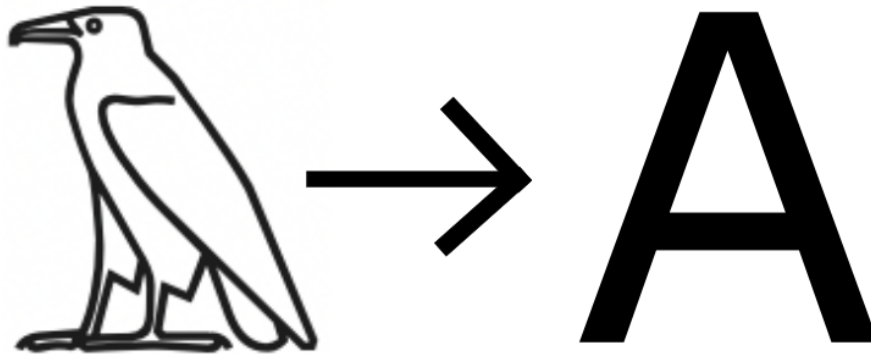


# Hieroglyph Reading System Using Scikit-Image

Politechnika Krakowska  
Natural Language Processing

May 2025



Rolfis Ramses Solano Méndez  
Paula Mosquera Del Río

# Contents

<b>1</b>	<b>Abstract</b>	<b>3</b>
<b>2</b>	<b>Introduction</b>	<b>4</b>
2.1	Aim . . . . .	4
2.2	Scope . . . . .	4
2.3	Methodology . . . . .	4
<b>I.</b>	<b>Theoretical Part</b>	<b>5</b>
<b>3</b>	<b>Definitions and Steps</b>	<b>5</b>
3.1	Image Processing Basics . . . . .	5
3.1.1	Resizing . . . . .	5
3.1.2	Grayscale Conversion . . . . .	5
3.1.3	Histogram Equalization . . . . .	5
3.1.4	Noise Removal . . . . .	6
3.2	Feature Extraction With HOG . . . . .	6
3.3	Classification with Logistic Regression . . . . .	7
3.4	Hieroglyph Recognition and Transliteration . . . . .	7
3.5	Fuzzy String Matching . . . . .	7
<b>II.</b>	<b>Practical Part</b>	<b>8</b>
<b>4</b>	<b>Practical Implementation</b>	<b>8</b>
4.1	Data Source and Implementation . . . . .	8
4.2	Feature Extraction . . . . .	8
4.3	Baseline Model and Class Imbalance . . . . .	9
4.4	Balancing with KMeansSMOTE . . . . .	10
4.5	Evaluation after Over-sampling . . . . .	10
4.6	Sequential Glyph Processing . . . . .	11
4.7	Transliteration and Name Matching . . . . .	12
4.8	Results . . . . .	14
<b>5</b>	<b>Summary</b>	<b>14</b>

# 1 Abstract

Egyptian hieroglyphs are one of the oldest writing systems in human history, consisting of thousands of hieroglyphic symbols used for over 3,000 years.<sup>[1]</sup> Unlike modern alphabets, hieroglyphs are symbolic, which makes them an interesting topic for computational interpretation, especially in the field of Natural Language Processing.

The purpose of this work is to build a system that recognizes Egyptian glyphs in images and convert them into their equivalent English words.

The approach is carried out using classical image processing techniques for feature extraction. These extracted features are classified by a trained model and afterwards, the transliterated terms are matched to the English equivalents.

Our system was able to successfully recognize, transliterate and translate a small set of assembled hieroglyphic signs.

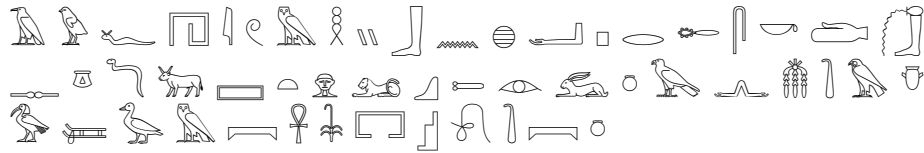
## 2 Introduction

### 2.1 Aim

The aim of this project is to train a model capable of reading a sequence of individual glyphs placed in an image. In addition, these identified pictograms are transliterated and translated.

### 2.2 Scope

This project focuses on the recognition of individual Egyptian pictograms on a sufficient contrasting background. The model is trained on the following hieroglyphs:



The system is designed specifically for the transliteration and translation of Egyptian divinity names into their respective English names. The recognition of sentence translation is beyond the scope of our implementation.

