Images of LEGO Bricks Clustering

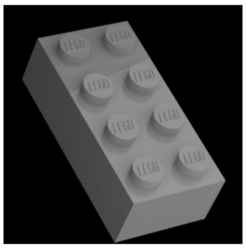
Bejenariu David-Cosmin
group 407

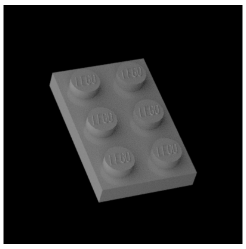
I. Introduction

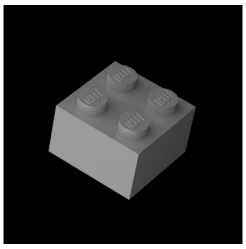
Images of LEGO Bricks Clustering is the second assignment of the Practical Machine Learning course. The task consists of finding an appropriate dataset for clustering and running two different algorithms while performing two different data preprocessing methods and hyperparameter tuning. The results should then be interpreted and compared with the performance of supervised models.

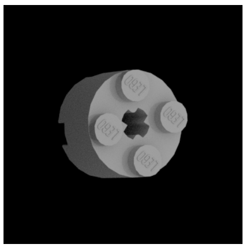
II. Working with the dataset

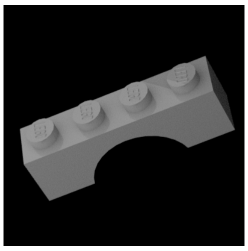
I opted for image clustering using a Images of LEGO Bricks dataset that contains 40000 images (400x400 px) of 50 different types of LEGO bricks, each class being equally split into 800 images each, where the bricks are viewed from different angles. For this particular task, I selected 8 from the entire pool of available classes, resulting in a total of 6400 samples. The chosen categories are highlighted below:

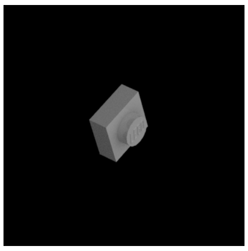

3001

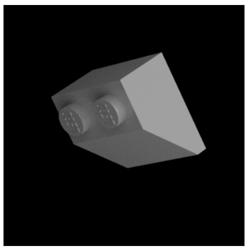

3021

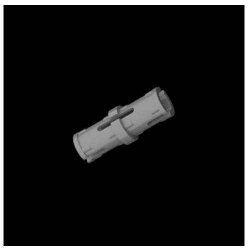

3003


6143


3659


3024


3039


2780

III. Data preprocessing

In an initial approach, different computer vision techniques as applying canny, blur or thresholding filters were applied. While the visualized results did not prove useful for this specific dataset, fitting the further processed images in a supervised model confirmed that a better approach was to skip this step. Thus, the following preprocessing methods were used to extract features from the images:

1. HOG descriptors

HOG (Histogram of Oriented Gradients) descriptors are feature descriptors widely used in computer vision and image processing for object detection and recognition tasks. The HOG descriptor represents the local distribution of gradient orientations in an image, capturing information about the structure, texture and shape of the object. This makes them useful in the process of feature vectors extraction that are fed into machine learning models for tasks such as classification and clustering.

For this particular task, I used a HOG configuration with (32, 32) pixels per cell ratio, (8, 8) cell per block ratio, L2 metric normalization and 8 orientations, extracting a total of 12800 features for each image.

2. ResNet processed features

ResNet (Residual Network) is a deep neural network architecture that introduced the concept of residual learning, allowing the training of very deep neural networks. ResNet architectures are known for their effectiveness in image classification tasks and have been used in various computer vision applications. Transfer learning with pre-trained ResNet models is advantageous because the lower layers have learned generic features from a diverse dataset, and these features can be valuable for tasks even if the target dataset is different.

