Post-print of: Boiński, T., Zawora, K., Szymański, J. (2022). How to Sort Them? A Network for LEGO Bricks Classification. In: Groen, D., de Mulatier, C., Paszynski, M., Krzhizhanovskaya, V.V., Dongarra, J.J., Sloot, P.M.A. (eds) Computational Science – ICCS 2022. ICCS 2022. Lecture Notes in Computer Science, vol 13352. Springer, Cham. https://doi.org/10.1007/978-3-031-08757-8_52

How to sort them? A network for LEGO bricks classification

Tomasz Boiński[1][0000−0001−5928−5782], Konrad Zawora[1]
, and Julian Szymański[1][2222−−3333−4444−5555]

Faculty of Electronics, Telecommunication and Informatics,
Gdańsk University of Technology,
11/12 Narutowicza Street, 80-233 Gdańsk, Poland
{tomboins}{julszyma}@pg.edu.pl

Abstract. LEGO bricks are highly popular due to the ability to build almost any type of creation. This is possible thanks to availability of multiple shapes and colors of the bricks. For the smooth build process the bricks need to properly sorted and arranged. In our work we aim at creating an automated LEGO bricks sorter. With over 3700 different LEGO parts bricks classification has to be done with deep neural networks. The question arises which model of the available should we use? In this paper we try to answer this question. The paper presents a comparison of 28 models used for image classification trained to classify objects to high number of classes with potentially high level of similarity. For that purpose a dataset consisting of 447 classes was prepared. The paper presents brief description of analyzed models, the training and comparison process and discusses the results obtained. Finally the paper proposes an answer what network architecture should be used for the problem of LEGO bricks classification and other similar problems.

Keywords: Image classification · LEGO · Neural Networks.

1 Introduction

LEGO bricks are highly popular among kids and adults. They can be used to build vast array of, both very simple and very complex, constructions. This is achieved by availability of multiple, sometimes very different, yet compatible brick shapes. For the smooth build process the bricks need to properly sorted and arranged - constant searching for proper bricks in a big pile of LEGO is discouraging and limits creativity. Usually the sorting is done by shape. The colors and decals can be easily distinguished even in a big pail of bricks [2]. Still, with over 3700 different LEGO parts [24] (and the number is constantly growing) even disregarding the color makes the problem complex.

No solution for this problem was proposed so far. LEGO Group provides only a simple sorting mechanism, based on the brick size, in form of the 2011 released, now discontinued, LEGO Sort and Store item. Fan offered solutions usually rely on optimization of the manual sorting process (e.g. [1]). Some fans tried to build

AI powered sorting machines [10,38] with some success. Independently from the way of building the sorting machine, it requires a well-trained neural network able to distinguish between different, often very similar bricks. The solution should at least divide them into smaller number of categories aggregating bricks similar in shape and usage, allowing further manual selection of proper bricks. Thus LEGO oriented object classification solution is needed.

Problems like object detection, image segmentation, content-based image retrieval, or most commonly, object classification lie in domain of computer vision. In the last case the given, previously detected object, is assigned a one or more labels. The objects can have either one label assigned (multi-class classification) or many labels assigned (multi-label classification).

Computer vision is an actively research sub-domain of machine learning. It originated as far as in late 60ties of the 20-th century [27]. What was at the beginning portrayed as a simple task, assigned to students in summer school, currently remains a complex and not yet fully solved problem.

Across the recent years multiple deep neural network architectures emerged. For their comparison a standardised approach was established - *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) competition [29]. During the competition the models should classify objects to one of the 1000 classes based on 1.2 million of training images. The model accuracy is tested on 150000 images. Two metrics are calculated – Top1 (the percentage of directly correctly classified images) and Top5 (the percentage of images that were classified among the 5 with the highest probability). There are other commonly used datasets like CIFAR-10 and CIFAR-100 [20], SIFT10M [8], Open Images Dataset [19,21], Microsoft Common Objects in Context (COCO) [23]. As each dataset contains photos from different categories, with different size etc., good standing with one of the datasets does not guarantee the same results with the other. Furthermore the datasets try to be very general whereas in some cases the images contain similar objects. That is why further evaluation is still required.

In our research we undertook construction of AI-powered sorting machine [6] treating LEGO recognition as multi-class classification. To search for the best architecture that matches our scenario we decided to base our dataset in that prepared for ILSVRC. This way we could speed up training process thanks to *transfer learning* approach. As candidate architectures we selected the ones that achieved the best results in the aforementioned competition.

The structure of this paper is as follows. In Section 2 a description of compared network topologies is given. Later on, in Section 3, the used dataset is presented. Further in Section 4 details how the training was done and the testing methodology are presented. Section 5 discusses results obtained during the tests. Finally, some conclusions are given.

