# Lego Bricks Classification

**Francesco Pappalardo** PAPPALARDOFG@OUTLOOK.IT
**Vanessa Perticone**[1] VANESSAPERTICONE12@GMAIL.COM

## 1. Model Description

We describe here a deep model that consists of two Convolutional Layers, to extract and recognize complex features, and by one Fully Connected Layer and a Classifier in order to perform an Image classification task on a set of Lego bricks images. Initially, input data flows trough the convolutional layers, to exploit image information. Then the extracted features are provided to the dense layer and to the classifier, which predicts the class result among one of the 16 possible classes.

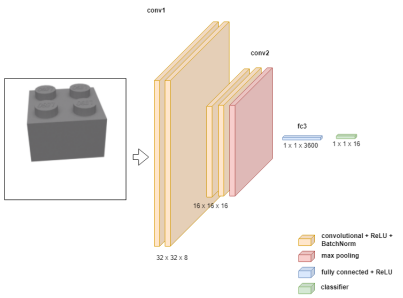


Figure 1: A simple representation of the proposed model.

At the beginning, it presents two convolutional layers, to process the input volume. The two convolutional layers have respectively as output 8, and 16 channels. The first layer has a kernel size of 3 applied with a stride of 1 in each dimension, followed by a ReLU activation function and by a layer of Batch Normalization. The second layer has a kernel size of 3 applied with a stride of 1 in each dimension, followed by a ReLU activation function and by a layer of Batch Normalization. Moreover, in this second layer, is applied also Max Pooling to reduce the size of the feature map for faster computations, keeping most important information. Features produced by these convolutional layers are then flattened and provided as an input to the Fully Connected layer. The dense layer has 1024 output features, taking as input 1568 features (the output of the convolutional layers). Finally, the classifier has an output size of 16, that corresponds to the number of the classes of the dataset. Indeed, We will use the Cross Entropy as Loss Function, so the output will be a probability vector, meaning it represents predicted probabilities of all classes, summing up to 1. This method combines the Softmax activation function principle and the negative log likelihood loss.

## 2. Dataset

LEGO is a popular brand of toy building bricks, usually sold in sets, in order to build a specific object, and each set contains a number of parts in different shapes, sizes and colors. The chosen dataset, available on the Kaggle webpage as "Images of LEGO Bricks", and contains 16 types of LEGO Bricks rendered in Blender and then rotated and exported on different angles. We apply a resize to the entire dataset, converting every image from the original size, that is 200x200, to 32x32.

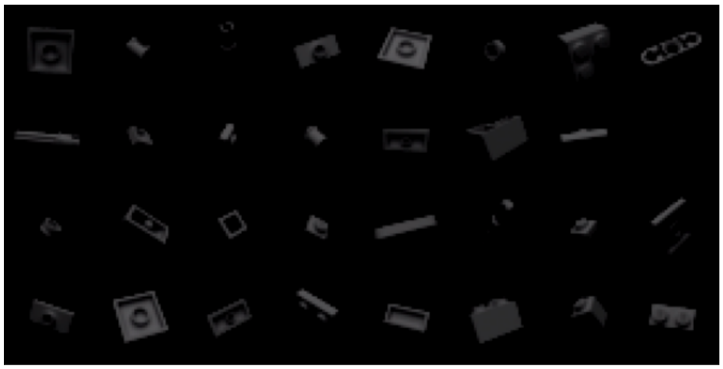

Figure 2: A sample batch from the training set.

The dataset is composed by 6379 images, divided as follow:

- 11214 Bush 3M friction with Cross axle: 400,
- 18651 Cross Axle 2M with Snap friction: 400,
- 2357 Brick corner 1x2x2: 379,
- 3003 Brick 2x2: 400,
- 3004 Brick 1x2: 400,
- 3005 Brick 1x1: 400,
- 3022 Plate 2x2: 400,
- 3023 Plate 1x2: 400,
- 3024 Plate 1x1: 400,
- 3040 Roof Tile 1x2x45deg: 400,
- 3069 Flat Tile 1x2: 400,
- 32123 half Bush: 400,
- 3673 Peg 2M: 400,

- 3713 Bush for Cross Axle: 400,
- 3794 Plate 1X2 with 1 Knob: 400,
- 6632 Technic Lever 3M: 400

Then we splitted data into 3 subsets:

- Training: 70
- Validation: 10
- Test: 20

## 3. Training procedure

No data augmentation was applied to the dataset, because of the fact that the data set is already rotated in the original dataset. We started training a deep network based on a sequence of five 2D convolutional layers. Every convolutional layer follows a ReLU activation function, a Batch Normalization and, except for the first layer. Then we tried different attempts, removing one layer for each attempt. We tried in total five models, starting from a model with 5 convolutional layers to a model with only one convolutional layer. Every model has one dense layer and a classifier. To compute the loss we apply as criterion the Cross Entropy Loss, that is the one used in classification models training:

$$-\frac{1}{n}\sum_{i=1}^{n}(y_i \log(\hat{y_i}) + (1-y_i)\log(1-\hat{y_i})) \tag{1}$$

where $y_i$ is the true output and $haty_i$ the predicted one. All the models are trained from scratch for 20 epochs on a Nvidia Tesla K80 with 12GB of VRAM. Batch size is set to 32 for all the models. Stochastic Gradient Descent is chosen as optimizer algorithm using a learning rate of 0.005.

