DOI: 10.15439/2016F211
ACSIS, Vol. 9. ISSN 2300-5963

A practical study of neural network-based image classification model trained with transfer learning method

Marek Dąbrowski, Justyna Gromada, Tomasz Michalik
Orange Polska, Centrum Badawczo-Rozwojowe, ul. Obrzeżna 7, 02-691 Warszawa
Email: {marek.dabrowski, justyna.gromada, tomasz.michalik}@orange.com

Abstract—This paper deals with algorithms for image classification, which aim to guess "what is on the picture" using human-readable labels or categories. A supervised learning approach with Convolutional Neural Networks (CNNs) is studied as an effective solution to different computer vision problems, including image classification. Main contribution of this paper is a set of practical guidelines to tackle the image classification problem using publicly available tools and typical hardware platforms.

I. Introduction

MACHINE learning is an area of computer science, which assumes that a computer program may "learn" by experience. In other words, the performance of a program to do its job may improve thanks to observation and analysis of actual data samples. As an example, carefully crafted artificial neural network may learn to classify images (name an object on a photo), after it has been trained with sufficiently large number of labeled image examples.

A. Computer vision

The ultimate goal of computer vision research is to teach computers to see and understand images, in a similar way as humans do. Remark that normally the images (photographs, sketches, figures) are represented in computer memory as sets of pixels, and more precisely as bytes with value depending on color intensity of particular pixel. The computer representation of an image does not normally hold any semantic information, unless the user has explicitly provided some semantic context metadata. Thus, it is very difficult for traditional computer algorithm to guess what is the semantic content of a picture.

Machine learning approach has recently been successfully used for solving numerous problems related with understanding of images:

- **Image classification**. A basic problem of computer vision, with goal of characterizing given image by assigning it a human-understandable text label (what is on the picture – is it a person? or a dog? or a building?). The label is usually chosen from a known set of categories, thus this problem is of "classification" type.
- **Classification with localization**. Instead of assigning a single label as in basic image classification, multiple labels may be more appropriate if an image displays not one, but several objects. Additionally, the localization function provides coordinates (bounding boxes) of objects detected on a photo.
- **Object detection**. Quite similar to the previous one, but it rather assumes that we would like to detect presence of a limited set of objects. For example, an object detection algorithm implemented in an autonomous car control system may localize pedestrians (persons) on the side of the road. In this case, there is a single type of object ("person") to be detected.
- **Instance segmentation**. The result of classification with localization is typically a set of identified objects with their approximate bounding boxes (rectangles). Sometimes a more detailed approach would be preferred, being able to assign particular pixels of the image to detected objects. For example, if we detect three persons on a photo, we would have also a precise boundary of each person's shape.

In this paper we focus specifically on the basic problem of image classification.

II. Convolutional Neural Networks: State of the Art

A. Artificial neural networks

Artificial neural network is a biologically-inspired machine-learning model for solving classification problems [1][2]. A simple model of an artificial neuron is presented in Fig.1. A neuron takes a set of inputs (x_i) and produces output value y, applying a non-linear activation function on a weighted sum of inputs. A neural network consists of multiple neurons, connected to each other to form multi-layer structures. If the output is calculated as linear combination of all inputs, such neuron is sometimes called a "fully-connected" unit.

Before being able to actually solve a classification problem, a neural network must be "trained". It means that its parameters (weights) are tuned by a special algorithm, which takes as input a large set of training data. Training data consist of input values together with "ground truth" result, that is a result, which we know is correct for given input.

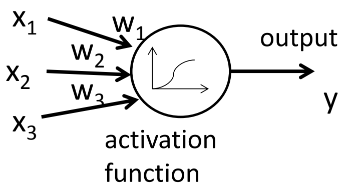


Fig. 1 Model of a simple artificial neuron

A special family of artificial neural networks, called “convolutional”, is especially successful for image classification [2]. Let us assume that input to the model consist of certain number of pixels, with value corresponding to color intensity (see Fig.2).