2 Network topologies

In this paper we tested 28 network topologies from 7 families:

MOST WIEDZY Downloaded from mostwiedzy.pl

- EfficientNet – EfficientNetB0, EfficientNetB1, EfficientNetB2, EfficientNetB3, EfficientNetB4, EfficientNetB5, EfficientNetB6 and EfficientNetB7 variants,
- NASNet – NASNetMobile and NASNetLarge variants,
- ResNet – ResNet50, ResNet50V2, ResNet101, ResNet101V2, ResNet152 and ResNet152V2 variants,
- MobileNet – MobileNet, MobileNetV2, MobileNetV3Large and MobileNetV3-Small variants,
- Inception – InceptionV3, InceptionResNetV2 and Xception variants,
- DenseNet – DenseNet121, DenseNet169 and DenseNet201 variants,
- VGG – VGG16 and VGG19 variants.

In this section a brief introduction to each family and variant is given, portraying its strengths and rationale behind the used architecture.

EfficientNet architecture was defined in 2019 [37]. The model aims at efficient scaling of convolutional deep neural networks. The authors distinguished three dimensions of scaling: depth scaling, width scaling and resolution scaling. Depth scaling is the most commonly used approach, as it allows increase in number and complexity of detected features by increasing the number of convolutions. However, with increasing network depth, the training process gets longer and a problem of vanishing gradient can be observed [13]. Width scaling relies on increase of number of channels in each convolution. It is commonly used in shallow networks, where width scaling increased both training speed and classification quality [39]. Resolution scaling allows potential extraction of additional features. With all three scaling approaches there is a point of diminishing returns, beyond which additional computational overhead is not being compensated by better accuracy. EfficientNet uses so-called compound scaling, where all three parameters are equally scaled using ϕ parameter.

The base model here is similar to MnasNet [36] and MobileNetV2 [30]. Each model in this family differs by the ϕ parameter value (starting with $\phi = 0$).

Care needs to be taken when using the model in TensorFlow framework [31], as zero-padding is used for convolutions with resolutions that cannot be divided by 8. The number of channels also needs to be divisible by 8. The real compound scaling parameters applied when using TensorFlow are thus different.

ResNet50 was proposed in 2015 [11], as a solution to vanishing and exploding gradient problems. Thanks to so-called residual connections, it allows training of very deep networks (over 1000 convolutional layers). Residual connections perform elementwise addition of identity function between convolution blocks. This improves gradient flow, by skipping non-linear activation functions usually placed in convolutional blocks.

In 2016 a revision of the original model was proposed (called ResNet V2) [12]. The whole family of this model (in both ResNet and ResNet V2 revisions) achieves very high results in ILSVR competition reaching 74.9%-78% accuracy in Top1 and 92.1%-94.2% accuracy in Top5 categories.

DenseNet was defined in 2016 [16]. Similarly as in ResNet, the aim is to solve the vanishing gradient by shortening its flow path. DenseNet uses so-called *dense blocks* to achieve it. The dense block consists of 1x1 and 3x3 blocks and output

MOST WIEDZY Downloaded from mostwiedzy.pl

of every block within it is connected with input of every next block. Each layer within a dense block has thus direct access to its output which limits the flow path. DenseNet has also low width of the convolutional layers. Each variant of DenseNet architecture differs in terms of size of the last two dense blocks.

In 2018 DenseNet achieved the highest score in ILSVR competition Top1 category reaching accuracy of 75% for DenseNet-121, 76.2% for DenseNet-169, 77.42% for DenseNet-201 and 77.85% for DenseNet-264 77.85%.

Inception architecture was defined in 2014 [34] with Inception v1/GoogLeNet. The aim was to reduce the risk of overfitting and eliminate the problems with gradient flow. A special Inception block was proposed - it is composed of three layers with different filters (1x1, 3x3 and 5x5). This led to high calculation complexity so a reduction was introduced that limited the number of entry channels. 9 Inception v1 blocks were combined as GoogLeNet architecture.

Inception v2 and v3 were defined in 2015 [35]. They increased performance, limiting information loss and computational complexity. Inception v3 achieves 77.9% accuracy in ILSVR competition Top1 category and 93.7% in Top5.

In 2016 Inception v4, InceptionResNetV1 and InceptionResNetV2 architectures were proposed [33]. The main goal was simplification and unification of the Inception models. ResNet residual connections were also included in the model. The best results were obtained by InceptionResNetV2 model. In Top1 category of the ILSVRC competition it achieved accuracy of 80.3% and in Top5 95.3%.

