Report on the project 2
Calssifying LEGO
Using Convolution Neural Networks
Nikita Gagaev

To make my project more independent of other projects I found out a Lego data set, so I will be using it.

The aim of the project:

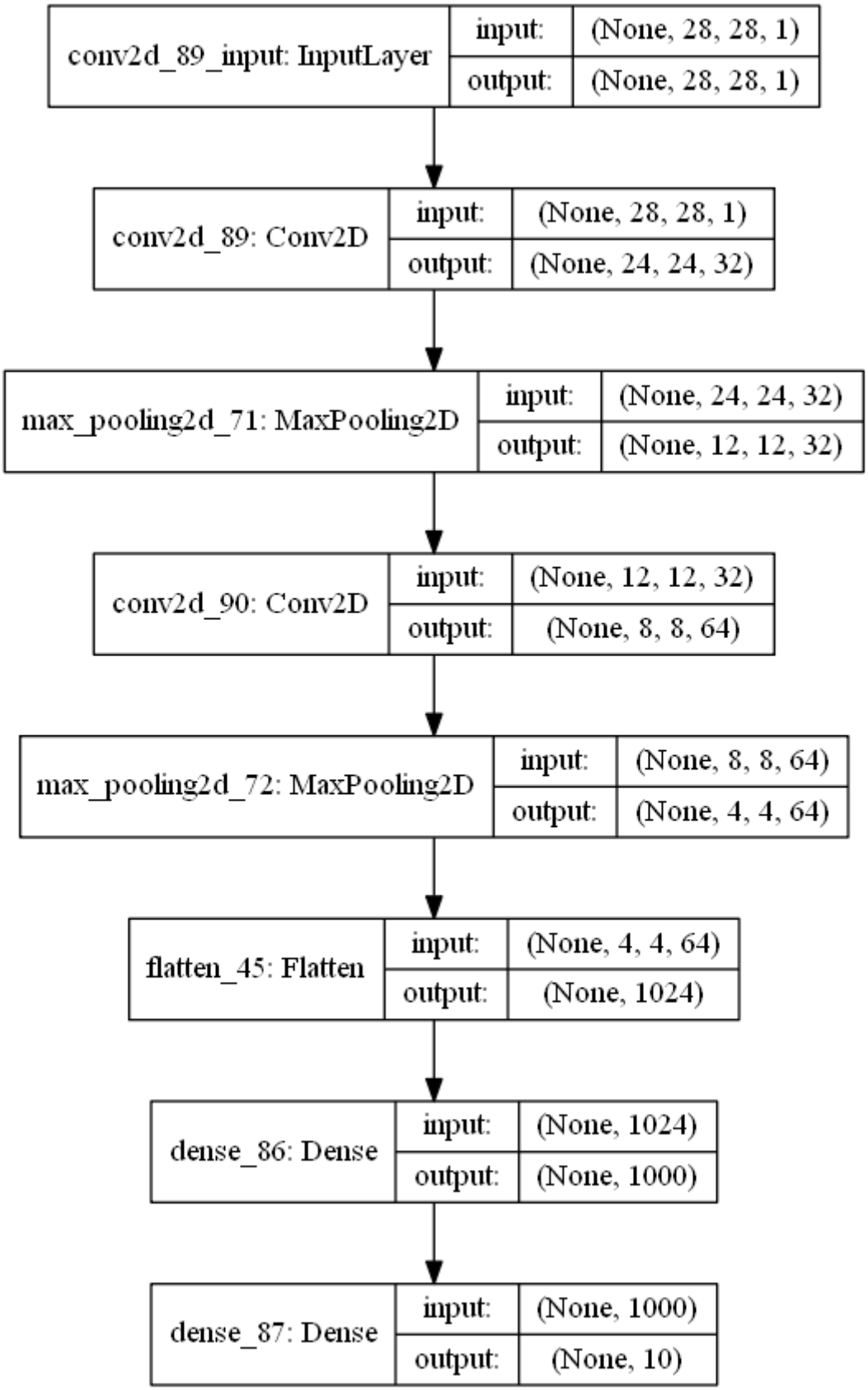
The task is to design and prototype an image processing system based on Convolution Neural Networks to detect the different shapes and classes of lego bricks and to classify them properly.

