Tasks 2 and 3 Report

Computer Vision

up202008462@fe.up.pt José Luís Cunha Rodrigues
up202008866@fe.up.pt Guilherme Cunha Almeida
up202005097@fe.up.pt Pedro Jorge da Rocha Balazeiro

June 12, 2024

1 Task 2

The second task of this project aims to identify the number of LEGO bricks on an image by employing Deep Learning Models based on CNN architectures.

1.1 DataSet

The provided dataset contained images with LEGOS in different quantities, from 1 LEGO to 32 LEGOS in an image. The disparity of the samples in each class was significant, given that the images with 1 and 2 LEGOS make up about 93% of the 2933 images. The chart representing the data distribution is shown on Figures B.1 and B.2.

1.2 Data Splitting

From the entirety of the dataset [1], it was necessary to define the samples used for model evaluation as a test set. This split was made using the information present in a .csv file provided in the project description. The remainder of the dataset was then divided into train and validation sets, using a random split of 80%-20%, respectively. The use of the validation set allowed for the most effective training of models in order to detect overfitting situations and evaluate the model performance during training to adjust hyperparameter settings.

1.3 Data sampling and augmentation

To deal with the problem of unbalanced data, the selected approach required diminishing the rate of unbalance by truncating the number of used samples in training for images with 1 or 2 bricks. This way, only a maximum of 50 images from each class were used, leading to a lower stratification ratio between classes and leading to a less skewed and biased resulting model.

However, given the remaining large disparity between the number of samples in most of the classes and the necessity of a higher amount of data for the successful training of Deep Learning models, a Data Augmentation strategy was adopted in the training set. In the less represented classes (classes 3 to 32), the samples were augmented by creating new images derived randomly from the rotation, flipping, and brightness adjustment of the existing ones. An example of a Data Augmentation result can be seen in Figure B.3.

Some classes didn't have a single photo in the train set and thus, augmentation was not enough to mitigate the derived problems. To tackle this issue, a process of manual generation of photos was conducted, using photo editing software. This process included erasing pieces, adding new ones, and adjusting the image's color levels. An example of this process can be seen in Figure B.4.

This process resulted in a dataset that, although still unbalanced, comprised enough samples of each class to allow for the training of the models. This way, and despite not having solved the problem of the lack of diversity and quantity of relevant data, successful engineering solutions were employed to try to battle it.

1.4 Models

To solve the given problem, CNN-based architectures were tested. The chosen models were employed in a transfer learning regimen, by using the pre-trained weights and adapting the model to the current problem. The latter was modeled both as a multiclass and regression problem, as explained below.

1.4.1 CNN Architectures

One of the most popular networks in Computer Vision is the ResNet model. This architecture model presents good results in these tasks due to the residual nature of its architecture and short-cutting of relevant details between layers, leading to better identification of key characteristics and solving the vanishing gradient problems. Several versions of this architecture are currently available for use in tranfer learning tasks, such as ResNet18, ResNet34, ResNet50. The adequacy of each model depends on the complexity of the task at hand and the nature of the features necessary for classification and regression. In this project, the ResNet18 and ResNet50 models were trained.

Secondly, one of the most relevant Deep Learning models, the VGG, was tested due to its simplicity and effectiveness in image classification tasks. This model is known for its deep architecture, which consists of 16 or 19 layers of convolutional neural networks and very small convolution filters, which help in capturing the fine details in images. One of the key advantages of VGG is its uniform architecture, making it easy to understand and implement. Additionally, it has shown strong performance in various image Deep Learning benchmarks.

The Batch Normalization (VGG-BN) version of the VGG network, with the inclusion of batch normalization layers, was also tested. The VGG-BN model incorporates batch normalization after each convolutional layer, which helps in stabilizing and accelerating the training process.

On another model attempt, Inception v3, also known as GoogLeNet, was used for this task. It is designed to perform image classification by utilizing a complex architecture that includes multiple types of convolutions within the same layer, as well as pooling operations. This approach allows the network to capture features at various scales and complexities.

Moreover, the LeNet architecture was also used. This network pioneered the area of convolutional neural networks in the late 1980s. LeNet features a relatively simple architecture with two convolutional layers followed by subsampling (pooling) layers, and then two fully connected layers. Its simplicity and efficiency paved the way for modern deep-learning models, demonstrating the effectiveness of convolutional layers in extracting spatial hierarchies in image data. The usage of this simpler architecture can be taken as a benchmark for the results and used for the comparison with more advanced Deep Learning Technologies, as the ones presented earlier.

Lastly, it was explored the DenseNet architecture. DenseNet, or Densely Connected Convolutional Network, is known for the dense connections between layers, where each layer receives inputs from all preceding layers and passes its output to all subsequent layers. This direct connection scheme ensures maximum information flow between layers, leading to more efficient feature reuse and a significant reduction in the number of parameters. DenseNet has shown a positive performance in various image classification and regression tasks, particularly due to its ability to alleviate the vanishing gradient problem and to better propagate features and gradients throughout the network.

