Computer Vision - Project 2

Dataset Characterization

As the team had started the second phase of the project far before the providing of dataset split by the teachers, we developed our own based on the original full 5.6GB dataset.

In a nutshell, the team took all 5841 lego images (with 5300 being of single lego pieces which is relevant due to dataset imbalance) and rescaled them into a normalized 224x224 size.

All models used in task 2 used a 70% (training) + 30% (validation/testing) split for their fine-tuning and 80% + 20% in task 3.

Task 2

Lego Count

The purpose of this task was to create a model that could predict how many legos were in an image using a CNN. The team explored different possibilities in order to develop a capable model.

The first approach was to implement a classification model with 32 output nodes (image maximum number of legos is 32). Unfortunately, due to the imbalance of the dataset, this model only predicted that images had a single lego.

As a way to solve this problem, the team used data augmentation to increase the number of images that had more than one lego piece (annex Fig. 1) by performing flip and rotation operations. Despite this, we also decreased the number of images that contained a single lego, since its proportion was still too high.

In our final models we decided to take the images that have more than one lego, and obtained the following results from the 541 images (absolute lego count error). This can also be used to compare with the results from task 3.

```
VGG16 num errors: 44
ResNet18 num errors: 33
VGG16 average error: 0.081
ResNet18 average error: 0.061
```

The next step was to switch from using classification to using regression instead. However, the regression models didn't perform well, when compared to the classification models.

During training, we noticed that in this task, the learning rate actually had a very big impact on the results, and sometimes a minor change to it, could lead to drastic changes to the model. Even with a learning rate of 0.001 the model didn't converge most of the time, and hence, needed to be adjusted, even while using the Adam optimizer.

Finally, we increased the number of epochs done during the training of the models and satisfying results were finally achieved. In order to access our results, 3 different models were trained using both classification and regression. The first model was a simple **custom CNN**, which was mostly used as a control, and produced the worst results. The other two, were pretrained models, **VGG16** and **ResNet18**, which achieved significantly better results.

Accuracy	Classification	Regression
Custom CNN	70,4%	19,0%
VGG 16	92,0%	66,5%
Resnet 18	94,4%	55,5%

The results were quite satisfying as you can see in the table above and in the confusion matrices of each in Fig. 2 and Fig. 3 in annex.

Task 3

Lego Detection

The team explored, for the purpose of lego detection using bounding boxes, two different already pre-trained CNN models: **YOLOV8** from *Ultralytics* and **Faster R-CNN**.

The models were fine-tuned on our dataset for 5 epochs (due to time and hardware constraints) with both models requiring very particular dataset input formatting and configuration that were very challenging to achieve.

The team, from the start, saw a very glaring difference between the models, being YOLOV8 much less reliable at lego detection and being constantly spoofed by shadows. Considering only the 541 images that contain more than 1 lego and measuring the predicted lego count delta from the ground truth and accumulating those results for all those images, we get the following statistics:

```
Faster R-CNN num errors: 163
YOLO num errors: 1519
Faster R-CNN average error: 0.30129
YOLO average error: 2.80776
```

These crude metrics clearly show that Faster R-CNN is better by an order of magnitude at correctly identifying the number of legos on the image (however still far from the Lego Count specialized CNNs from task 2). Thus, this Faster R-CNN model was the selected one to then use for segmentation on each of the identified bounding boxes.

Lego Segmentation

Regarding lego segmentation, the team had already developed a traditional solution for the last project (task 1). However, the results were far from satisfactory. With that in mind, we explored many alternatives, including applying CNNs to each bounding box to extract the foreground.

The CNN approach is quite expensive in terms of both memory and computational resources and the results were very poor because we didn't have a way to train them due to lack of ground truth masks. The team experimented with pre-trained **U-Net** and **Mask R-CNN**, with the first yielding unusable masks almost 100% of the time.

The second fared a bit better (as the results in annex show), but still failed to be consistent by sometimes not detecting anything as foreground or giving badly defined lego shapes. In 30 bounding box images, Mask R-CNN only gave masks different than blank to less than a third, making it an unreliable solution.

Faced with those poor results, the team fell back to exploring traditional segmentation methods and found a much better approach based on the initial one. Firstly, we take the image within each bounding box with a padding of 2 pixels.

This padding value is very sensible because, one one side, we need padding to help the function get a more zoomed-out look at the piece and get more context of the background. But, it also may include other pieces that are very close together. The trade off is however worth it.