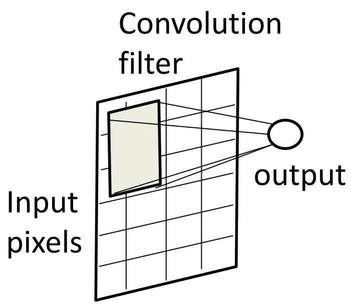


Fig. 2 Model of a convolutional unit

Instead of calculating the output value by taking a combination of all inputs, the output value of convolutional unit is based on a locally constrained region of several pixels (e.g. 3x3, 5x5, 7x7). Moreover, the output is calculated for each local area on the image, like a filter which is slided all over it. The idea is to take into account the structure of computer image and analyze in more detail pixels in a local neighborhood, in order to detect basic shapes, edges and other small characteristic graphical features. Then, a multi-layer hierarchy of convolutional units is applied, being able to detect more high-level shapes and structures of an image.

As a final layer of neural network model, fully-connected neurons are always used. In particular, the last layer consists of *k* neurons, where *k* is the number of image classes that the model is able to distinguish. After passing an image (represented as pixels intensity values) through all layers of neural network, the value outputted by k_{th} neuron of the final layer will correspond to estimated probability that the image belongs to the k_{th} class. Thus, this final layer is sometimes called a “classification layer” because it is aware of set of classes of a given model, and is trained to recognize the class of the image based on features that are spotted by internal layers.

B. State-of-the-art models for image classification

The basic artificial neuron structures as depicted on Fig.1. and Fig.2 constitute main building blocks of modern neural networks. But to really appreciate its power for solving classification tasks, we need much more complex structures with thousands, or even millions of such small basic units. Multi-layer structures appear to be more effective, which led to a concept “deep learning”, which denotes artificial neural networks consisting of many layers of connected units [1][2].

Looking back into history of research work in this area, a model published in 1998, named “LeNet” [3], is considered as pioneering work in applying convolutional neural networks to computer imaging problems. It had 8 layers (convolutional and fully-connected), with around 1mln parameters in total. LeNet primary application was to recognize handwritten digits in banking information systems.

A new wave of research in neural networks came few years ago thanks to several breakthrough advances: invention of deep networks, more efficient training algorithms, availability of more powerful hardware for computationally intensive calculations, and availability of large data sets for training. In 2012, so-called “AlexNet” model was proposed [14], with 14 layers and about 60mln parameters. In the trend of building deeper networks, the “GoogleNet Inception” model [4] has been proposed in 2014, with 22 layers (Fig.3). It consists of convolutional (blue boxes on Fig.3) and fully-connected units (yellow), together with pooling units (red) and joining units (green). Thanks to more efficient use of convolutional units it had only 5mln parameters, which made it easier to train and run on less powerful hardware, while being more effective than the older models. A newer version of “GoogleNet” architecture will be used in experiments presented later in this paper.

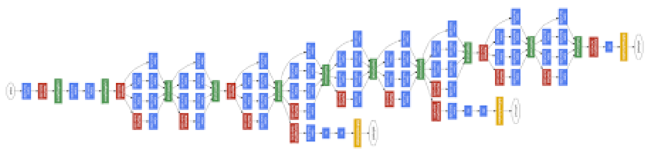


Fig. 3 “GoogleNet”: state-of-the-art multi-layer convolutional model for image classification

C. Imagenet image corpus

The neural network models described in previous section could never work well without being trained on a very big sets of learning data. A publicly available ImageNet database [5] is a widely used repository of properly labelled photos (i.e. photos with text label correctly describing its content). The ImageNet corpus contains 21841 classes of images. Each class is described by a label, which is called a “synset” by referring to the WordNet taxonomy of concepts. With an ultimate goal of collecting around 1000 images for each class, the ImageNet repository has now about 14mln human-annotated images.

Once a year the ImageNet team organizes a competition for computer vision researchers to propose world-best models for solving image classification problem. For the purpose of this competition, a subset of 1000 basic ImageNet classes has been selected as a reference corpus. This image corpus with 1000 classes has been used in experiments presented in this paper.