### 2.3 Methodology

The implementation was achieved through image processing techniques, including grayscale conversion, Gaussian blurring, and Histogram of Oriented Gradients (HOG). The resulting features are classified using a Logistic Regression model.

In the next step, the detected symbols are transliterated based on the model's output. Finally, fuzzy string matching is used to map the transliterated terms to their English equivalents.

# I. Theoretical Part

## 3 Definitions and Steps

### 3.1 Image Processing Basics

In order to increase the quality and discriminative power of the Histogram of Oriented Features (HOG), we took several steps to preprocess the samples in the dataset.

#### 3.1.1 Resizing

HOG expects a fixed aspect ratio, which means that the relationship between width and height is proportional, as the feature vector size depend on the image dimensions.<sup>[2]</sup>

#### 3.1.2 Grayscale Conversion

Since a color image has 3 channels (red, green and blue), computing HOG in all three would be computationally expensive. This high dimensionality is reduced with the grayscale to one channel, preserving the edge information and allowing HOG to just focus on geometric structure.<sup>[3]</sup>

#### 3.1.3 Histogram Equalization

The equalization of the histogram, or frequency of intensity levels, leads to more uniform adjusted pixel intensities. In other words, the image contrast is more uniform, has a higher contrast and is easier to process by feature descriptors.<sup>[4]</sup>

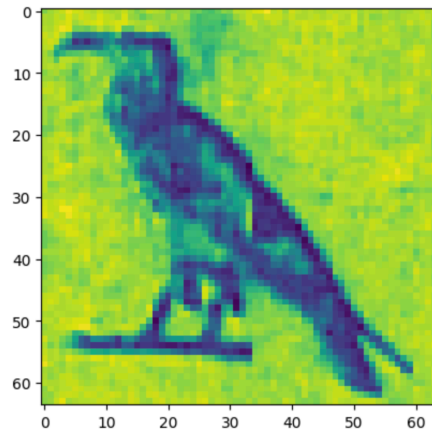


Figure 1: Not equalized picture.

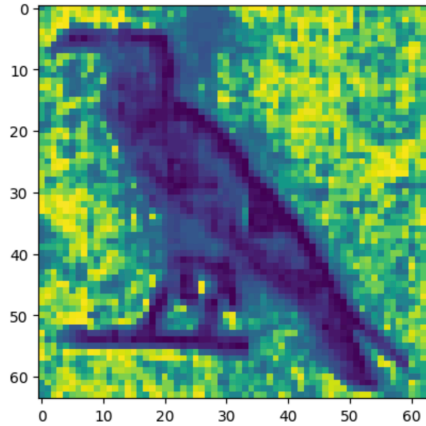


Figure 2: Equalized picture.

### 3.1.4 Noise Removal

In our case we employed Gaussian blur for noise removal. This classic technique reduces the unwanted noise in a frame by averaging neighboring pixels using a Gaussian function. At the same time, it preserves the edges and makes contours cleaner.<sup>[5]</sup>

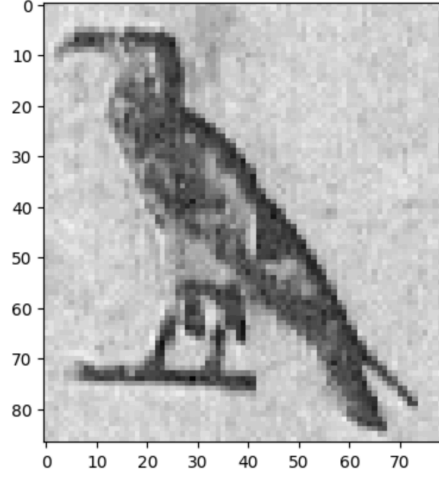


Figure 3: Not blurred picture.

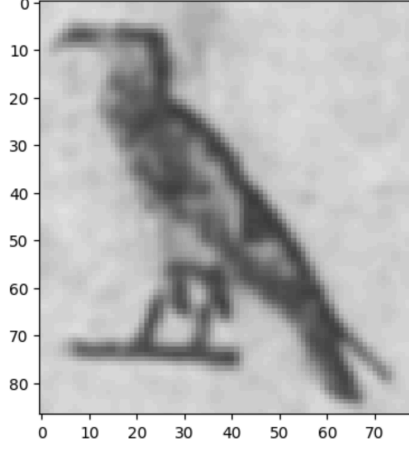


Figure 4: Blurred picture.