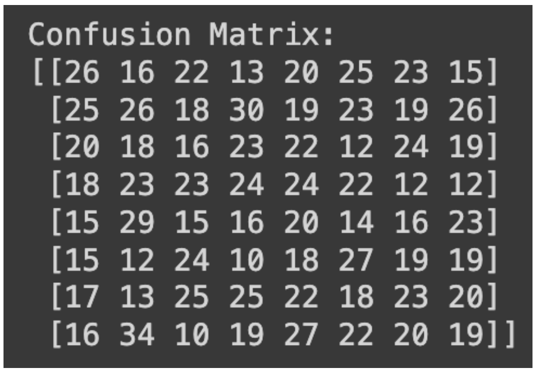
To extract features from this pre-trained neural network, the images were reshaped to 224x224 px and early preprocessed using the **preprocess_input()** method from *tensorflow.keras.applications* library, which ensures the data is standardazed and adjusted to match the pre-trained model requirements. The result was then fed into a **ResNet50** model from the *tensorflow* library mentioned before, resulting in 100352 features for each of the images.

IV. Supervised model training

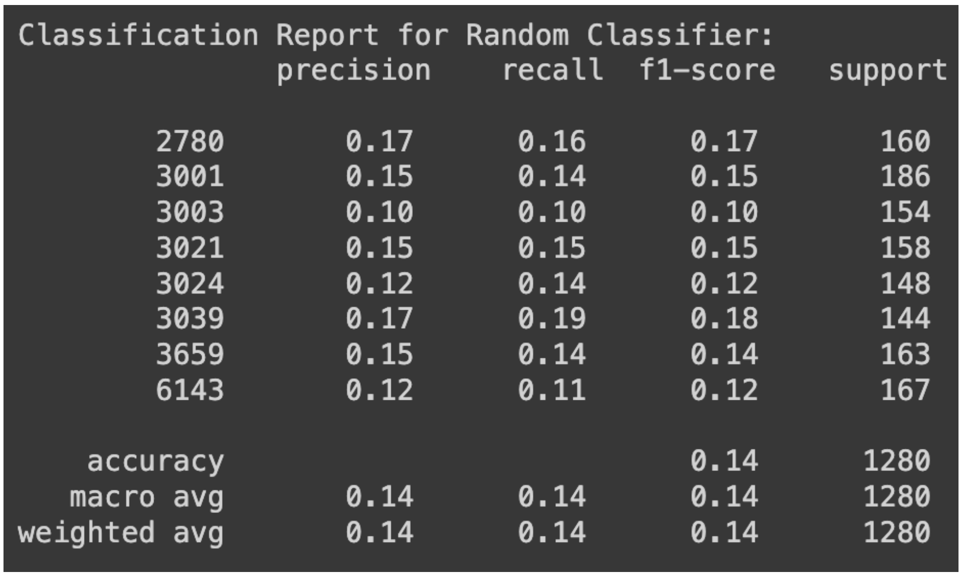
For this part of the assignment, it is required that two types of supervised models are trained and scored agains a train/test split for better comparison with the unsupervised outcome. The first model, specifically a random classifier, serves as a minimum requirement for final comparison with the unsupervised model and the other one, which can be any type of performant supervised classifier, would represent the best expected outcome.

1. DummyClassifier

The **DummyClassifier** from *scikit-learn* library was used for random classification. Results on both preprocessing methods were similarly bad, as expected, at around **11-14%** accuracy score.



```
Confusion Matrix:
[[26 16 22 13 20 25 23 15]
 [25 26 18 30 19 23 19 26]
 [20 18 16 23 22 12 24 19]
 [18 23 23 24 24 22 12 12]
 [15 29 15 16 20 14 16 23]
 [15 12 24 10 18 27 19 19]
 [17 13 25 25 22 18 23 20]
 [16 34 10 19 27 22 20 19]]
```