We also found that upscaling the bb image 5x also helped the traditional methods to have more wiggle room among pixels for the used 3x3 kernels. Next, we apply like the old method, a K-means to reduce the color space to only 3 colors in the hopes of having 2 of them being the lego color and the background (and one extra for shadows or light variations that may occur).

With this, we get very consistent color blobs as can be seen in the annex image's second column. We then apply a GrabCut and perform two final morphological transformations to connect near pixel islands and remove disconnected ones to obtain a more cohesive single blob mask (third to last column in the image). The results are far better than what we had previously achieved, even for more complicated lego shapes or challenging light conditions and yield a great segmentation for almost all situations.

Group

João Alves - up202007614 Marco André - up202004891 Rúben Monteiro - up202006478

Annex - Image Results

Task 2

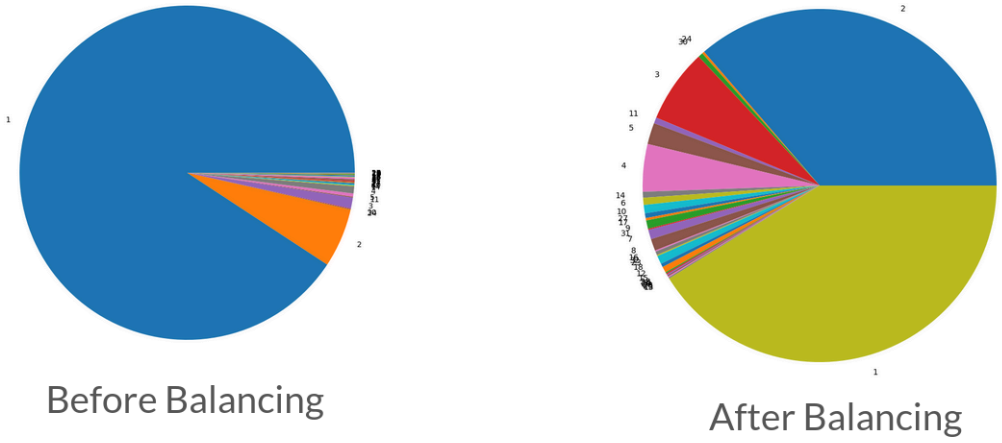


Fig. 1 - Balancing of the dataset performed in task 2

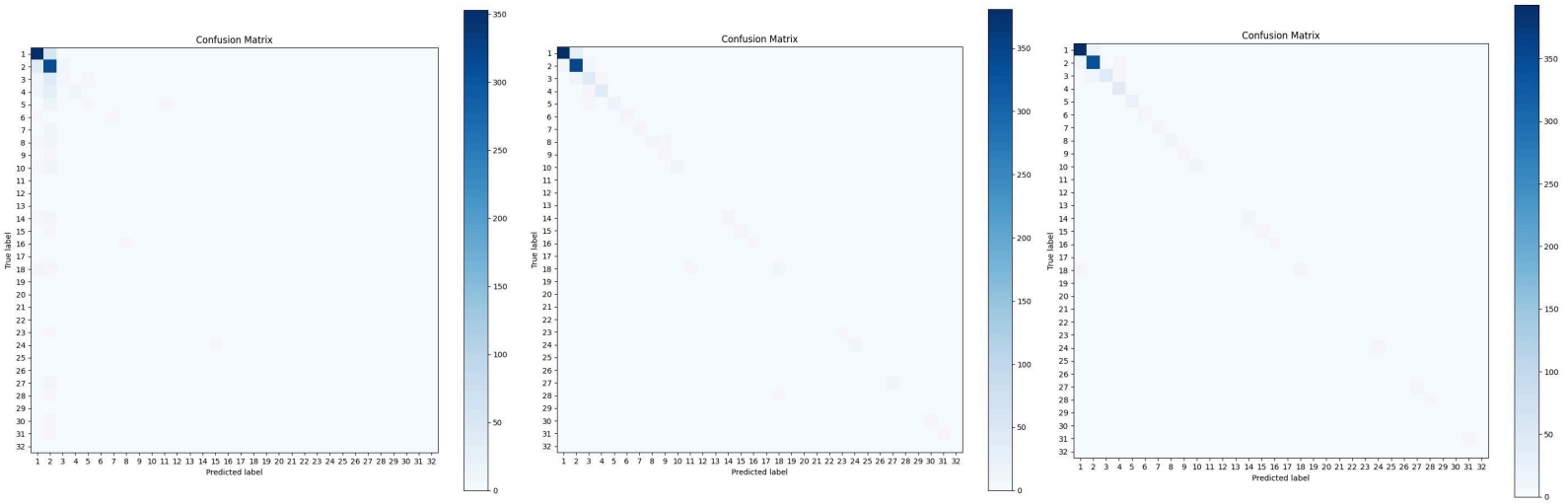


Fig. 2 - Confusion matrix for Classification (Custom CNN; VGG16; ResNet18)