## 3.2 Feature Extraction With HOG

The Histogram of Oriented Gradients is a feature descriptor used for image processing to detect objects or patterns, characterized by capturing the distribution of gradient orientations in localized image regions.<sup>[6]</sup>

1755	1756	1757	1758	1759	1760	1761	1762	1763	target
0.368727	0.084597	0.051519	0.058797	0.234477	0.024358	0.056171	0.323352	0.162896	G1
0.167880	0.016888	0.063458	0.028240	0.086927	0.268027	0.268027	0.268027	0.190149	G1
0.341430	0.241406	0.179246	0.098209	0.210073	0.000000	0.000000	0.149929	0.182183	G1
0.146512	0.174489	0.269308	0.195522	0.269308	0.269308	0.146588	0.189456	0.050232	G1
0.205384	0.096347	0.031477	0.056259	0.103416	0.177207	0.191659	0.098022	0.313693	G1
...	...	...	...	...	...	...	...	...	...
0.219285	0.242186	0.055931	0.027106	0.127643	0.016266	0.086485	0.242186	0.150634	W24
0.238142	0.285875	0.089166	0.024682	0.027138	0.005463	0.000000	0.056310	0.151475	W24
0.197660	0.216700	0.053246	0.204677	0.247444	0.147598	0.202498	0.213175	0.049592	W24
0.194104	0.048073	0.014485	0.013363	0.050692	0.045374	0.087874	0.120575	0.222267	W24
0.352921	0.079366	0.050034	0.026607	0.217567	0.064905	0.036489	0.012515	0.179903	W24

Figure 5: Visualization of a data frame with HOG features.

This feature descriptor returns a structured vector of numerical values which can be used by a model to classify a given image, in our case, a hieroglyph.

### 3.3 Classification with Logistic Regression

Our dataset contains over 50 distinct hieroglyph classes in our dataset, which are used to train a multinomial logistic regression model. In contrast to binary logistic regression, which applies the sigmoid function, the multi-class logistic regression uses the softmax function to assign a probability to each possible class label. This softmax function distributes the probabilities between the multiple classes, so that their sum equals one.

The model measures the difference between the predicted probability and the actual class, called cross-entropy loss. The aim of the model is to reduce this loss. Another important characteristic of this classification algorithm is the possibility of giving weights in order to minimize the cross-entropy loss.<sup>[7]</sup>

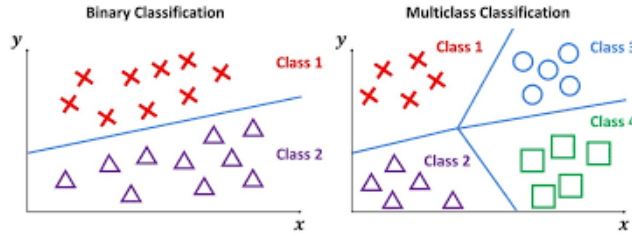


Figure 6: Multinomial Logistic Regression.<sup>1</sup>

### 3.4 Hieroglyph Recognition and Transliteration

Given that the input is an image containing individual hieroglyphs, we applied Canny Edge Detection and contour detection in order to identify the outlines of the connected pictograms. Afterward, the detected contours are extracted and each extracted region undergoes the same image processing pipeline as the individual pictographic signs, and is subsequently classified by the trained model. These encoded character representations are concatenated to form a complete output.

### 3.5 Fuzzy String Matching

Due to the discrepancy between Egyptian deity names in English and their transliteration from pictographic signs, we implemented the fuzzy string matching to find close matches for the transliterated deity name. If the character representation is close enough to the modern English name, the matched output is mapped to the corresponding standard English deity name.

---

<sup>1</sup>[Image Source](#)

## II. Practical Part

### 4 Practical Implementation

#### 4.1 Data Source and Implementation

The labeled images and the dictionary with the transliterations originate from the project *Rozpoznawanie hieroglify*, available at <https://ii.pk.edu.pl/~rkycia/classes/2024/lato/NLPNiestacjonarne.html>, in which this project is based on.

The only modification made was unification of the images labeled with the key *G35* to the key *G1*. Upon review, we concluded that both sets of images were in reality the same heroglyphic symbol.