1.5 Classification

This approach consisted of adapting 32 classes, one for each possible amount of bricks in an image. Following the Transfer Learning strategy, the training of each model involved the definition of a final Fully Connected Layer in each model with 32 output nodes. The loss function used was Cross Entropy.

1.6 Regression

The problem at hand can also be interpreted as a regression task, by adapting the continuous nature of this strategy to a discrete output. This can be done by defining a fully connected layer with a single output node, that returns the number of pieces found. This result can then be rounded to obtain the value of bricks in the image. Mean Squared Error was used as the loss function.

1.7 Training and Evaluation

For an adequate amount of epochs, determined by the convergence of training loss and validation loss, only this layer was trained, with the weights of the last layer updating using the Adam optimizer algorithm. Then, the process was repeated including the training of the whole network. The values for learning rate were obtained by empirical testing of each model and the batch size was defined as 32 for every model.

1.8 Results

To determine the number of epochs corresponding to the best-performing model, with a higher potential of obtaining a good performance when using generalized data, the Early Stopping Technique was used. This consists of storing the model that achieves the highest value of accuracy in the validation set, corresponding to the convergence of the training and validation results, which occurs before the overfitting of the training set, where accuracy in training increases but accuracy in validation decreases. Given the unbalanced nature of the dataset used, it is important to employ such techniques to reduce overfitting and detain the models from classifying only the majority classes. All the selected models were trained as classification and regression under similar conditions. The results obtained are represented in Tables 1 and 2.

Table 1: Results obtained for all classification models.

Model	**Accuracy**	**Balanced Accuracy**	**Precision**	**Recall**	**F1**	**MAE**	**MSE**
ResNet18	0.723	0.193	**0.809**	0.723	0.754	1.821	17.975
ResNet50	0.770	0.146	0.781	0.770	0.775	1.376	6.033
VGG16	0.697	0.116	0.681	0.697	0.684	2.639	33.938
VGG16 BN	0.672	0.204	0.796	0.672	0.720	1.646	13.719
VGG19	0.453	0.075	0.702	0.453	0.542	2.883	35.861
VGG19 BN	0.642	0.185	0.787	0.642	0.698	1.745	15.672
Inception V3	0.690	0.262	0.803	0.690	0.734	1.260	5.405
LeNet	0.777	0.174	0.815	0.777	**0.793**	1.263	4.212
DenseNet	0.785	0.173	0.808	0.785	0.791	1.383	6.646

Table 2: Results obtained for all regression models.

Model	**Accuracy**	**Balanced Accuracy**	**Precision**	**Recall**	**F1**	**MAE**	**MSE**
ResNet18	0.756	**0.293**	0.751	0.756	0.748	0.894	2.800
ResNet50	0.763	0.199	0.760	0.763	0.758	1.078	3.479
VGG16	0.646	0.164	0.753	0.646	0.692	1.603	5.090
VGG16 BN	0.752	0.120	0.706	0.752	0.723	0.946	2.711
VGG19	0.624	0.063	0.705	0.624	0.662	1.587	5.581
VGG19 BN	0.741	0.101	0.718	0.741	0.726	0.979	2.917
Inception V3	0.745	0.107	0.752	0.745	0.747	0.859	2.220
LeNet	0.781	0.159	0.730	0.781	0.752	0.663	**2.007**
DenseNet	**0.796**	0.178	0.768	**0.796**	0.770	**0.615**	2.233

Considering these results, the best-performing model was the regression DenseNet, having scored higher in more metrics. Given this information, this model was then fine-tuned to obtain the final submission algorithm. This process resulted in a better-performing model, with evaluation metrics shown in Figure B.5.

The accuracy value represents a positive value achieved, with 0.80. Nevertheless, it is possible to analyze through the value of balanced accuracy of 0.26, and Confusion Matrix, represented in Figure B.6, that the model is mostly successful when predicting the majority classes. This fact is resulting from the dataset quality, and although efforts have been made to overcome this shortcoming, more work should be approached on this front.

1.9 Future Works

When considering future possibilities for improving the current work, several options are possible. Firstly, the employment of more distinct and state-of-the-art networks is always a possibility, along with the creation of new CNN and perform the entire training in this dataset. However, the shortcoming of limited data will always yield significant importance and be a limitation when simply exploring new models. To overcome this problem, the conjunction of other datasets for training and testing of these models could help improve training and enhance test performance. This could be done by diversifying the datasets used by searching the Web for LEGO piece pictures or even producing a new dataset by manually taking pictures of different numbers of LEGOs with different intensities and guaranteeing an even number of samples for each class.

2 Task 3

Task 3 can be divided into two sub-tasks: brick detection and brick segmentation.

2.1 Task 3.1 - Brick Detection

2.1.1 Introduction

This subtask focused on detecting LEGO pieces using object detection models. Several different models were explored (YOLOv8, Faster R-CNN) along with training setups, culminating in the comparison of their performance using relevant metrics.