In 2017 an extension to Inception V3, by replacing the inception block with so-called extreme inception, was defined [7]. The original block was modified so that for each 1x1 convolution output corresponds one 3x3 convolution. This architecture, called Xception, proved to be easier to define and modify in software frameworks than the original Inception model.

Xception achieved better results in ILSVR competition than the original Inception v3 model. For Top1 category the accuracy was 79% and for Top5 94.5%. It also had less parameters (22.86 million vs 23.63 million).

NASNet model was defined in 2017 [41]. It was created thanks to Google AI's *AutoML* [28] and Neural Architecture Search [40]. The creation of optimal network architecture is treated here as reinforcement learning problem, with the final network accuracy as a reward. This induced a very high computational cost, so the search space had to be narrowed considerably. Based on the analysis of other models the authors first defined a general architecture, which composed of only 2 blocks - *normal cell* and *reduction cell*.

This significantly reduced the time needed to find the optimal model. Still, the training time remained very long. However, the model achieved good results. For ILSVRC Top 1 category it reached accuracy of 74.4% and 82.5% for smaller NASNetMobile variant, and larger NASNetLarge variant respectively.

MobileNet model was defined in 2017 [15]. It was designed to allow fast inference on mobile and embedded devices. The authors of this solution point out that after a certain level of network complexity, the increase in inference time is much bigger than the increase in accuracy, making the potential gain computationally unprofitable. To further increase the performance of inference,

MOST WIEDZY Downloaded from mostwiedzy.pl

authors defined a special convolution, called *depthwise separable convolution*. It separates the operation into two phases - filtering and combination. This approach allowed up to 9 times lower computational complexity with only a 1% lower accuracy [15] (for ILSVRC Top 1 category).

Few versions of MobileNet architecture were proposed, each introducing usage of different approaches (like residual connections) or different numbers of channels. The original MobileNet model achieved for Top1 category of ILSVRC competition the accuracy equal to 70.6%. MobileNetV2 [30] achieved 72.0% with around 20% lower number of parameters and 47% lower computational cost. MobileNetV3 [14] introduced 2 versions - Small with 2.5 million parameters and Large with 5.4 million parameters. The accuracy for Top 1 category of ILSVRC competition was 75.2% for MobileNetV3Large and 67.4% for MobileNetV3Small.

VGG is one of the oldest architectures, was defined in 2014 [32]. Different variants of this model vary by the number of trainable layers. For Top 1 category of ILSVRC competition, VGG16 and VGG19 reach accuracy of 71.3%. For Top5 category, VGG16 reaches accuracy of 90.1%, whereas VGG19 of 90%.

As we can see, all of the aforementioned models achieved very good result in the ILSVRC competition. At the time of their publication they gained the highest score and usually became the state of the art. As mentioned in Section 1 it doesn't always translate to the same results for other datasets.

3 The dataset

During the training we used custom dataset containing both real photos and renders of LEGO bricks, belonging to 447 classes. The bricks were taken from authors personal collection of over 150 LEGO sets and represents the most commonly available brick shapes. The whole dataset consists of 620082 images, where 52601 were real photos and 567481 were life-like renders. The renders were created using Blender tool [9] based on 3D models from LDraw library [17].

The renders were used to speed up data gathering. We created a script that randomly selected a brick type, color and alignment simulating its move on a conveyor belt below a fixed positioned camera. Thanks to Blender and its extension called ImportLDraw [26] we managed to generate realistic images of LEGO bricks. Sample renders, after being cropped, can be seen in Fig. 1.

Real photos were created to increase the representativeness of the training set. For that we created a dedicated Android app allowing quick tagging and automatic cropping of LEGO bricks on pictures taken with phone camera. Sample real photos can be seen in Figure 2.

The full set of rendered images (before cropping) and real photos are publicly available – [5] and [3] respectively. The complete dataset is also available [4].

Before the training the dataset was prepared so that all networks would be trained on the same images. The images need to be standardised in terms of size and proportions. As some of the bricks are long and narrow (e.g. brick 3002), we decided to scale the longer edge to the desired size, and the shorter edge proportionally (otherwise we could loose some information). Then, the image

