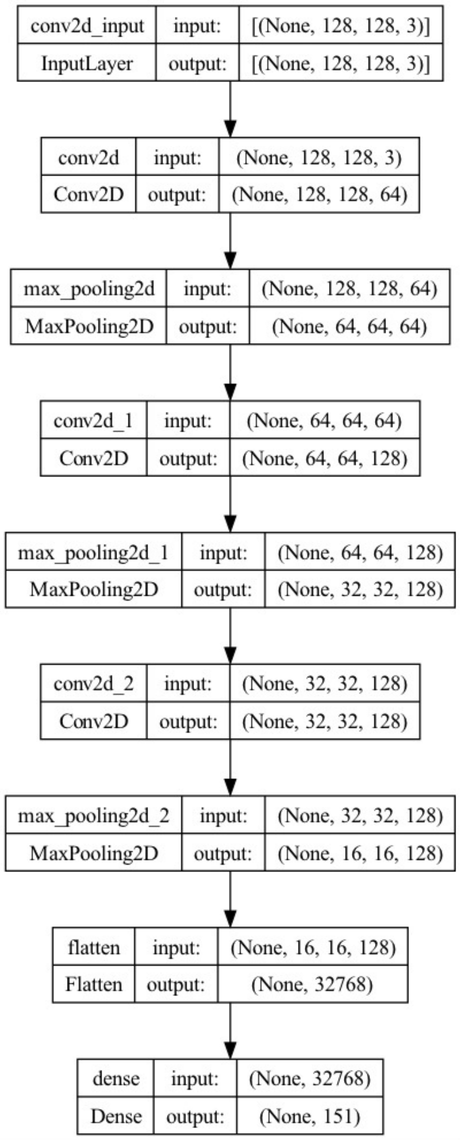

conv2d_input | InputLayer | input: [(None, 128, 128, 3)] | output: [(None, 128, 128, 3)]
conv2d | Conv2D | input: (None, 128, 128, 3) | output: (None, 128, 128, 64)
max_pooling2d | MaxPooling2D | input: (None, 128, 128, 64) | output: (None, 64, 64, 64)
conv2d_1 | Conv2D | input: (None, 64, 64, 64) | output: (None, 64, 64, 128)
max_pooling2d_1 | MaxPooling2D | input: (None, 64, 64, 128) | output: (None, 32, 32, 128)
conv2d_2 | Conv2D | input: (None, 32, 32, 128) | output: (None, 32, 32, 128)
max_pooling2d_2 | MaxPooling2D | input: (None, 32, 32, 128) | output: (None, 16, 16, 128)
flatten | Flatten | input: (None, 16, 16, 128) | output: (None, 32768)
dense | Dense | input: (None, 32768) | output: (None, 151)

**Fig. 5.** CNN 2

## 3.4 Results

For the first model, a maximum of 96.9% accuracy during training with loss function of 0.11 and 96.7% accuracy on validation with loss function of 0.12 were achieved.
For the second model, a maximum of 94% accuracy during training with loss function of 0.21 and 94% accuracy on validation with loss function 0.22 were achieved.
The values for hyperparameters were found useful, regardless of the loss of detailing due to the small scaling of the input image.