2.2 Implementation

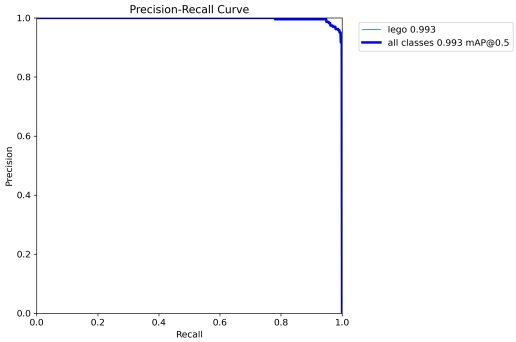
In an initial phase, ultralytics' YOLO framework was implemented. For context, YOLO (You Only Look Once) is a real-time object detection and image segmentation framework that contains several pre-trained models.

The first training session was performed on top of the base YOLOv8s model, a smaller and faster version of YOLO. The model was trained for **20 epochs** with images **resized to 640x640** pixels. The provided *Bridge of Knowledge* dataset was used, focusing only on the **photos containing 1 Lego piece** (2392 images). The dataset was splitted into train, validation and test sets with proportions of 70%, 20%, 10%. This initial step was designed to assess the model potential, while reducing the time spent on this training.

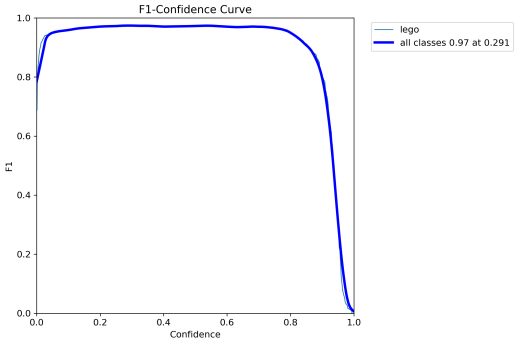
As it can be observed in figure 2.1, the results from the first training iteration were promising (these results correspond to the validation regarding the testing images split):

- the mean average precision with a intersection over union (IoU) threshold of 50% (mAP@50), presents a score of **99%**, as it can be observed in figure 2.1a. This indicates the model's effectiveness at both correctly identifying the presence of a LEGO piece, as well as locating the bricks with minimal false positives. Additionally, considering a more strict metric, **mAP@50-95**, which measures the mean average precision across different thresholds, the model achieved a score of **93.3%**.

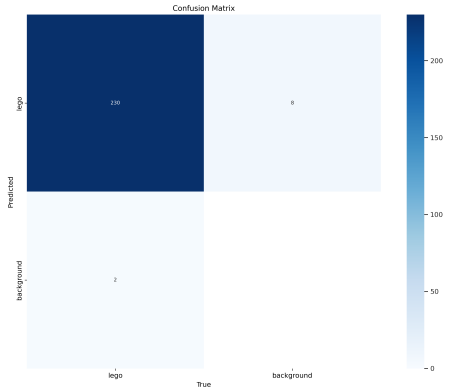
- the F1 curve, represented in Figure 2.1b, illustrates that an F1 score of **0.97** was achieved at a confidence threshold of **0.291**. This result is an indication that the model is able to effectively balance precision and recall. The confusion matrix, displayed in Figure 2.1c, presents only 8 false positives and 2 false negatives, further cementing the model's strong performance



(a) Precision and Recall Curve



(b) F1 Curve

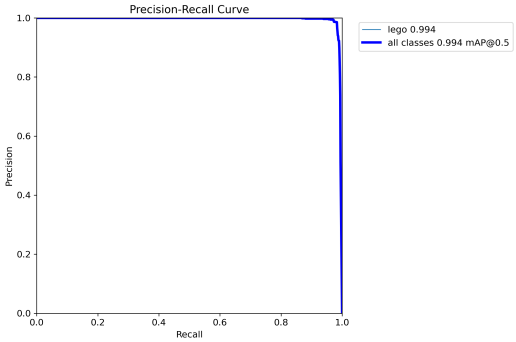


(c) Confusion Matrix

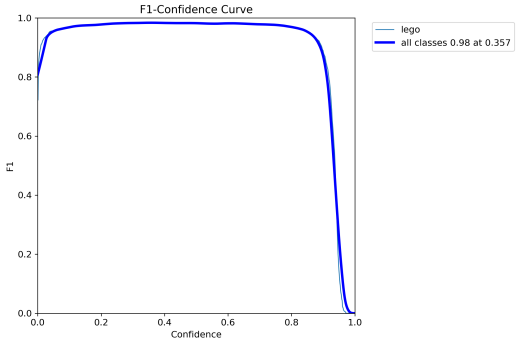
Figure 2.1: First Training Results

The next round of training has as main objectives the improvement of the model by training it for longer and using a larger sample of the data. The dataset used in this task comprised images whihch contained between **1 and 15 LEGO** pieces, with a size of **2903 images** in total. The model was trained for **100 epochs** with pictures resized to **640x640 pixels**. This training was designed to help the model identify LEGO pieces in various scenarios.