MOST WIEDZY Downloaded from mostwiedzy.pl

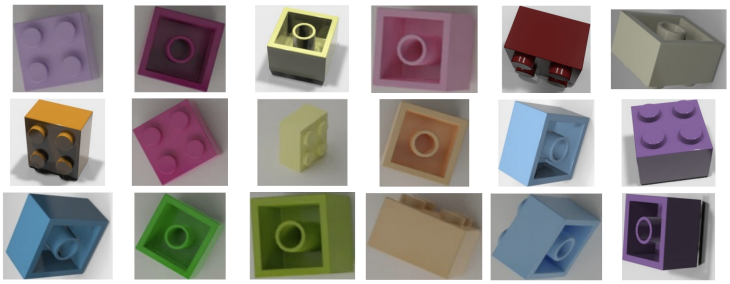

Fig. 1. Sample renders for brick number 3003

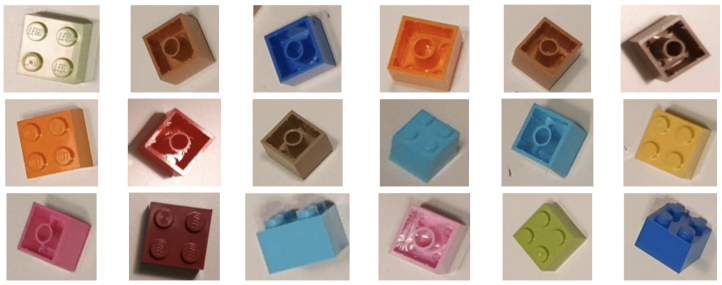

Fig. 2. Sample real photos of brick number 3003

canvas was extended to form a square and was filled with white background. Next, all images were augmented using imgaug library [18]. The transformation included the following operations applied with 50% probability:

- scaling to randomly selected size (80%-120% of the original size),
- random rotation between $-45°$ and $45°$,
- random shift by up to 20%,
- random transformation into a trapezoid with an angle of up to 16°.

Next, 5 randomly selected operations were applied, from the following list:

- Gaussian, median or averaged blur with a random intensity,
- sharpening filter with random blending factor and brightness,
- emboss filter with random blend factor and brightness,
- superimpose the contours detected by the edge detection filter, with a probability of 50%,
- Gaussian noise of random intensity,
- dropout of random pixels or a group of pixels,
- inversion of every image channel, with probability of 5%
- addition of a random value to each pixel,
- random brightness change of the image,
- random contrast change of the image,

MOST WIEDZY Downloaded from mostwiedzy.pl

– generation of a grayscale image and overlaying it with random transparency over the original photo.

The augmentations were done once, so that the results will be comparable. During the training process, to reduce the risk of overfitting, we performed additional augmentation before each epoch - the images were rotated by random angle up to 15° and the contrast was changed by random value (up to 10%).

The data gathered were divided into training and validation sets. The training set contains 447000 images (1000 images each class, 650 renders and 350 real photos). The validation set contained 44700 images (100 images each class, 50 renders, 50 real photos). The numbers were obtained experimentally.

The test set consisted of real photos created independently. It contains 4000 images of bricks belonging to 20 classes (200 images each). The set was created in separate session using bricks from other set (Lego Creative Box Classic – 10698). We used 2 variants of the set - easy and hard. Both have the same number of photos, however the hard set contained images that are hard or even impossible to distinguish but belongs to different classes (e.g. bricks 3001 and 3010).

4 Training process

All models presented in Section 2 were trained using transfer learning approach. It consisted of 2 phases:

– pre-training – done with the base model locked, only the newly added top layers are trained,
– fine tuning – the base model was partially or completely unlocked, all unlocked layers could be trained.

Pre-training is characterised by a high learning rate (we've used 0.01) with relatively low computation cost, as the backward pass needs to be calculated only for the newly added layers. After this stage, we could observe Top1 accuracy for the 447 LEGO classes at around 50-70%. During the fine-tuning stage, some of the layers are unlocked and the training is repeated for those layers. The problem here is how many layers should be unlocked. If the number will be low, then the training process will be faster, but we might not get to the desired accuracy. The number of unlocked layers also depends on the initial size of the base model.

We aimed at comparing different architectures so we designed adaptive fine tuning algorithm. It goes as follows:

1. `N := 0`
2. `N := N + unfreeze_interval`
3. `top1_history := []`
4. Unlock `N` top layers and recompile the model
5. Perform 1 training epoch
6. Perform 1 validation epoch
7. Add the Top1 accuracy on the validation set to `top1_history` list

MOST WIEDZY Downloaded from mostwiedzy.pl

8. If `top1_history` contains no less than `patience` elements and the Top1 accuracy on validation set did not increase by at least `min_delta` during last `patience` epochs, go to step 2
9. If `top1_history` contains `max_epochs_per_fit` elements, go to step 2.
10. Go to step 5.

where:

- `unfreeze_interval` (15 by default) – the number of layers to be unlocked within on fine-tuning iteration,
- `max_epochs_per_fit` (50 by default) – max number of training epochs in one fine-tuning iteration,
- `patience` (5 by default) – the number of epochs in one iteration, after which the model quality is evaluated,
- `min_delta` (0.01 by default) – minimal requested Top1 accuracy improvement reached in `patience` epochs .