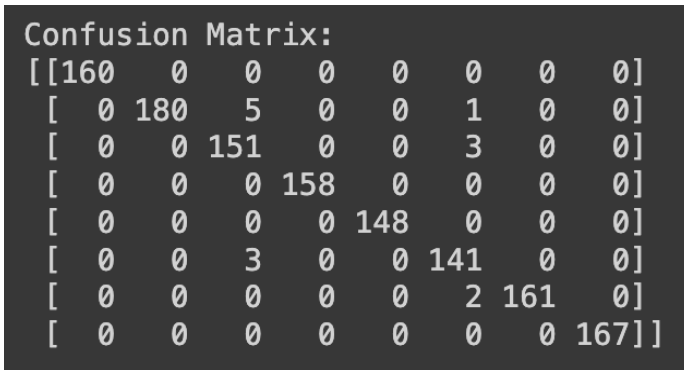


```
Classification Report for Random Classifier:
              precision    recall  f1-score   support

        2780       0.17      0.16      0.17       160
        3001       0.15      0.14      0.15       186
        3003       0.10      0.10      0.10       154
        3021       0.15      0.15      0.15       158
        3024       0.12      0.14      0.12       148
        3039       0.17      0.19      0.18       144
        3659       0.15      0.14      0.14       163
        6143       0.12      0.11      0.12       167

    accuracy                           0.14      1280
   macro avg       0.14      0.14      0.14      1280
weighted avg       0.14      0.14      0.14      1280
```

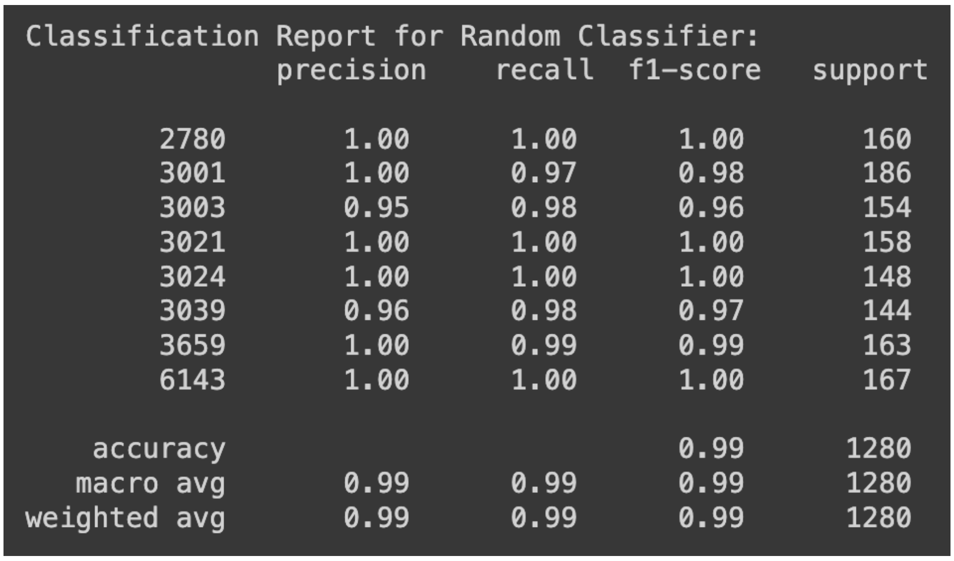
Confusion Matrix and Classification Report for DummyClassifier

2. SVM

The **SVM** (Support Vector Machines) classifier approach suited very well both preprocessing methods. For best results, I ran a **Grid Search** using different values for *C* and *kernel* parameters, resulting in the highest performance for the combination of **C=10** and **kernel='rbf'** parameters. Final accuracy testing scores were **95.16%** for HOG extracted features and **98.9%** for ResNet extracted features.



```
Confusion Matrix:
[[160   0   0   0   0   0   0   0]
 [  0 180   5   0   0   1   0   0]
 [  0   0 151   0   0   3   0   0]
 [  0   0   0 158   0   0   0   0]
 [  0   0   0   0 148   0   0   0]
 [  0   0   3   0   0 141   0   0]
 [  0   0   0   0   0   2 161   0]
 [  0   0   0   0   0   0   0 167]]
```



```
Classification Report for Random Classifier:
               precision    recall  f1-score   support

        2780       1.00      1.00      1.00       160
        3001       1.00      0.97      0.98       186
        3003       0.95      0.98      0.96       154
        3021       1.00      1.00      1.00       158
        3024       1.00      1.00      1.00       148
        3039       0.96      0.98      0.97       144
        3659       1.00      0.99      0.99       163
        6143       1.00      1.00      1.00       167

    accuracy                           0.99      1280
   macro avg       0.99      0.99      0.99      1280
weighted avg       0.99      0.99      0.99      1280
```

Confusion Matrix and Classification Report for SVM Classifier (ResNet features)

V. Unsupervised model scoring

Finally, for the main part of the current assignment, 2 different clustering algorithms should be ran against the already processed data and the results then be interpreted.