III. TRANSFER LEARNING

The "deep learning" approach adopts artificial neural network models with large number of hidden layers and millions of parameters. For training such models, the backpropagation method seems to work well [2]. Basically, it assumes minimization of cost function (also called loss function), which describes overall difference between the predicted and actual (ground truth) classification results, for a given set of training examples. Several methods are typically used for the optimization algorithm, usually being variations of Stochastic Gradient Descent [2].

The computational cost of the backpropagation method is noticeable, especially that the training sets should be huge, with tens of millions of training examples. Despite of recent advances in hardware processing, distributed computing and Graphical Processing Units (GPUs), which greatly accelerate computations, it may take weeks to train a model until it reaches sufficient accuracy [4].

To avoid the lengthy learning process and allow for training with smaller amount of training examples, so-called Transfer Learning method has been proposed [6]. Transfer Learning is a machine learning method, which is improved thanks to transfer of knowledge from a related task, that has already been learned. In other words, new image classification models may be trained much faster, thanks to using parameters of a previously trained model. The process for practical usage of Transfer Learning is the following (see Fig.4):

1. Get a previously trained model. Pre-trained models published by other researchers are available e.g. from [8][9].
2. Split the old model architecture into two parts:
   a. All hidden layers of the neural network, with their structure (connections) and previously learned weights, will be copied into the new model.
   b. The last layer of neural network, which performs actual classification into one of the classes, is strictly related with the old model and will be disregarded in the new model.
3. Prepare a new set of training examples (images labelled with appropriate class name, as required for the new model).
4. For a new set of training images, calculate the output values after passing through the first part of neural network (the one that is transferred into the new model). The numerical value calculated as output of given image, at the next-to-last layer of original model, will be called a "bottleneck".
5. Add a new final fully-connected layer, which will now constitute the last layer of new neural network model. This new final layer will calculate the probability of given image belonging to a given class.
6. Train the new final layer with previously calculated "bottlenecks" as input, and a set of new "ground truth" labels that denote true classes of training images.

Remark that in this method only the last layer of artificial neural network model has to be trained from scratch, while for all previous layers (and remind that for example the GoogleNet model has 22 of them) the weight values are copied from the previously pre-trained model. Thanks to that, we can create a new image classification model, with our own classes and labels, within several hours instead of weeks, on standard hardware.

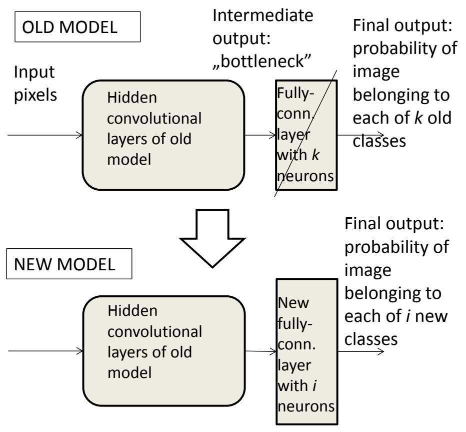

OLD MODEL
Intermediate output: „bottleneck"
Final output: probability of image belonging to each of k old classes
Input pixels
Hidden convolutional layers of old model
Fully-conn. layer with k neurons
NEW MODEL
Final output: probability of image belonging to each of i new classes
Hidden convolutional layers of old model
New fully-conn. layer with i neurons

Fig. 4 Illustration of "Transfer Learning" method

IV. DEPLOYMENT SCENARIO

Running Machine Learning algorithms and neural networks especially requires certain amount of computing resources, not only processing power, but also storage and access to big repositories of training data. Thus, a practical deployment approach may assume a networked environment, with computational resources deployed in cloud data center, with Web Services API developed for client applications to upload images and receive results over the web. An example deployment scenario is presented in Fig.5.