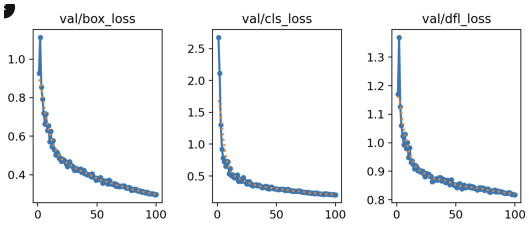
In Figure 2.2, the results of the second training iteration are displayed. The precision and recall curve, represented in Figure 2.2a, shows a slight improvement in the **mAP@50** metric, indicating that this training setup was able to produce a slightly more accurate model. The f1 curve, 2.2b, also reflects this enhancement, with a higher F1 score at various confidence thresholds. In the **mAP@50-95** metric, the model obtained a score of **0.95976**, a considerably higher value when comparing to the first training iteration, demonstrating that the model's accuracy across different thresholds has increased. The validation loss values also present good results, indicating that the model is not overfitting and generalizes well to unseen data.



(a) Precision and Recall Curve



(b) F1 Curve



(c) Confusion Matrix

Figure 2.2: Second Training Results

One final training session was conducted. Given the steady decrease in loss function during the previous training, a natural decision of increasing the number of epochs to **200** was done and the **'patience'** parameter was set to **10**. This setup was chosen to allow the model to benefit from more extensive learning while preventing overfitting by altering the patience parameter. This parameter ensures that training stops early if the model does not improve its performance for 10 consecutive epochs.

The results from this final session were similar to the ones obtained in the previous training session. The training was automatically stopped at **epoch 133** due to the having reached maximum patience. The best results were observed at **epoch 123**.

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    133/200      4.09G     0.4374     0.3288     0.9103         24        640: 100%|██████████| 129/129 [00:25<00:00,  5.10it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 19/19 [00:05<00:00,  3.57it/s]                   all        581        904
EarlyStopping: Training stopped early as no improvement observed in last 10 epochs. Best results observed at epoch 123, best model saved as best.pt.
To update EarlyStopping(patience=10) pass a new patience value, i.e. `patience=300` or use `patience=0` to disable EarlyStopping.


133 epochs completed in 1.076 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 22.5MB
Optimizer stripped from runs/detect/train/weights/best.pt, 22.5MB