The structure of the train directory, containing the images with the labels, was equivalent to the provided keys, what simplified the data preprocessing.

```
letters = {
    "G1" : "a",
    "M17" : "i",
    "Z4" : "j",
    "D36" : "a",
    "G43" : "u",
    "Z7" : "u",
    "D58" : "b",
    "Q3" : "p",
    "I9" : "f",
    ...}
train/
Aa1/
D2/
D21/
D35/
D36/
D4/
D46/
D58/
D60/
```

#### 4.2 Feature Extraction

The corresponding label was assigned to each frame, based on the directory in which they were stored. This was followed by the creation of a *pandas* data frame, containing the path to the image and the respective target.

```
raw_df = pd.DataFrame({'image': image_paths, 'label': labels})
```

This function was created with the utilization of *OpenCV* and *Scikit-Image* libraries to apply the preprocessing steps discussed in the theoretical part and extract the HOG features. The parameters chosen for the HOG are necessary to obtain cleaner, more robust and discriminative features for each glyph. This method was applied to each image in the first data frame, resulting in a new data frame, which has the corresponding HOG features and label.

```
def extract_hog_features(image_path):
    try:
        image = cv2.imread(image_path)
        if image is None:
```



```

        raise FileNotFoundError(f"Image not found at {image_path}")
    except:
        print(f"Image not found at {image_path}")
        return None
    image = cv2.resize(image, (64, 64),
                       interpolation=cv2.INTER_AREA)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image = cv2.equalizeHist(image)
    image = cv2.GaussianBlur(image, (5, 5), 0)
    return hog(image, orientations=9,
               pixels_per_cell=(8, 8),
               cells_per_block=(2, 2),
               block_norm='L2-Hys',
               transform_sqrt=True)

```

### 4.3 Baseline Model and Class Imbalance

After examining the data distribution, we found that the number of frames corresponding to each hieroglyphic symbol was not well distributed, being the maximum count 448 for the label *N35* and the minimum 6 for the target *D60*. This could lead the logistic regression to favor the majority class to minimize the overall error. Nevertheless, we decided to take a look at the performance of our model with this kind of data and defining the class weights as *balanced*, obtaining an accuracy of 98%.

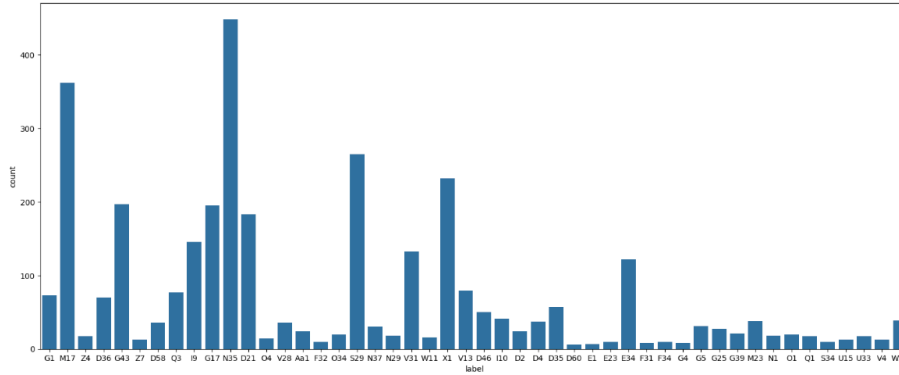


Figure 7: Distribution of class labels.

Even though this is a priori a good result, we can observe that some classes underperformed:

label	precision	recall	f1-score	support
...				
V31	0.97	1.00	0.98	30
V4	1.00	1.00	1.00	4

W11	0.50	1.00	0.67	1
W24	0.83	0.71	0.77	7
X1	1.00	1.00	1.00	50
Z4	1.00	1.00	1.00	3
Z7	1.00	1.00	1.00	3

#### 4.4 Balancing with KMeansSMOTE

With the aim of having a better distribution of the data, we decided to use over sampling with Synthetic Minority Over-sampling Technique (SMOTE), which creates synthetic examples applying *KMeansSMOTE* from the *imblearn* library. This type of over-sampling technique adds clustering with K-Means before implementing the usual SMOTE technique.<sup>[8]</sup> This creates more meaningful and better distributed synthetic samples. Once applied, we observe that the count of all class labels skews between 448 and 450.

```
smote = KMeansSMOTE(
    sampling_strategy='not majority',
    cluster_balance_threshold=0.001,
    random_state=42)
X_balanced, y_balanced = smote.fit_resample(x, y)
```