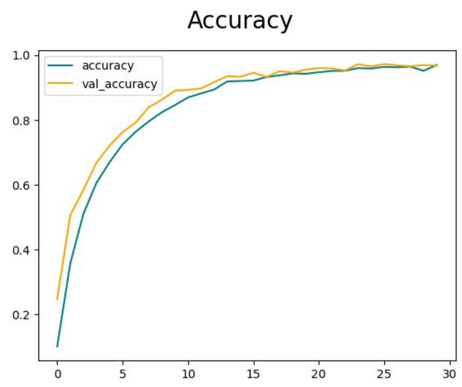


**Fig. 6.** Accuracy, model 1

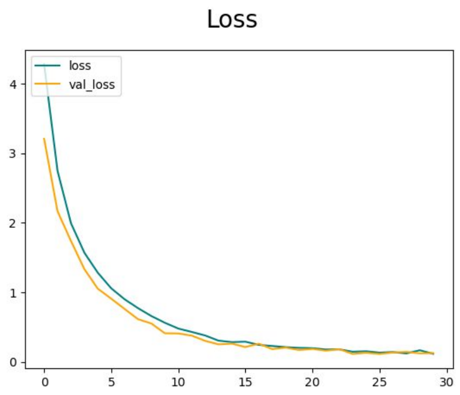


**Fig. 7.** Loss Function, model 1

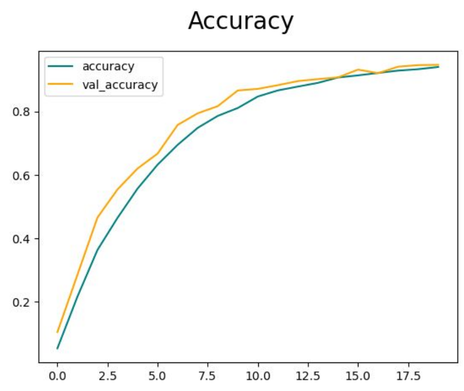


**Fig. 8.** Accuracy, model 2

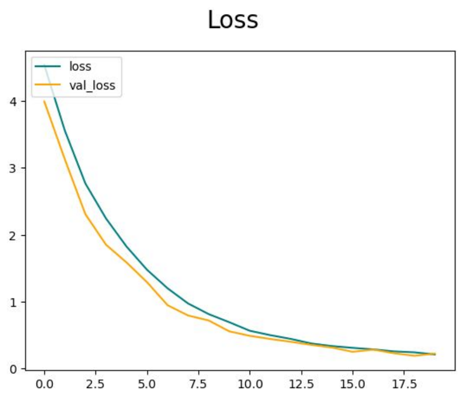
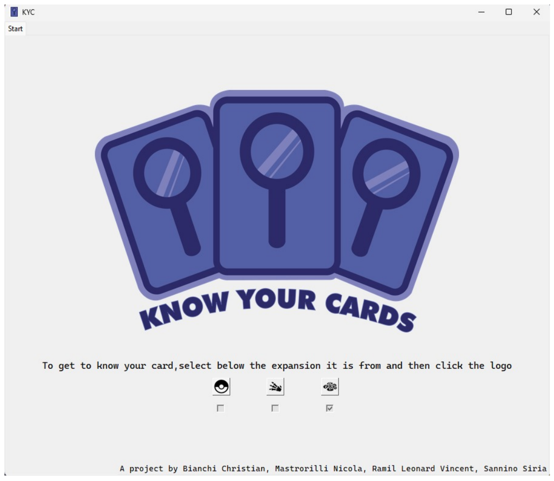


**Fig. 9.** Loss Function, model 2

## 4 GUI

The GUI was implemented using Tkinter, a library in the standard Python toolkit, and Pillow. This was made in order to make the experience more user-friendly. When starting, the user is prompted to select the expansion between Base, Fossil and Jungle, afterwards, the snapped picture of the card is displayed along with marks in pure BGS style and all the card's info. The GUI takes the Pokémon's type as color palette, as seen below.

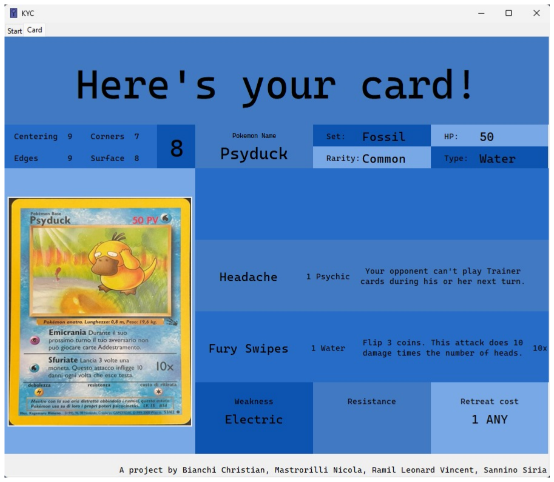


To get to know your card,select below the expansion it is from and then click the logo

☐ ☐ ☑

A project by Bianchi Christian, Mastrorilli Nicola, Ramil Leonard Vincent, Sannino Siria

**Fig. 10.** Starting GUI



Here's your card!

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Centering 9 | Corners 7 | 8 | Pokemon Name | Set: | Fossil | HP: | 50 |
| Edges 9 | Surface 8 | | Psyduck | Rarity: | Common | Type: | Water |

| | | | |
|---|---|---|---|
| Headache | 1 Psychic | Your opponent can't play Trainer cards during his or her next turn. | |
| Fury Swipes | 1 Water | Flip 3 coins. This attack does 10 damage times the number of heads. | 10x |

| Weakness | Resistance | Retreat cost |
|---|---|---|
| Electric | | 1 ANY |

A project by Bianchi Christian, Mastrorilli Nicola, Ramil Leonard Vincent, Sannino Siria

**Fig. 11.** Displaying results with GUI

## 5 Future developments

In the future, the aims of the project are:

1. To be extended to other popular trading card games, such as Magic The Gathering and Yu-Gi-Oh, by adapting the structure.
2. To be implemented in a mobile version for fast and practical use on the go.

## References

1. Zhang, Lance. Dataset from Kaggle: 7000 hand-cropped and labeled Pokemon images for classification, November 2019
2. Rawat, Akash. Article on Pokémon Classification: Pokémon Classification Using CNN, June 2021
3. Beckett Grading Services. The official grading scale