Validating runs/detect/train/weights/best.pt...
Ultralytics YOLOv8.2.29 🚀 Python-3.10.12 torch-2.3.0+cu121 CUDA:0 (NVIDIA L4, 22700MiB)
Model summary (fused): 168 layers, 11125971 parameters, 0 gradients, 28.4 GFLOPs
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/conv.py:456: UserWarning: Plan failed with a cudnnException: CUDNN_BACKEND_EXECUTION_PLAN_DESCRIPTOR: cudnnFinalize Descriptor F
  return F.conv2d(input, weight, bias, self.stride,
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 19/19 [00:08<00:00,  2.29it/s]
                   all        581        904      0.986      0.982      0.994      0.952
Speed: 1.9ms preprocess, 4.1ms inference, 0.0ms loss, 2.0ms postprocess per image
Results saved to runs/detect/train
```

Figure 2.3: Training 3 Screenshot

2.2.1 Conclusion

Initially, there were concerns regarding the existent processing power and time was able to achieve satisfactory results, which later were proved to be unfounded. The final training session, with 200 epochs, was completed in approximately 1.07 hours and produced positive outcomes. However, the authors of this project had to invest some of their own money to be able to perform the extensive training within a reasonable time, being forced to buy GPU time in Google Colab. Although the Faster R-CNN model was also explored, its complexity led to a greater focus on the YOLO framework.

2.3 Task 3.2 - Traditional segmentation

Introduction

This subsection documents the methodology used for the segmentation task. The primary objective was to perform traditional segmentation within bounding boxes derived from object detection done in the task before, followed by a qualitative assessment of the segmentation quality.

Methodology

The segmentation process commenced with the selection of a traditional method, with k-means chosen as the preferred approach. Manual hyperparameter tuning was conducted to optimize segmentation quality.

The optimal number of clusters was determined to be five, balancing the risk of oversimplification with that of over-segmentation.

- Fewer clusters: The resulting merge of distinct regions into the same cluster posed significant challenges, particularly in scenarios characterized by similar backgrounds and a high presence of shadows. Consequently, this phenomenon led to the loss of crucial details and a deterioration in segmentation quality. Specifically, the amalgamation of disparate objects within a single cluster resulted in inaccurate segmentation outcomes.

- More clusters: Using a higher number of clusters than warranted led to an over-segmentation of the image, wherein even regions with subtle similarities were partitioned into separate clusters. This outcome introduced an excess of detail and induced the appearance of noise within the segmentation.

Through empirical efforts, ten iterations were found to be optimal, balancing convergence with computational efficiency.

An epsilon value of 0.2 was chosen to facilitate convergence without premature termination.

To mitigate noise and enhance clustering effectiveness, Gaussian blur was applied. Parameter tuning determined a (5,5) kernel as optimal, striking a balance between noise reduction and preservation of essential details (smaller kernels resulted in minor noise reduction).

Consequent challenges and improvements

Several challenges were identified during the segmentation process, including shadow interference, background similarity, and transparency issues. To address these challenges, several strategies were employed:

- Color Space Transformation and Histogram Equalization: Conversion to the LAB color space followed by histogram equalization of the L channel was performed to enhance contrast and mitigate shadow interference. Alternative approaches, such as HSV color space conversion and equalization in the value channel, were explored but yielded comparable results and were subsequently discarded. Implementing Contrast Limited Adaptive Histogram Equalization (CLAHE) presented itself as an alternative solution in this scenario. However, its application resulted in an amplified presence of noise, thereby rendering this option unsuitable for further consideration.

- Foreground Extraction with Grab Cut: The Grab Cut algorithm was employed to improve foreground extraction and mitigate background similarity issues. This approach significantly enhanced the accuracy of segmenting LEGO pieces from similar backgrounds and helped in the observation of the LEGO segmentation.

- Threshold Techniques: Various threshold methods, including global binary threshold, Otsu's method, and adaptive threshold, were applied to refine segmentation results and enhance output quality (all of them showed similar results).

- Morphological Operations: Post-segmentation morphological operations, such as opening to remove small objects and closing to fill small holes, were attempted to improve segmentation accuracy. However, no significant improvements were observed.

Observation: The conversion of the cropped image, specifically the bounding box, to the HSV color space was initially employed to enhance color differentiation, effectively distinguishing between elements and similar backgrounds (not entirely). However, when integrated with the subsequent implementation of the Grab Cut algorithm, it did not yield satisfactory outcomes. Consequently, this approach was subsequently discarded from consideration.

Qualitative evaluation

The segmentation results were qualitatively evaluated to assess both the strengths and weaknesses of the approach. While the methodology succeeded in effectively segmenting LEGO pieces from complex backgrounds and mitigating shadow interference to some extent, challenges such as transparency issues, minor noise, bounding boxes overlap and LEGOs that resemble hollow tires persisted (the cavity is mistakenly identified as a component of the LEGO itself). In Annex B, there are examples of the qualitative evaluation in Gigures B.7, B.8, B.9 and B.10.

Conclusions and future work

So, in conclusion, the segmentation task employed a systematic approach combining traditional segmentation methods with advanced techniques to address specific challenges inherent to LEGO piece segmentation. While notable progress was achieved in mitigating shadow interference and background similarity, further optimization is warranted to address the issues mentioned. Future efforts will focus on refining the segmentation pipeline to enhance overall accuracy, and robustness and try to solve the current challenges like removing the overlap in the bounding boxes. Some illumination-related techniques may be selected to deal better with the shadows, further hyperparameter tuning can be done and also testing with other traditional segmentation techniques.

References

[1] K. Zawora, S. Zaraziński, B. Śledź, B. Łobacz, and T. Boiński. Tagged images with lego bricks, 2021.

A Program Instructions

The resulting program for task 2 is stored under `script.py`. It uses the model under `model.pth` to make predictions on an input file and can be run as:

```
python script.py <input_file>

# Example
python main.py input.json
```

, where `input.json` follows the rules specified for the first deliverable.

Regarding task 3, in order to run the notebook correctly, you need to have access to the Google Drive folder containing all the necessary files. Therefore, the notebook is simply to illustrate the implementation and not to run the actual code.

B Images

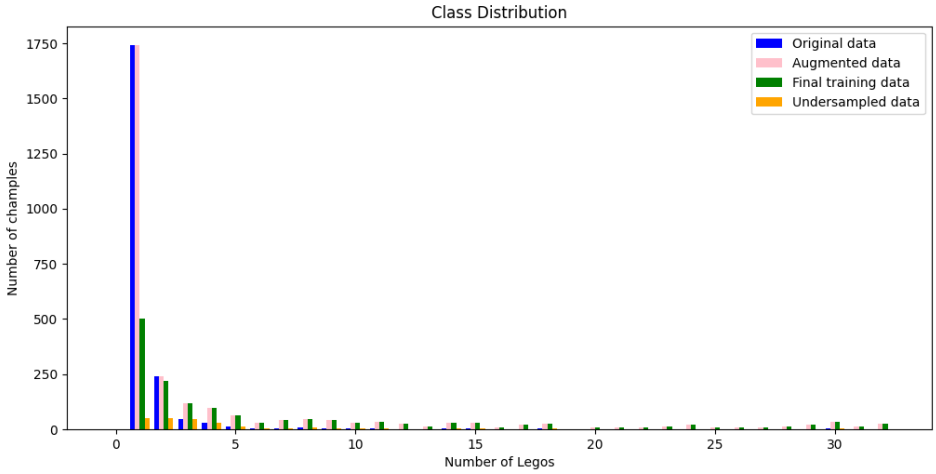


Figure B.1: The chart shows the imbalance of classes in the dataset. This can be seen by the blue line and the significant difference for classes 1 and 2.

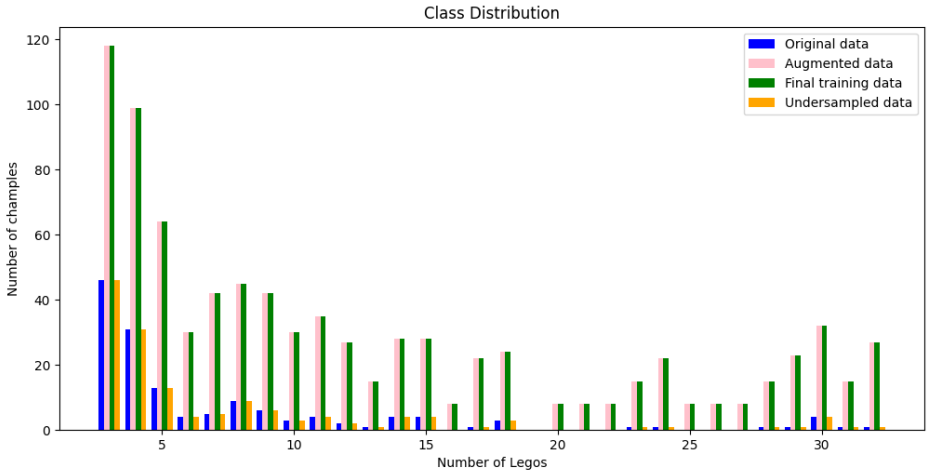
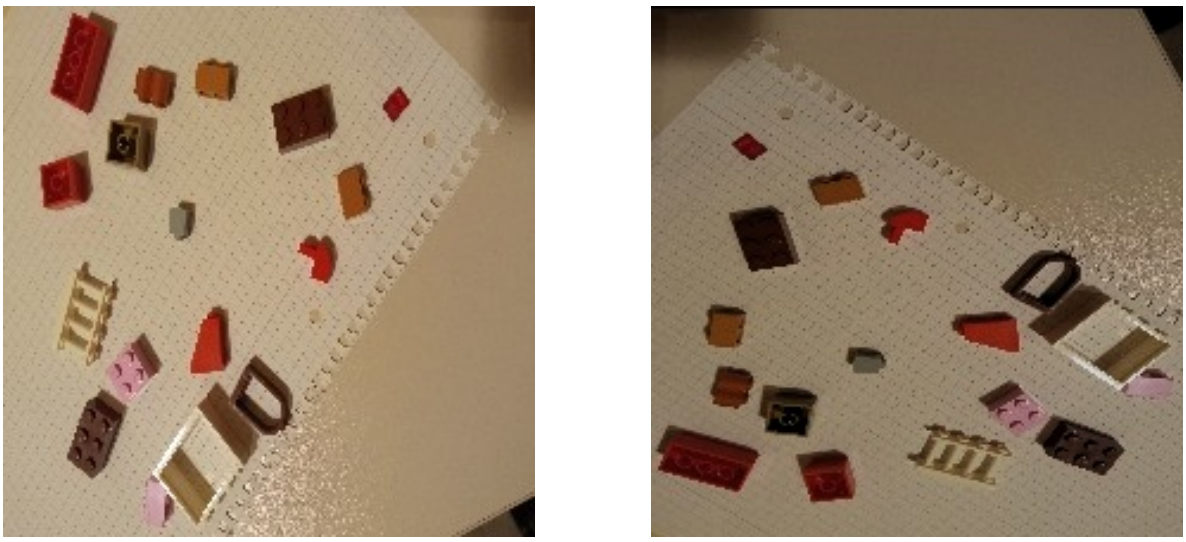


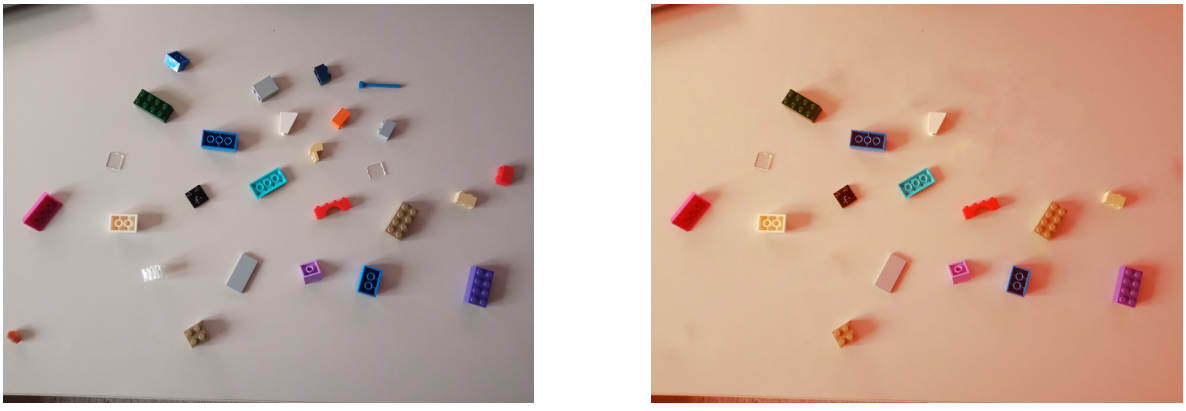