#### 4.5 Evaluation after Over-sampling

This new data, composed by our original and synthetic data, was split and fit into the logistic regression model, which shows the following results:

label	precision	recall	f1-score	support
Aa1	1.00	0.99	0.99	78
D2	1.00	1.00	1.00	103
D21	1.00	0.97	0.98	87
D35	1.00	1.00	1.00	93
D36	1.00	1.00	1.00	95
D4	1.00	1.00	1.00	91
D46	1.00	1.00	1.00	98
D58	1.00	0.99	1.00	105
D60	1.00	1.00	1.00	85
E1	1.00	1.00	1.00	87
E23	1.00	1.00	1.00	93
E34	1.00	1.00	1.00	81
F31	1.00	1.00	1.00	74
F32	1.00	1.00	1.00	90
F34	1.00	1.00	1.00	102
G1	0.99	0.99	0.99	82
G17	1.00	1.00	1.00	75
G25	1.00	1.00	1.00	93

G39	1.00	0.99	0.99	94
G4	1.00	1.00	1.00	88
G43	1.00	1.00	1.00	100
G5	0.99	1.00	0.99	87
I10	1.00	1.00	1.00	90
I9	1.00	1.00	1.00	97
M17	0.99	0.99	0.99	84
M23	0.99	1.00	0.99	85
N1	1.00	0.99	0.99	87
N29	1.00	1.00	1.00	101
N35	0.98	1.00	0.99	86
N37	1.00	1.00	1.00	98
O1	1.00	1.00	1.00	95
O34	1.00	1.00	1.00	81
O4	1.00	1.00	1.00	77
Q1	1.00	1.00	1.00	81
Q3	1.00	1.00	1.00	85
S29	1.00	1.00	1.00	87
S34	1.00	1.00	1.00	86
U15	1.00	1.00	1.00	88
U33	1.00	1.00	1.00	95
V13	1.00	1.00	1.00	105
V28	1.00	1.00	1.00	82
V31	1.00	1.00	1.00	92
V4	1.00	1.00	1.00	87
W11	1.00	1.00	1.00	79
W24	0.98	1.00	0.99	87
X1	0.99	1.00	0.99	89
Z4	1.00	1.00	1.00	108
Z7	1.00	1.00	1.00	95

Moreover, with the implementation of a 5-fold cross-validation using this well distributed data we achieved a score of 99% as mean accuracy.

## 4.6 Sequential Glyph Processing

It is necessary to apply the same steps of image processing to the new glyphs sequence which is being processed.

```
def predict_heroglyphs_series(image_path, model):
    img = cv2.imread(image_path)
    if img is None:
        raise FileNotFoundError(f"Image not found at {image_path}")

    im_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    im_gray = cv2.equalizeHist(im_gray)
```

```

im_gray_blur = cv2.GaussianBlur(im_gray, (5, 5), 0)
...

```

Further steps are to be done in order to retrieve the individual hieroglyphs from the sequence. We achieved this finding the contours, which helps locate the boundaries of each glyph in the image, filtering small noise in form of small contours and sorting the founded areas in aleft-to-right order.

```

...
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
edges = cv2.morphologyEx(edges, cv2.MORPH_CLOSE, kernel)

ctrs, hier = cv2.findContours(edges, cv2.RETR_EXTERNAL,
                              cv2.CHAIN_APPROX_SIMPLE)

filtered_ctrs = []
for c in ctrs:
    if cv2.contourArea(c) > 100:
        epsilon = 0.01 * cv2.arcLength(c, True)
        approx = cv2.approxPolyDP(c, epsilon, True)
        filtered_ctrs.append(approx)

bboxes = [cv2.boundingRect(c) for c in filtered_ctrs]
sorted_bboxes = sorted(bboxes, key=lambda b: b[0])
...

```

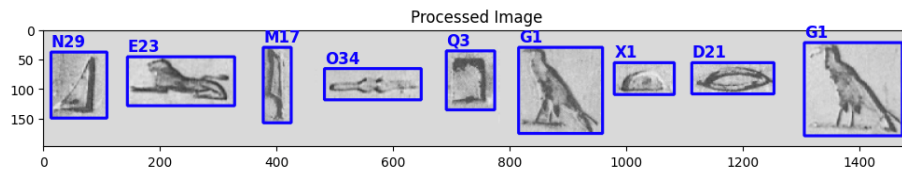


Figure 8: Recognizing of the individual glyphs for the word 'Tutankhamun'.