The aforementioned algorithm was run to train each model for limited time. To increase the training speed and limit memory footprint we used so called *mixed precision training* [25] and XLA [22] compiler. Both approaches allowed us to train the networks with larger batch sizes.

For fine tuning we've used `learning_rate` = 0.0001. Both phases were trained using categorical cross-entropy loss function and Adam optimizer. All networks were trained with `batch_size` = 128, except for EfficientNetB5, EfficientNetB6 and EfficientNetB7, which used 64, 32 and 32 respectively.

5 The results

In total 28 network topologies were tested. The comparison process was divided into two stages. First, all models were trained for four hours using adaptive fine tuning approach described in Section 4. The second stage lasted twelve hours. It was done with the same approach as stage 1, but only 5 best models and the best out of each family was trained. All tests were done on dual Intel Xeon Gold 6130 server with 256 GiB RAM and dual NVIDIA GeForce RTX 2080 (8 GiB GDDR6 RAM each) GPU cards. Each training was done on single GPU (two models were trained at once). The default batch size was 128. Due to the memory constraints some models used smaller batch size, namely: EfficientNetB5 (64), EfficientNetB6 (32) and EfficientNetB7 (32). In both stages we used transfer learning, where for the first stage we used a model trained on ImageNet data.

5.1 Stage I - the four-hour training

Summary of obtained results (ranked from best to worst) are presented in Table 1. The best model in each family is marked with bold font.

EfficientNet models achieved varied results. The best variant was EfficientNetB1. It reached 84.4% Top1 accuracy and 95.85% Top5 accuracy, giving it the 8th place. EfficientNetB3 and EfficientNetB0 got slightly worse results, whereas

MOST WIEDZY Downloaded from mostwiedzy.pl

Table 1. Results after the first stage (measured on the validation set)

Model	Top1 accuracy	Top5 accuracy	Epochs run	Training time
VGG16	**92.99%**	**99.00%**	**11**	**04:01:07**
VGG19	91.70%	98.64%	11	04:05:07
ResNet50	**87.40%**	**96.96%**	**14**	**04:04:07**
ResNet101V2	87.20%	96.94%	14	04:10:42
ResNet152V2	86.92%	96.74%	15	04:11:10
ResNet50V2	86.57%	96.72%	14	04:05:15
ResNet152	86.00%	96.28%	13	04:09:27
EfficientNetB1	**84.40%**	**95.85%**	**15**	**04:12:06**
ResNet101	84.09%	95.44%	14	04:16:57
EfficientNetB3	83.58%	95.63%	10	04:14:06
EfficientNetB0	82.87%	95.03%	15	04:15:20
MobileNetV3Large	**82.32%**	**94.80%**	**17**	**04:09:28**
Xception	**82.31%**	**94.95%**	**9**	**04:27:31**
EfficientNetB2	81.33%	94.50%	11	04:04:16
InceptionResNetV2	79.43%	93.90%	7	04:01:10
MobileNet	78.56%	94.04%	14	04:09:26
DenseNet201	**77.41%**	**92.18%**	**14**	**04:01:57**
MobileNetV2	75.75%	92.29%	16	04:06:31
DenseNet169	75.44%	91.38%	13	04:14:47
DenseNet121	74.01%	90.49%	14	04:06:56
MobileNetV3Small	73.07%	89.97%	17	04:13:45
InceptionV3	71.19%	89.22%	10	04:16:48
EfficientNetB5	66.72%	87.58%	3	04:14:28
EfficientNetB4	65.60%	86.67%	6	04:25:51
NASNetMobile	**59.60%**	**82.08%**	**16**	**04:11:54**
EfficientNetB6	58.34%	82.15%	2	04:35:13
EfficientNetB7	54.38%	78.60%	1	04:03:32
NASNetLarge	53.39%	77.89%	4	04:26:13

EfficientNetB7 and EfficientNetB6 were one of the worst models. The reason for such outcome was the compound scaling which caused small number of frames (images) processed in the give time frame. This led to relatively small number of epochs and thus lower accuracy. The differences between EfficientNet variant are sustainable. For the next stage only EfficientNetB1 variant was selected.

ResNet models achieved very good results. The best variant was ResNet50 reaching 87.40% Top1 and 96.96% Top5 accuracy. The other variants achieved similar results. 3 models were selected: ResNet50, ResNet101V2 and ResNet152V2.