This task required an additional preparation step. Given that clustering algorithms could perform worse and also compute longer if the dimensionality of the data is too high, I took into consideration reducing the feature vectors of each sample by using PCA. **Principal Component Analysis** (PCA) is a dimensionality reduction technique widely used in data analysis and machine learning. It works by transforming the original features of a dataset into a new set of uncorrelated variables called principal components. The key steps in PCA include data standardization, covariance matrix calculation, eigendecomposition, selection of principal components and projection.

That being said, the algorithms were tested on both the full and reduced dimension versions of the dataset, against vectors of 100352, 6400, 3179, 2580 or 1789 features, based off the *n_dimensions* parameter of the PCA, computed by adjusting the cumulative variance ratio of the features.

For better results on both algorithms, I ran a custom **Grid Search** over various combinations of hyperparameters specific to each method.

1. K-means Clustering

The primary objective of K-means is to group data points that are similar to each other while minimizing the variance within each cluster. K-means is computationally efficient and can scale well to large datasets. What's more, the algorithm assumes that clusters are spherical and equally sized, which suits the dataset very well.

For this algorithm, the grid search selected the best combination for *n_clusters*, *n_init* and *init* hyperparameters

2. Agglomerative Clustering

Unlike K-means, which partitions the dataset into a fixed number of clusters, Agglomerative clustering starts with individual data points as separate clusters and iteratively merges them based on a specified linkage criterion. The linkage criterion determines the distance between clusters and influences the structure of the resulting hierarchy.

For this algorithm, I found useful to tune the *n_clusters*, *metric* and *linkage* hyperparameters.

VI. Results

- For evaluation, I used three different metrics: Silhouette score, Davies-Boulding score and finally, Adjusted Rand score for comparison with assigned truth labels.
- The HOG descriptors did not fit into the two algorithms very well, thus the Silhouette and Adjusted Rand score are positive but very close to zero, indicating not a very good performance compared to the randomized classifier.
- For ResNet extracted features, running against the full feature vectors resulted in high Silhouette scores for Agglomerative clustering, although later finding out that all the samples had been assigned to a single cluster. Silhouette scores were generally low for each length of the feature vectors, but I managed to score better using the Adjusted Rand score which indicated the highest value of 0.45283003361857477 for K-mean with parameters: *{'init': 'random', 'n_clusters': 8, 'n_init': 10}*.
- Other relevant scores for K-means using a feature vector of length 6400:

Iteration 1/30 - Silhouette: 0.049430347979068756, Davies-Bouldin: 3.3636011995736554, Adjusted Rand: 0.2544331931955409 - Params: {'init': 'k-means++', 'n_clusters': 4, 'n_init': 10}