## 4.7 Transliteration and Name Matching

The identified glyphs are extracted and normalized to the same format expected by our trained logistic regression model.

```

...
hieroglyphic = []
text = []
positions = []
for _, i_bboxes in enumerate(sorted_bboxes):

```

```

x, y, w, h = i_bboxes
aspect_ratio = w / float(h)
if h > 16 and w > 16 and 0.2 < aspect_ratio < 5:
    cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 3)
    roi = im_gray_blur[y:y+h, x:x+w]
    roi = cv2.resize(roi, (64, 64),
                     interpolation=cv2.INTER_AREA)
    nbr = predict_single_hieroglyph(roi, model)

    positions.append((x, y, w, h, str(nbr)))
    hieroglyphic.append(letters.glyphs[str(nbr)])
    text.append(letters.letters[str(nbr)])
...

```

The following method predicts the class of the given hieroglyph image.

```

def predict_single_hieroglyph(img, model):
    img_resized = img

    features = hog(img_resized,
                   orientations=9,
                   pixels_per_cell=(8, 8),
                   cells_per_block=(2, 2),
                   block_norm='L2-Hys',
                   transform_sqrt=True)

    features = features.reshape(1, -1)

    prediction = model.predict(features)
    return prediction[0]

```

Word:  
**tutanchimn**  
 Hieroglyphic:



Figure 9: Transliteration of the term 'tutanchmin'.

Taking the predicted word which can be compared with the listed divinity names. Taking into account that the transliteration does not always match 1 : 1 the modern English name, the fuzzy string matching of the *difflib* library can be employed to achieve this lightweight translation. Through the use of *SequenceMatcher* from the same library, the *get\_close\_matches* method calculates a ratio which compares the longest common subsequence between the two strings.<sup>[9]</sup>

```
divinities = ['Cheops', 'Osiris', 'Cleopatra',
              'Tutankhamun', 'Ramses', 'Bastet',
              'Anubis', 'Nefertiti']
divinities_lower = [x.lower() for x in divinities]

def match_query(query):
    matches = get_close_matches(query.lower(),
                                divinities_lower, n=1, cutoff=0.6)
    if matches:
        index = divinities_lower.index(matches[0])
        return divinities[index]
    return "No match found"
```

translation:  
**Tutankhamun**

Figure 10: Matching of the transliteration to 'Tutankhamun'.

## 4.8 Results

Our trained model was able to identify and correctly transliterate 7 out of 8 given examples of assembled hieroglyphs. In our context, “correctly” means that the model assigns the expected label to each glyph image. The translation yielded an accuracy of 6 out of 8, where the failed translations could not be translated due to the difference of the Egyptian terms to the modern English equivalents.

## 5 Summary

The objective of this project was successfully achieved. The system was able to recognize and transliterate Egyptian hieroglyphs as intended. This project provided us with valuable insights into the intersection of image preprocessing and natural language processing. Beyond visual recognition, the implementation treats the individual glyphs as tokens, does transliteration, fuzzy string

matching and sequence reconstruction, fundamental tasks in natural language processing.

Indeed there is still room for improvement. It could be possible to apply a better contour recognition in order to process other images with not so high contrasting background, or expand the model to process other hieroglyphs. This shows us that this interesting combined field of computer sciences always offers room for new ideas and improvements.

## Bibliography

- [1] British Museum, *Hieroglyphs: Unlocking Ancient Egypt*,  
<https://www.britishmuseum.org/exhibitions/hieroglyphs-unlocking-ancient-egypt>.
- [2] OpenCV, *Resizing and Rescaling Images with OpenCV*,  
<https://opencv.org/blog/resizing-and-rescaling-images-with-opencv/#h-resize-an-image-in-opencv>.
- [3] Geeks For Geeks, *Grayscale Of Images using OpenCV*,  
<https://www.geeksforgeeks.org/python-grayscale-of-images-using-opencv/>.
- [4] Science Direct, *Histogram Equalization*,  
<https://www.sciencedirect.com/topics/computer-science/histogram-equalization>.
- [5] Geeks For Geeks, *Image Processing in Python*,  
<https://www.geeksforgeeks.org/image-processing-in-python/>.
- [6] Medium, *All about HOG (Histogram of Oriented Gradients)*,  
<https://medium.com/@abhishekjainindore24/all-about-hog-histogram-of-oriented-gradients-869b5fab7bd5>.
- [7] Medium, *Multi-Class Logistic Regression: A Friendly Guide to Classifying the Many*,  
<https://medium.com/@jshaik2452>.
- [8] Imbalanced Learn, *KMeansSMOTE*,  
[https://imbalanced-learn.org/stable/references/generated/imblearn.over\\_sampling.KMeansSMOTE.html](https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.KMeansSMOTE.html).
- [9] Python Docs, *difflib — Helpers for computing deltas*,  
<https://docs.python.org/3/library/difflib.html>.