Inception models reached mediocre results. The best one was Xception reaching 82.31% Top1 and 94.95% Top5 accuracy. All models finished pre-training stage and reached the fine-tuning phase. However, we observed very low performance in terms of processed images per second, which might have been the cause of mediocre accuracy. Thus only the Xception model was selected.

MOST WIEDZY Downloaded from mostwiedzy.pl

DenseNet models did not perform too well. The best results were obtained by DenseNet201 variant (77.41% Top1 and 92.18% Top5 accuracy). Contrary to other models, the poor quality did not come from performance problems. DenseNet training showed one of the highest images per second rate. The problem lies in low increase of accuracy between epochs. We suspect it is caused by the design of DenseNet, specifically the concatenation operation. Unlike other tested architectures, in DenseNet, last convolutional layers are just a small part of the final feature map. By tuning a small amount of top convolutional layers, we're potentially leaving a big part of the feature map intact. This could be fixed by changing the training methodology and training all convolutional layers, but it has not been attempted in this phase.

NASNet models also got poor results. The best one, NASNetMobile, reached 59.60% Top1 and 82.08% Top5 accuracy placing 25 out of 28 tested models. Once again performance was the reason for the results. For the second stage only NASNetMobile was selected.

MobileNet scored averagely, the best variant being MobileNetV3Large reaching 82.32% Top1 and 94.8% Top5 accuracy. This variant, despite being targeted for mobile devices, outperforms deeper models like Xception or DenseNet201 thanks to the highest images per second rate and thus the highest training performance. For the second stage MobileNetV3Large was selected.

The best results were obtained by the VGG network variants - VGG16 placed first (with 92.99% Top1 and 99% Top5 accuracy) and VGG19 placed second (with 91.70% Top1 and 98.64% Top5 accuracy). The results came unexpected, as this is the oldest tested architecture. During the ILSVRC competition it was outperformed over the years by all other tested models, with exception of some MobileNet variants. The VGG are relatively shallow, but very wide. This allows fast unlocking of many layers in the fine-tuning approach and thus leads to very fast learning times. For the second stage both VGG16 and VGG19 were selected.

5.2 Stage II - the twelve-hour training

During this stage 10 models were further trained. The aggregated results (ranked from best to worst) can be seen in Table 2.

All models managed to get better results. In most cases (except NASNetMobile and DenseNet201) twelve-hour limit was sufficient to achieve convergence.

During this stage, we observed the similar results as in the previous one. Once again, VGG16 and ResNet50 proved to be the best. However, the quality difference, both in Top1 and Top5 accuracy, between models that reached convergence is not big - the biggest difference is only 2.18 percentage points. This is true even for mobile models, like MobileNetV3Large. This network required however more epochs to reach convergence.

What came as a surprise is that, once again, VGG16 model achieved the best results. In ILSVRC competition this model is outperformed by every other non-mobile approach presented in this paper. In the problem presented here (distinguishing LEGO bricks), VGG16 model trains very fast and reaches superb accuracy. This model is, however, characterised by high number of parameters

MOST WIEDZY Downloaded from mostwiedzy.pl

Table 2. Result after the second stage (measured on the validation set)

Model	Top1 accuracy	Top5 accuracy	Epochs run	Training time	Parameters
VGG16	94.56%	99.21%	31	12:13:49	138.3M
ResNet50	93.81%	99.10%	41	12:13:23	25.6M
ResNet101V2	93.19%	98.77%	45	12:08:03	44.6M
MobileNetV3Large	92.65%	98.68%	48	12:02:37	5.4M
VGG19	92.62%	98.79%	29	12:26:21	143.6M
Xception	92.49%	98.69%	27	12:06:12	22.9M
ResNet152V2	92.45%	98.54%	40	12:02:22	60.3M
EfficientNetB1	92.38%	98.51%	37	12:19:43	7.8M
DenseNet201	85.26%	95.85%	41	12:07:23	20.2M
NASNetMobile	78.59%	93.46%	41	12:09:57	5.3M

and thus costly in terms of calculation time both at the time of training and inference. For practical application, the second model, ResNet50, might be thus a better choice, as it has Top1 accuracy lower only by 0.75 percentage point, while 5.4 times lower the number of parameters. This model might also be a better choice after extending the training set with images representing other LEGO bricks, that currently are not taken into consideration (and thus extending the number of classes almost tenfold).

Very good results were also obtained by a mobile-oriented models, especially MobileNetV3Large, which had only 1.91 percentage point lower Top1 accuracy than VGG16 model. Furthermore, it contains only 5.4 million parameters (in contrast to 138.3 million for VGG16). Thus in applications where computing performance is scarce, MobileNetV3Large should be used over any more complicated model. Despite its size, it outperforms in terms of accuracy other, more complicated models (VGG19, Xception, ResNet152V2 and EfficientNetB1).