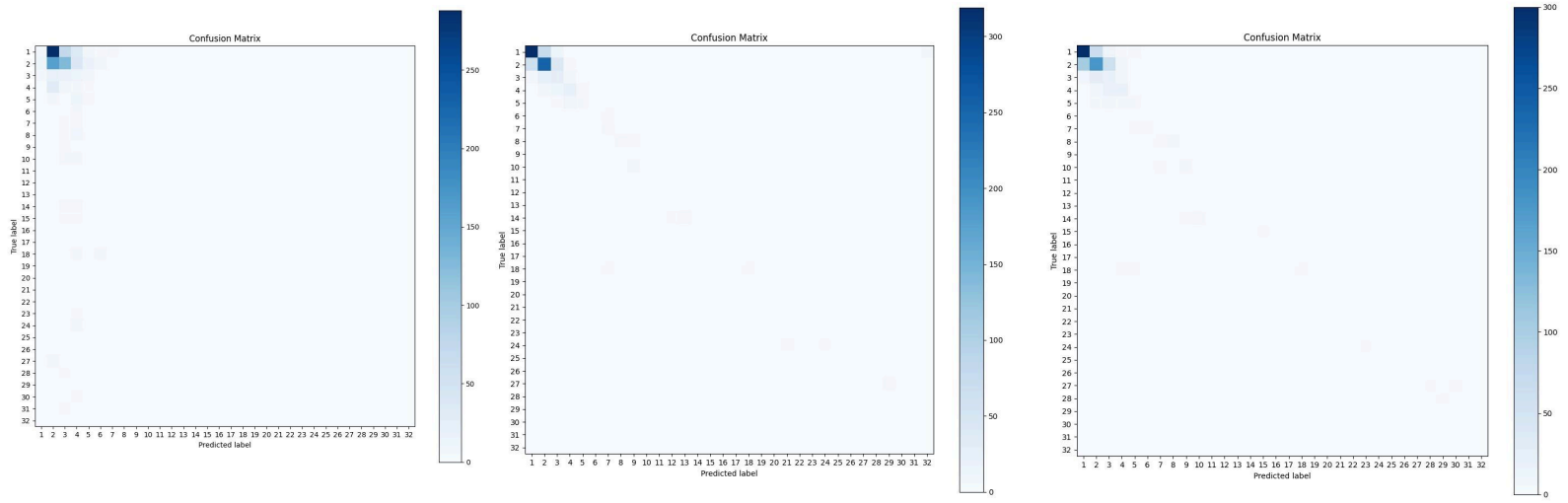
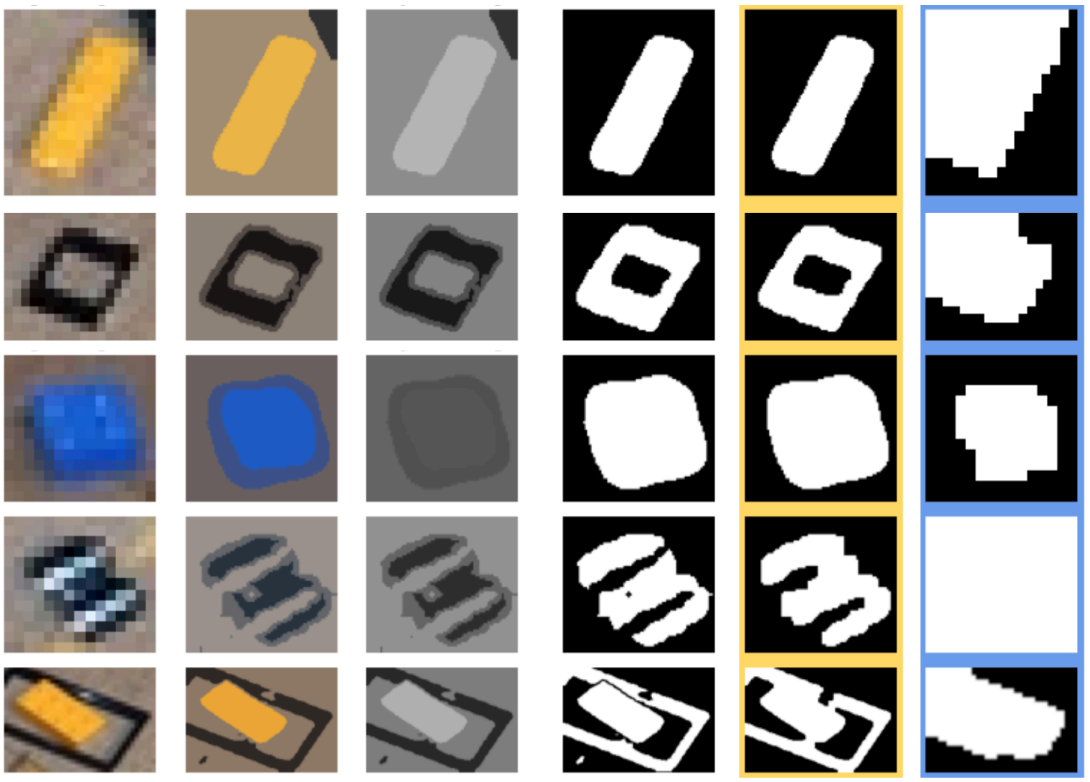