We have a set of images of Lego bricks, which should be sorted in a proper way. The goal is to identify the demension of the birck to sort it in proper way and put it in right LEGO box.

Project content:

1. '\lego-brick-images': The directory contains the training material, divided in 10 classes and contain over 3900 images, Classes are 2x2 brick, 2x1 brick, 1x1 brick, 2x2x1 birck, 1x2 plate brick, 2x2 plate brick, 1x1 plate brick, 1x2 roof brick, 1x2 flat brick, 1x2 brick with one knob.
2. 'CNN.py': This keras based deep-learning algorithm which trains the Convolutional Neural Networks to classify the bircks and our main program.
3. 'model.h5': The file contains weights that have been saved after the training. File is produced by the algorithm and can be used again to use old weights and update them.
4. ‘prediction_comparison.txt’: The comparison of actual data classification and the predictions made by algorithm.
5. 'Loss and accuracy graphs': that shows the loss and accuracy.
6. 'Confusion_matrix': that compares the prediction and real class.
7. 'Data per epoch': that shows updates of every epoch.

8. 'model.png': The graphical representation of network layers structure:



For the input we have 2-dimentional array and as the output we get a 1-dimentional array of 10 elements (the same as the number of classes).

Neural network itself:

First, we load the training and testing data from '\lego-brick-images\test' and '\lego-brick-images\train' directories. The number of training data is 3984 and the number of test data is 618 we set color mode to grayscale.

Then using the iterators we create the training and testing numpy image arrays and label arrays.

Then we have 2D convolution which is computed in a similar way one would calculate 1D convolution: you slide your kernel over the input, calculate the element-wise multiplications and sum them up. But instead of your kernel/input being an array, here they are matrices. On convolution 2D we use Relu activation function.

Then we have max pooling 2D which is a sample-based discretization process. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned. For each of the regions represented by the filter, we will take the max of that region and create a new, output matrix where each element is the max of a region in the original input.

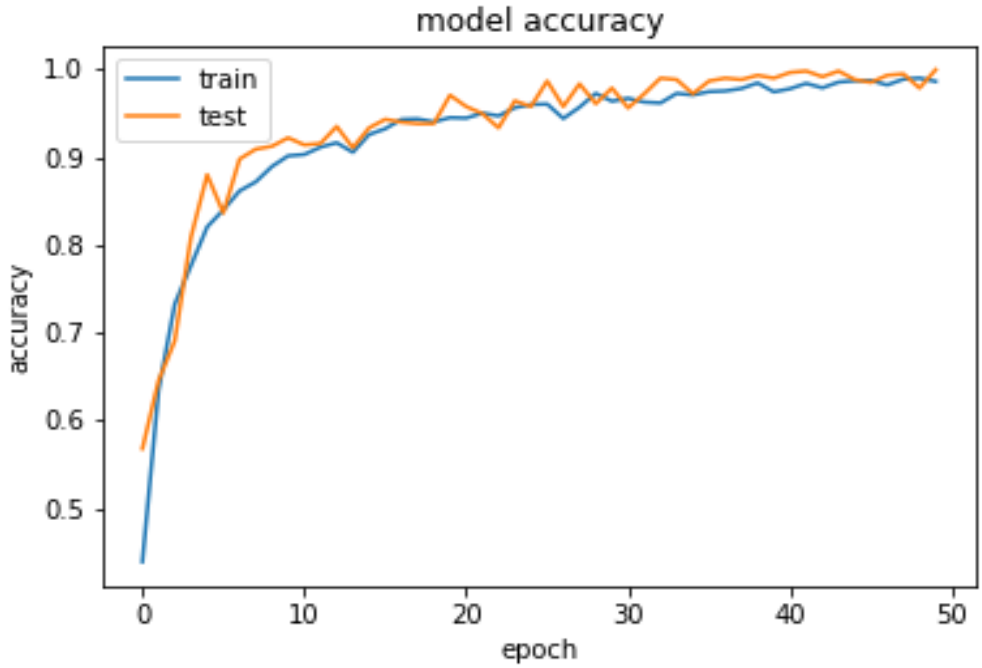
Then we repeat the procedure with convolution and pooling, and Flatten the image. The flatten() function is used to get a copy of an given array collapsed into one dimension.