DenseNet201 and NASNetMobile did not reach convergence in the twelve-hour time limit and thus did not achieve good results. DenseNet201 suffered from overfitting and NASNetMobile had very slow accuracy increase and would require much longer training time.

5.3 Final tests

We performed some final tests on the two best models. The results for the easy and hard sets are presented in Table 3. As can be seen, both models reach similar accuracy.

To test the models in real life application we implemented a mobile app which took photos of LEGO bricks laying on a white background and combined it with the pre-trained models. VGG16 correctly recognized 39 out of 40 bricks. Wrongly labeled 822931 brick was classified as 3003 due to their similarity from the camera perspective. ResNet50 correctly classified all bricks. The networks were tested in different conditions. We used an intensive pink light to illuminate the test environment. This made the background pink and most of the bricks appeared

MOST WIEDZY Downloaded from mostwiedzy.pl

Table 3. VGG16 and ResNet50 accuracy for easy and hard tests

	Easy set		Hard set	
	Top1 accuracy	Top5 accuracy	Top1 accuracy	Top5 accuracy
VGG16	92.37%	99.02%	86.08%	98.22%
ResNet50	90.40%	99.05%	86.30%	98.60%

as having different, not seen before, colors. VGG16 correctly recognized 37 bricks out of 40. ResNet50 model once again correctly classified all of the bricks.

6 Conclusions

The paper presents extensive analysis of deep neural network architectures in order to verify their suitability for classification of LEGO bricks. The problem is characterized with the need to distinguish objects between multiple, often similar classes, as there are over 3700 different LEGO brick shapes. For this purpose, a new dataset was created containing 447 classes and a set of tools automating the analysis process were implemented. In total, 28 network architectures, belonging to 7 families, were analyzed and compared. For the comparison, we used our proposed training algorithm with adaptive fine-tuning approach.

Results showed that VGG16 model proved to be the best with its Top1 accuracy of 94.56% and Top5 accuracy of 99.21%). Surprisingly, in ILSVRC competition this model was outperformed by other solutions. The model is characterized, however, with very big number of parameters (138.3 million) and high number of floating point operations during training and inference process (15.3 GFLOPs). Not falling far behind was ResNet50 model (Top1 93.81%, Top5 99.10%) which had lower parameter count (25.6 million) and required far lower system performance (3.87 GFLOPs). In many cases, this might be the best choice for similar problems, where there are a lot of similar objects to classify. Surprisingly, also the smaller, mobile models proved to be worthwhile. MobileNetV3Large achieved very good accuracy (Top1 92.65%, Top5 98.68%), with very low parameter count (5.4 million) and low performance requirements (0.21 GFLOPs).

The two best models also were tested in real life application. They proved to be very accurate in both synthetic test on predefined test sets and during live classification of LEGO bricks.

In the near future we plan on extending the dataset with additional classes to cover as much LEGO brick shapes as possible to provide a deep neural network able to classify any type of LEGO bricks. Such network could be used in LEGO sorting machines, software recommending constructions based on the bricks available, automatic brick database creation and many more.

Acknowledgment

The authors would like to thank Bartosz Śledź and Sławomir Zaraziński for help with part of the implementation and dataset creation.