Iteration 2/30 - Silhouette: 0.04949737712740898, Davies-Bouldin: 3.354169863384886, Adjusted Rand: 0.26539011126879153 - Params: {'init': 'k-means++', 'n_clusters': 4, 'n_init': 20}

```
Iteration 3/30 - Silhouette: 0.04949895292520523, Davies-Bouldin: 3.354347568024218, Adjusted
Rand: 0.2652233413053 - Params: {'init': 'k-means++', 'n_clusters': 4, 'n_init': 30}

Iteration 4/30 - Silhouette: 0.0639672577381134, Davies-Bouldin: 3.2373666662328615, Adjusted
Rand: 0.3310110487860859 - Params: {'init': 'k-means++', 'n_clusters': 5, 'n_init': 10}

Iteration 5/30 - Silhouette: 0.06399770826101303, Davies-Bouldin: 3.2367054707420473, Adjusted
Rand: 0.3306106956237716 - Params: {'init': 'k-means++', 'n_clusters': 5, 'n_init': 20}

Iteration 6/30 - Silhouette: 0.0639968290925026, Davies-Bouldin: 3.238486862139748, Adjusted
Rand: 0.3305580053471025 - Params: {'init': 'k-means++', 'n_clusters': 5, 'n_init': 30}

Iteration 7/30 - Silhouette: 0.062305957078933716, Davies-Bouldin: 3.199479554964459, Adjusted
Rand: 0.44419016014390716 - Params: {'init': 'k-means++', 'n_clusters': 6, 'n_init': 10}

Iteration 8/30 - Silhouette: 0.061996422708034515, Davies-Bouldin: 3.1896637159604286,
Adjusted Rand: 0.4176163511433063 - Params: {'init': 'k-means++', 'n_clusters': 6, 'n_init': 20}

Iteration 9/30 - Silhouette: 0.06209462136030197, Davies-Bouldin: 3.192974574927191, Adjusted
Rand: 0.41953059475997845 - Params: {'init': 'k-means++', 'n_clusters': 6, 'n_init': 30}

Iteration 10/30 - Silhouette: 0.06789650768041611, Davies-Bouldin: 3.0126856480128863, Adjusted
Rand: 0.4112852800588723 - Params: {'init': 'k-means++', 'n_clusters': 7, 'n_init': 10}

Iteration 11/30 - Silhouette: 0.06774341315031052, Davies-Bouldin: 3.01058082739145, Adjusted
Rand: 0.41240609985235177 - Params: {'init': 'k-means++', 'n_clusters': 7, 'n_init': 20}

Iteration 12/30 - Silhouette: 0.06787282228469849, Davies-Bouldin: 3.009645903984185,
Adjusted Rand: 0.41225148031265313 - Params: {'init': 'k-means++', 'n_clusters': 7, 'n_init': 30}

Iteration 13/30 - Silhouette: 0.0659775584936142, Davies-Bouldin: 3.188564449358389, Adjusted
Rand: 0.45125964093629595 - Params: {'init': 'k-means++', 'n_clusters': 8, 'n_init': 10}

Iteration 14/30 - Silhouette: 0.06606848537921906, Davies-Bouldin: 3.1899447011018296, Adjusted
Rand: 0.4519286891805997 - Params: {'init': 'k-means++', 'n_clusters': 8, 'n_init': 20}

Iteration 15/30 - Silhouette: 0.06589320302009583, Davies-Bouldin: 3.1897818592330465,
Adjusted Rand: 0.45196174371113035 - Params: {'init': 'k-means++', 'n_clusters': 8, 'n_init': 30}

Iteration 16/30 - Silhouette: 0.04777228459715843, Davies-Bouldin: 3.4911859717683034, Adjusted
Rand: 0.2924596821026274 - Params: {'init': 'random', 'n_clusters': 4, 'n_init': 10}

Iteration 17/30 - Silhouette: 0.04777228459715843, Davies-Bouldin: 3.4911859717683034, Adjusted
Rand: 0.2924596821026274 - Params: {'init': 'random', 'n_clusters': 4, 'n_init': 20}
```

Iteration 18/30 - Silhouette: 0.049479998648166656, Davies-Bouldin: 3.3526292038437515, Adjusted Rand: 0.2648817847189867 - Params: {'init': 'random', 'n_clusters': 4, 'n_init': 30}

Iteration 19/30 - Silhouette: 0.0639968290925026, Davies-Bouldin: 3.238486862139748, Adjusted Rand: 0.3305580053471025 - Params: {'init': 'random', 'n_clusters': 5, 'n_init': 10}

Iteration 20/30 - Silhouette: 0.06399308890104294, Davies-Bouldin: 3.2386250040780205, Adjusted Rand: 0.3304551316526535 - Params: {'init': 'random', 'n_clusters': 5, 'n_init': 20}