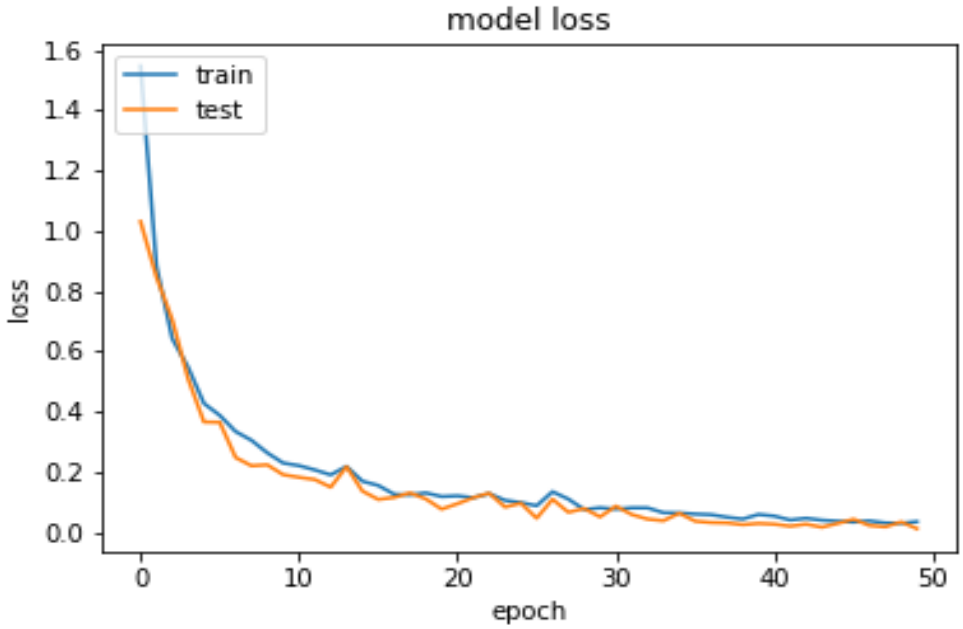
After that we set our training model by having 2 layers: 1x Relu and 1 softmax. We can see it in the 'model.png' picture.

Then we train the model using training data and labels, and evaluated the obtained model (with the obtained weights) using the testing data and labels.

We print the actual classification of testing data and the predicted classification of testing data. Usually the accuracy is close to 0.9.

After that we print the accuracy and the loss plots.
When we compile our algorithm on 50 epochs we can see something like this:

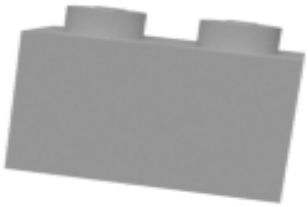




For obvious reasons our neural network is confused as we are! So sometimes the accuracy never can reach 1. And there is a reason for this:

This is 2x2 brick:

And this is 1x1 brick:



On the table (confusion matrix) below we can see the comparison of prediction model to real labels.

		Prediction									
Real value		0	1	2	3	4	5	6	7	8	9
	0	30	0	0	0	0	0	0	0	0	0
	1	0	48	0	0	0	0	0	0	0	0
	2	0	0	82	0	0	0	0	0	0	0
	3	0	0	0	41	0	0	0	0	0	0
	4	0	0	0	0	34	0	0	0	0	0
	5	0	0	0	0	0	34	0	0	1	0
	6	0	0	0	0	0	0	90	0	0	0
	7	0	0	0	0	0	0	0	82	0	0
	8	0	0	0	0	0	0	0	0	137	0
	9	0	0	0	0	0	0	0	0	0	39

Conclusion:

To classify and sort lego bricks convolution neural network that classifies based on the pictures is extreamly good option even better that simple neural network, because it learns faster and probably it's sophisticated but good way to solve some problems especially with images. The more epoch we have the more accuracy we obtain, but we should avoid some images that are look like one another. But this can be handled with large masives of data, the lack of data is important we can not solve it by just incremention iterations. Even after 500 epochs the accuracy is around 1, but not 1 for every epoch. So we don't have enough data to train it more, so sometimes mistakes can occure.