## 4. Experimental Results

Models were trained as described in the previous section. Table 1 shows validation and test accuracy for the previously described models,

| Model | Train Accuracy | Validation Accuracy | Test Accuracy |
|---|---|---|---|
| One layer | 98.7% | 93.58% | 93.81% |
| – + Layer two | 98.9% | 93.11% | 94.44% |
| – + Layer three | 97.12% | 90.58% | 91.06% |
| – + Layer four | 97.93% | 91.23% | 90.35% |
| – + Layer five | 98.4% | 96.09% | 93.97% |
| Best model (Two layers) | 98.9% | 93.11% | 94.44% |

Table 1: Test performance of the models.

As shown in the table, the model that we described in this paper performed slighly better than the deepest ones.

The model has an high accuracy in recognizing almost all classes, except for a few ones, as visible from the confusion matrix shown below.

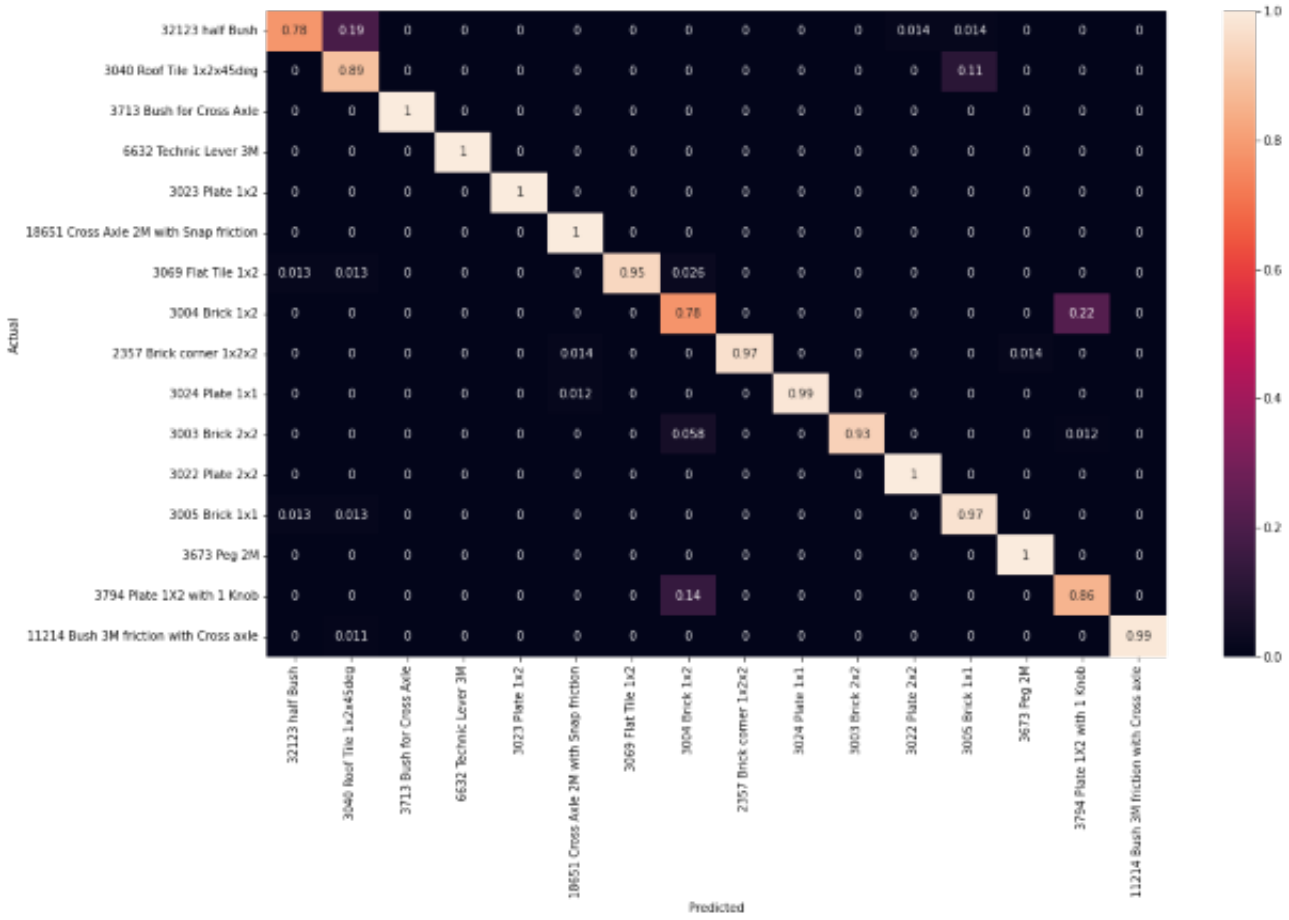


Figure 3: The confusion matrix.