Iteration 21/30 - Silhouette: 0.0639968290925026, Davies-Bouldin: 3.238486862139749, Adjusted Rand: 0.3305580053471025 - Params: {'init': 'random', 'n_clusters': 5, 'n_init': 30}

Iteration 22/30 - Silhouette: 0.06208803132176399, Davies-Bouldin: 3.1907137919926214, Adjusted Rand: 0.41972747798212406 - Params: {'init': 'random', 'n_clusters': 6, 'n_init': 10}

Iteration 23/30 - Silhouette: 0.06193634495139122, Davies-Bouldin: 3.191292530773102, Adjusted Rand: 0.4202216873662921 - Params: {'init': 'random', 'n_clusters': 6, 'n_init': 20}

Iteration 24/30 - Silhouette: 0.06208803132176399, Davies-Bouldin: 3.190713791992622, Adjusted Rand: 0.41972747798212406 - Params: {'init': 'random', 'n_clusters': 6, 'n_init': 30}

Iteration 25/30 - Silhouette: 0.0679742619395256, Davies-Bouldin: 3.0110539582531115, Adjusted Rand: 0.40803505037681376 - Params: {'init': 'random', 'n_clusters': 7, 'n_init': 10}

Iteration 26/30 - Silhouette: 0.06775093078613281, Davies-Bouldin: 3.009707576669616, Adjusted Rand: 0.4123973857099791 - Params: {'init': 'random', 'n_clusters': 7, 'n_init': 20}

Iteration 27/30 - Silhouette: 0.06777946650981903, Davies-Bouldin: 3.010475435754786, Adjusted Rand: 0.41223353762833054 - Params: {'init': 'random', 'n_clusters': 7, 'n_init': 30}

Iteration 28/30 - Silhouette: 0.06597146391868591, Davies-Bouldin: 3.188672777509006, Adjusted Rand: 0.45283003361857477 - Params: {'init': 'random', 'n_clusters': 8, 'n_init': 10}

Iteration 29/30 - Silhouette: 0.06598387658596039, Davies-Bouldin: 3.1878087836355355, Adjusted Rand: 0.45223773876803186 - Params: {'init': 'random', 'n_clusters': 8, 'n_init': 20}

Iteration 30/30 - Silhouette: 0.06588892638683319, Davies-Bouldin: 3.188901464702057, Adjusted Rand: 0.4521205557646146 - Params: {'init': 'random', 'n_clusters': 8, 'n_init': 30}

- Relevant scores for Agglomerative clustering with all ResNet features include:

Iteration 1/15 - Score: -0.08018693327903748 - Params: {'affinity': 'euclidean', 'linkage': 'ward'}

Iteration 2/15 - Score: 0.7216450572013855 - Params: {'affinity': 'euclidean', 'linkage': 'complete'}

Iteration 3/15 - Score: 0.7364316582679749 - Params: {'affinity': 'euclidean', 'linkage': 'average'}

```
Iteration 1/8 - Score: 0.4038870334625244 - Params: {'affinity': 'l1', 'linkage': 'complete'}

Iteration 2/8 - Score: 0.719290018081665 - Params: {'affinity': 'l1', 'linkage': 'average'}

Iteration 3/8 - Score: 0.7216450572013855 - Params: {'affinity': 'l2', 'linkage': 'complete'}

Iteration 4/8 - Score: 0.7364316582679749 - Params: {'affinity': 'l2', 'linkage': 'average'}

Iteration 5/8 - Score: 0.4038870334625244 - Params: {'affinity': 'manhattan', 'linkage': 'complete'}

Iteration 6/8 - Score: 0.719290018081665 - Params: {'affinity': 'manhattan', 'linkage': 'average'}

Iteration 1/2 - Score: -0.043978024274110794 - Params: {'affinity': 'cosine', 'linkage': 'complete'}

Iteration 2/2 - Score: -0.037424106150865555 - Params: {'affinity': 'cosine', 'linkage': 'average'}
```