MOST WIEDZY Downloaded from mostwiedzy.pl

References

1. Adam: LEGO sorting chart. https://go.gliffy.com/go/publish/12232322 (2019), accessed: 2022-03-24
2. Aplhin, T.: The LEGO storage guide. https://brickarchitect.com/guide/ (2020), accessed: 2022-03-24
3. Boiński, T.: Images of LEGO bricks (2021). https://doi.org/10.34808/xz76-ez11, accessed: 2022-03-24
4. Boiński, T., Zaraziński, S., Śledź, B.: LEGO bricks for training classification network (2021). https://doi.org/10.34808/3qfs-rt94, accessed: 2022-03-24
5. Boiński, T., Zawora, K., Zaraziński, S., Śledź, B., Łobacz, B.: LDRAW based renders of LEGO bricks moving on a conveyor belt (2020). https://doi.org/10.34808/jykr-8d71, accessed: 2022-03-24
6. Boiński, T.M.: Hierarchical 2-step neural-based LEGO bricks detection and labeling. In: Proceedings of 37th Business Information Management Association Conference. pp. 1344–1350 (2021)
7. Chollet, F.: Xception: Deep learning with depthwise separable convolutions (2017)
8. Dua, D., Graff, C.: UCI machine learning repository (2017), http://archive.ics.uci.edu/ml, accessed: 2021-11-22
9. Foundation, T.B.: Blender. https://www.blender.org/ (2002), accessed: 2021-11-22
10. Garcia, P.: LEGO Sorter using TensorFlow on Raspberry Pi. https://medium.com/@pacogarcia3/tensorflow-on-raspbery-pi-lego-sorter-ab60019dcf32 (2018), accessed: 2021-11-22
11. He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition (2015)
12. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks (2016)
13. Hochreiter, S.: The vanishing gradient problem during learning recurrent neural nets and problem solutions. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **6**(02), 107–116 (1998)
14. Howard, A., Sandler, M., Chu, G., Chen, L.C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q.V., Adam, H.: Searching for mobilenetv3 (2019)
15. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications (2017)
16. Huang, G., Liu, Z., van der Maaten, L., Weinberger, K.Q.: Densely Connected Convolutional Networks (2018)
17. Jessiman, J.: LDraw. http://www.ldraw.org (1995), accessed: 2021-11-22
18. Jung, A.: imgaug source code. https://github.com/aleju/imgaug, accessed: 2021-11-22
19. Krasin, I., Duerig, T., Alldrin, N., Ferrari, V., Abu-El-Haija, S., Kuznetsova, A., Rom, H., Uijlings, J., Popov, S., Kamali, S., Malloci, M., Pont-Tuset, J., Veit, A., Belongie, S., Gomes, V., Gupta, A., Sun, C., Chechik, G., Cai, D., Feng, Z., Narayanan, D., Murphy, K.: Openimages: A public dataset for large-scale multi-label and multi-class image classification. Dataset available from https://storage.googleapis.com/openimages/web/index.html (2017)
20. Krizhevsky, A., Nair, V., Hinton, G.: Cifar-10 and cifar-100 datasets, https://www.cs.toronto.edu/~kriz/cifar.html, accessed: 2022-03-24

MOST WIEDZY Downloaded from mostwiedzy.pl

21. Kuznetsova, A., Rom, H., Alldrin, N., Uijlings, J., Krasin, I., Pont-Tuset, J., Kamali, S., Popov, S., Malloci, M., Kolesnikov, A., Duerig, T., Ferrari, V.: The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. IJCV (2020)
22. Li, M., Liu, Y., Liu, X., Sun, Q., You, X., Yang, H., Luan, Z., Gan, L., Yang, G., Qian, D.: The deep learning compiler: A comprehensive survey. IEEE Transactions on Parallel and Distributed Systems **32**(3), 708–727 (Mar 2021). https://doi.org/10.1109/tpds.2020.3030548
23. Lin, T., Maire, M., Belongie, S.J., Bourdev, L.D., Girshick, R.B., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft COCO: common objects in context. CoRR **abs/1405.0312** (2014), http://arxiv.org/abs/1405.0312
24. Maren, T.: 60 fun LEGO facts every LEGO fan needs to know. https://mamainthenow.com/fun-lego-facts/ (2018)
25. Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., Wu, H.: Mixed precision training (2018)
26. Nelson, T.: ImportLDRaw. https://github.com/TobyLobster/ImportLDraw, accessed: 2021-06-16
27. Papert, S.A.: The summer vision project (1966), https://dspace.mit.edu/handle/1721.1/6125, accessed: 2021-11-22
28. Quoc Le & Barret Zoph, Research Scientists, G.B.t.: Using machine learning to explore neural network architecture (2017), https://ai.googleblog.com/2017/05/using-machine-learning-to-explore.html, accessed: 2021-11-22
29. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) **115**(3), 211–252 (2015). https://doi.org/10.1007/s11263-015-0816-y
30. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks (2019)
31. Shukla, N., Fricklas, K.: Machine learning with TensorFlow. Manning Shelter Island, Ny (2018)
32. Simonyan, K., Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition (2015)
33. Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A.: Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning (2016)
34. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going Deeper with Convolutions (2014)
35. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the Inception Architecture for Computer Vision (2015)
36. Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: Mnasnet: Platform-aware neural architecture search for mobile. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 2820–2828 (2019)
37. Tan, M., Le, Q.V.: EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks (2020)
38. West, D.: LEGO sorting machine. https://twitter.com/JustASquid/status/1201959889943154688 (2019), accessed: 2021-02-08
39. Zagoruyko, S., Komodakis, N.: Wide residual networks (2017)
40. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning (2017)
41. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning Transferable Architectures for Scalable Image Recognition (2018)

MOST WIEDZY Downloaded from mostwiedzy.pl