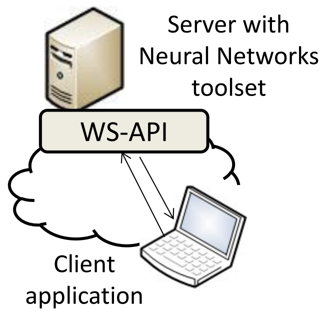


Fig. 5 Networked deployment scenario

V. EXPERIMENTAL SETUP

A. *Tensorflow: open source tool for neural networks*

For our experiments we have used Tensorflow [9] library for numerical computation. Tensorflow has been created by Google and made available as open source. It supports various types of numerical computations, including complex neural network models, on various types of hardware, CPU and GPU. It supports the Transfer Learning method.

B. *Lab infrastructure*

It is well known that GPU hardware greatly improves performance of machine learning computations. Having a goal to find optimal hardware setup for our machine learning task (minimize platform cost, assuming satisfactory training and testing time) we have evaluated performance of several typical medium-level hardware platforms.

We wanted to simulate a home or small company environment, in addition to large expensive data center.

1) PC computer: a typical desktop with 4 CPU cores and 8GB RAM. It had a GPU card NVIDIA GeForce GTX 960. Tensorflow computations may or may not use GPU depending on software configuration, so both hardware settings were tested (PC-CPU and PC-GPU).

2) ODROID: „mini" home computer of SBC (Single Board Computer) type [12]. It is a fully-fledged computer with a very small size, low power consumption and low price (see Fig. 6). It is not as powerful as a desktop PC, but has sufficient capabilities for simple home tasks.

- *ARM architecture*
- *SOC: Samsung Exynos*
- *32bit architecture*
- *2GB RAM*
- *Flash disk*
- *USB, Eth, HDMI*
- *Linux*
- *Price:* ~70$

Fig. 6. ODROID: exemplary SBC device (Single Board Computer)

VI. EXPERIMENTAL VERIFICATION OF TRANSFER LEARNING METHOD

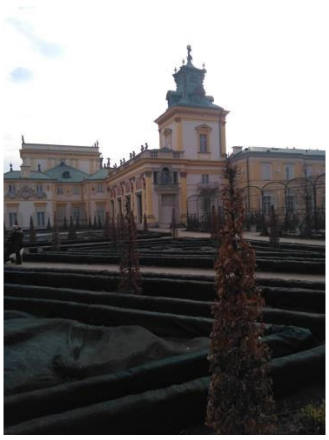
The goal of experiments was to verify if the Transfer Learning method, applied to training image classification models, allows for achieving results that are as good as full-fledged training, in much quicker time and with less computational resources. The GoogleNet model [10] trained with ImageNet-1000 corpus has been used as basis for re-training. The ImageNet database of basic 1000 classes has been downloaded and used as re-training examples, following the Transfer Learning concept.

Remark that we have decided to use in re-trained model the same 1000 ImageNet classes as in the original model. At first look it may be counterintuitive: why we would re-train the model to have the same result at the end? Of course, in a target scenario the re-training procedure would assume completely different target set of classes, with different set of training images than the original model. But remind that the goal of experiments presented in this paper was to evaluate the correctness of Transfer Learning and so it seems methodologically correct to compare the re-trained model with the original one, that was created with the same initial assumptions and target set of classes.

A. *Testing the image recognition capability*

First, let us discuss what is the expected result of image classification. Say, we have a photo, and we would like to tag it automatically with a text label. The result of image classification algorithm should thus be a word, or a few words, matching with certain level of confidence semantic contents of the photo. Strictly speaking, the result of passing the image through neural network model trained with 1000 classes will be a vector of 1000 numbers, corresponding to the "score" associated to each class. The "scores" are conceptually related to a likelihood that given result is a correct one. The distribution of score values for all classes should form a proper probability distribution, so the scores will sum up to 1.0.