Fig. 3 - Confusion matrix for Regression (Custom CNN; VGG16; ResNet18)

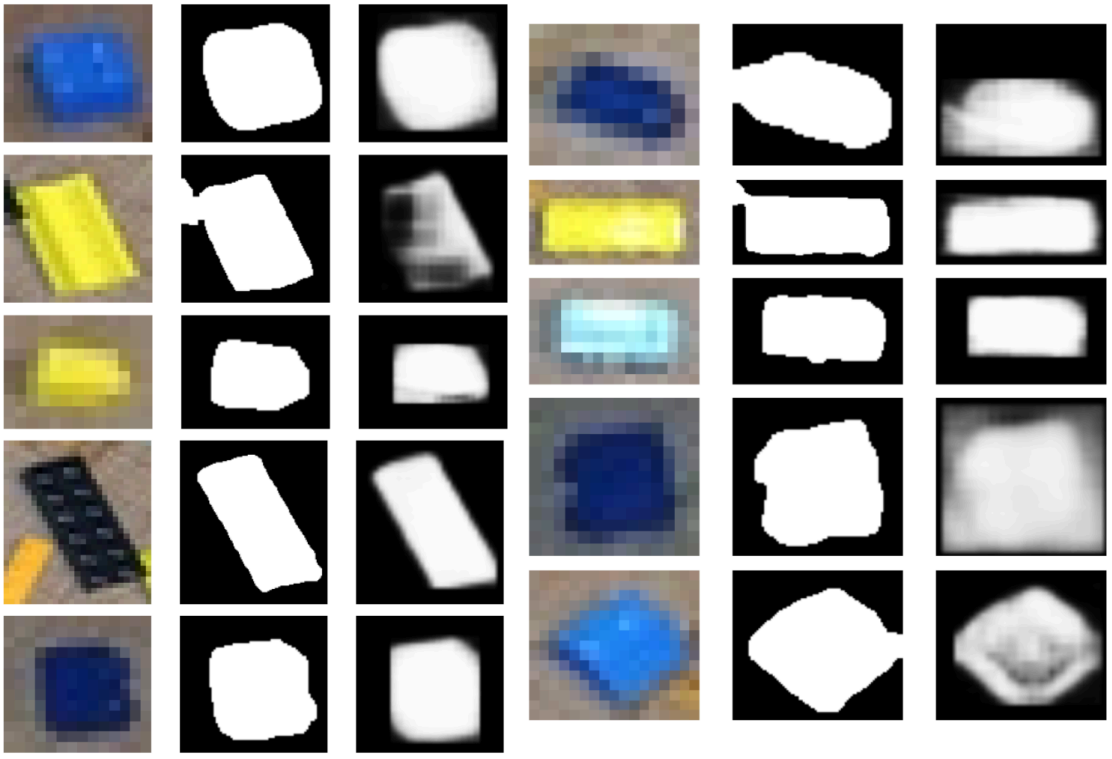
Task 3 - Single Lego Segmentation

In the image below there are all steps taken to achieve the final mask used for segmentation for multiple legos:



Yellow Bg. - new method's mask output **| Blue Bg. -** mask output from method used in task 1

Regarding the CNN segmentation, as stated in the report, U-Net almost always gave just blank masks or some small pixel blobs far from what the mask was supposed to be. The results using Mask R-CNN (third column) were a bit more tolerable but still far from our final traditional method (second column) and some of the better examples are shown below:



Task 3 - Detection + Segmentation