Figure B.2: This graph is similar to B.1, with the difference that the imbalanced classes were removed for better visualization. Its possible to understand the changes made from the original data to the different approaches taken.



(a) Original

(b) Augmented example

Figure B.3: Example photo obtained through the augmentation pipeline



(a) Original

(b) Generated example

Figure B.4: Example photo generated by hand

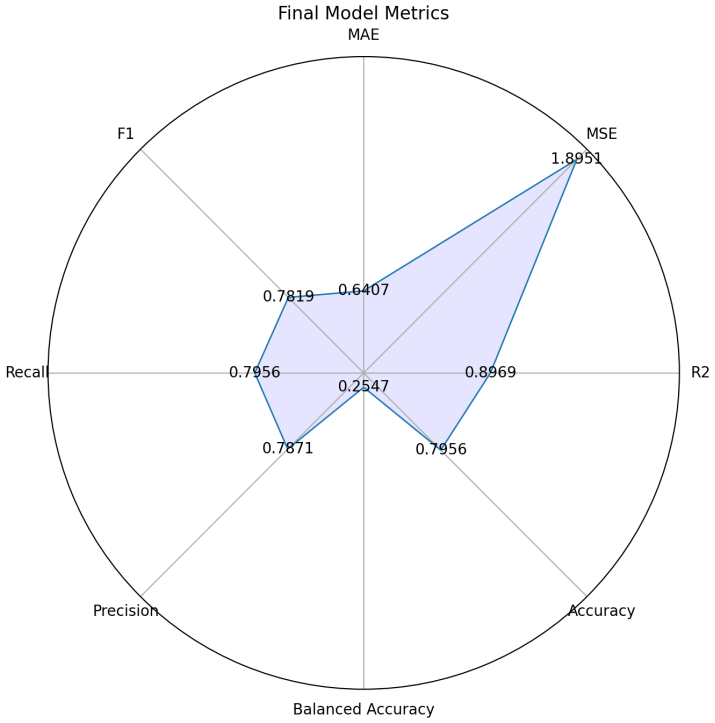


Figure B.5: The plot shows the metrics obtained for the final submission model.

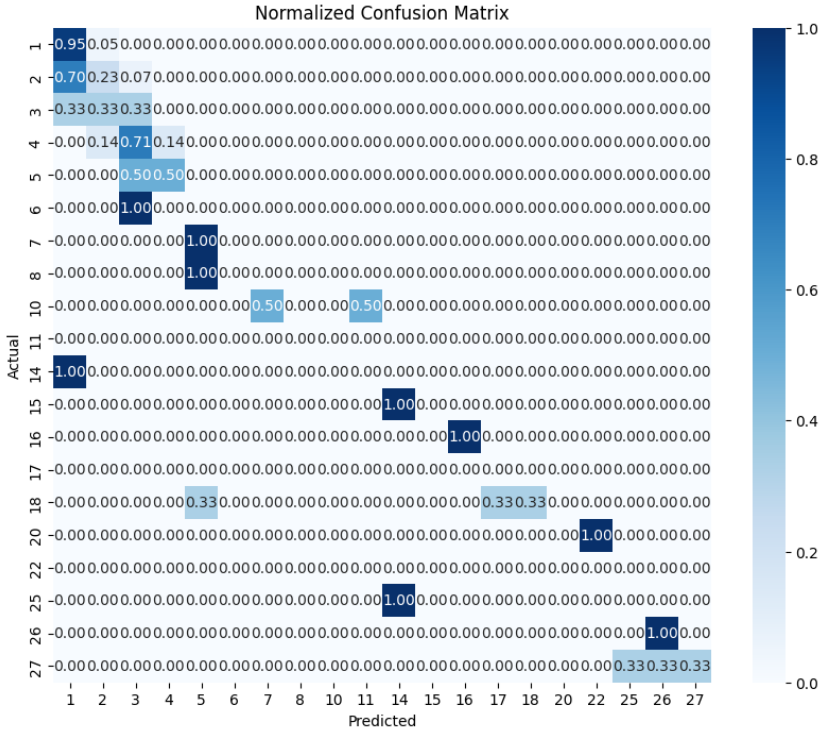
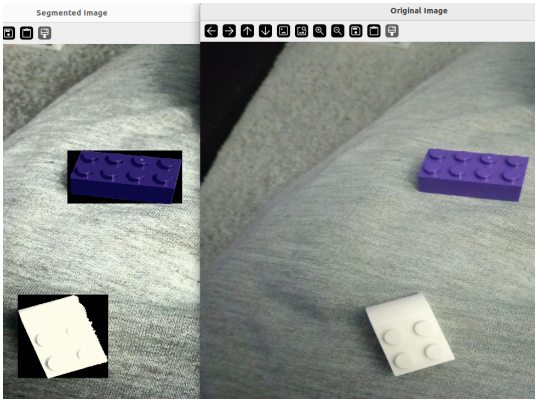
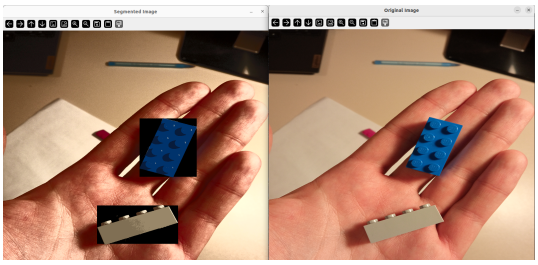


Figure B.6: The normalized confusion matrix obtained for the submission model shows a high percentage of correct classifications for the diagonal. Nevertheless, the skewed results are yet another evidence of the data imbalance. Another important aspect to note is that more often than not, predictions are off by at most 2 classes.

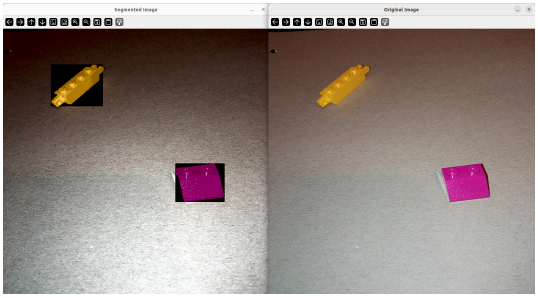


(a) Simple case

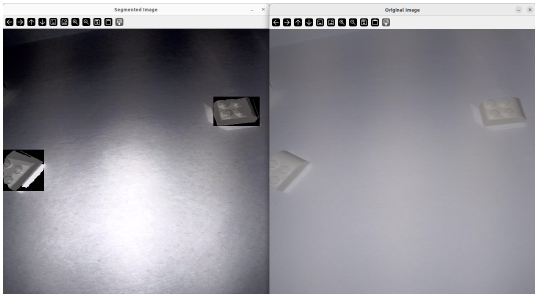


(b) Shadows presence

Figure B.7: Good results

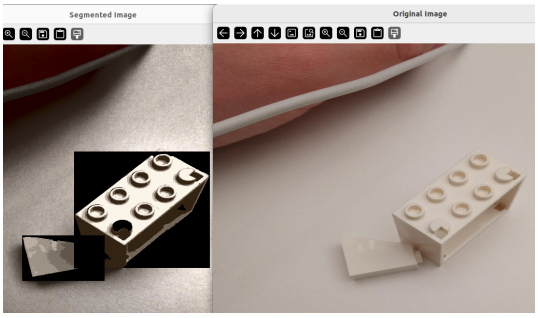


(a) Simple case

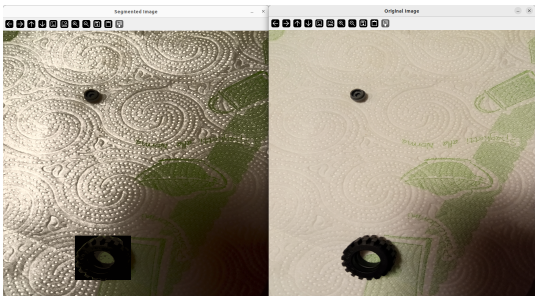


(b) Similar background

Figure B.8: Good result and semi good result

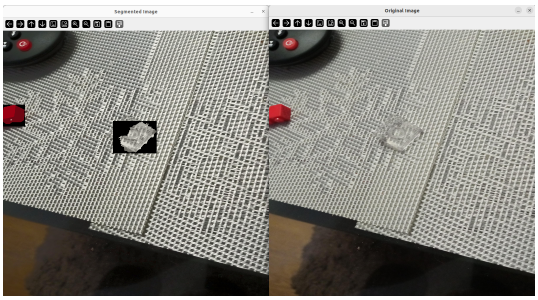


(a) Bounding boxes overlap

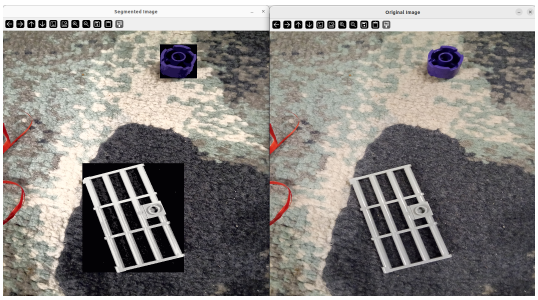


(b) Tire cavity example

Figure B.9: Bad results



(a) Transparent



(b) Similar to tire cavity example

Figure B.10: Bad results