An exemplary photo with classification result is presented in Fig.7 (the result has been calculated with one of the models trained in the scope of this study, but this is not important at this moment, as it is presented merely as illustration of intended goal of the algorithm). More precisely, this is a "Top-5" result, that is five class names with the highest values of scores. In our example, the score associated with class "palace" is 0.97, which means that the algorithm thinks with very high confidence that there is a "palace" on the photo, which is actually true.



- "palace" (0.97)
- "monastery" (0.11)
- "fountain" (0.0036)
- "castle" (0.0022)
- "church, church building" (0.0019)

Fig. 7. Result of re-trained classification algorithm on a real-life photo example

This simple example shows that image classification algorithm put in the realistic setting produces quite accurate results. However, we need more rigorous and repetitive method to evaluate objectively the correctness of the method, as applied to a larger set of images.

Assume that we have a test set of *N* images, drawn from the ImageNet corpus and thus human-annotated in controlled way with a "ground truth" label. Remark that according to Machine Learning established practice, the examples from the test set must not be previously used as training data, since that would bias the test result towards positive outcome. For all images of the test set, the test result is produced by the evaluated algorithm, in the form of 5 classes with maximum associated scores among the all the *k* classes.

The metric "Top-1 accuracy" will be defined as ratio of correct classification results in the entire test set (where positive result means: "the label with maximum score is equal to ground truth"). The Top-1 accuracy metric may be too restrictive in realistic scenarios. Take an example in Fig.1 – "palace" is a true object on the image indeed, but if you just look at the photo, without prior knowledge of what it actually is, it may appear that "castle" could also a possible name for an object on photograph. Thus, another metric, "Top-5 accuracy" has been defined to take into account that sometimes the ground truth label may be among the best recognized, but not necessarily at the first place. Top-5 accuracy is also defined as ratio of correct classification results in the test set, but the positive result is now: "the ground truth label is among the 5 maximum-score labels assigned by the classification algorithm".

We have validated the studied model re-trained with Transfer Learning method on a test set of 50000 random images from the ImageNet corpus. The obtained Top-1 and Top-5 accuracy metric results are presented in Table 1.

TABLE I. RESULT OF FINAL EVALUATION OF RE-TRAINED MODEL ON A TEST SAMPLE OF 50000 IMAGES FROM IMAGENET CORPUS

Metric	Test result
Top-1 accuracy	88.2 %
Top-5 accuracy	98.0 %

These results for re-trained model are impressively good. In fact, for 88% of ImageNet photos the algorithm is able to classify correctly the true label, while for 98% the true label is among top 5 assigned ones. Surprisingly, the achieved accuracy appears even better than the accuracy of original model, reported in [10] (top-5 accuracy equal to 93.33%). We think that the reason for this misleading result is that for testing the re-trained model we have used the ImageNet images that belonged to the training set of original model. Thus, the test result is higher than expected, because in some sense the artificial neural network is tested with images that it has already seen before. Unfortunately, the test set that was used by authors of the original model in [10] is not publicly available, so we were not able to make a truly relevant comparison. Nevertheless, we think that the conclusion that can be drawn from our study is that re-training the artificial neural network using Transfer Learning method may give us a model that is as good as the original one, in relatively short time on typical modern computers (see later in the paper).

B. Finding a parameter setup for transfer learning method

There are several parameters that we can shuffle in order to obtain a satisfying model within reasonable training time. Our goal was to achieve well trained model (not over fitted, nor under fitted) with the test accuracy around 90% for Top-1 accuracy metric and close to 100% for Top-5 accuracy metric. The intended time for re-training a model on our infrastructure was max 12h, which meant running experiments taking 32000 training steps which is around 3 epochs for our training data set.

The main parameters that we tuned were: type of the optimizer and value of learning rate. The optimizer takes the loss computed in forward propagation part (in our case, it is the loss function for softmax classifier [1], called cross-entropy loss), calculates the gradients in backward propagation and then changes the weights of the model trying to minimize the loss. In case of Transfer Learning algorithm, the optimization process concerns only one layer – the last one. The learning rate value is a hyper-parameter which tells how fast the optimizer should converge to minimal loss. When the learning rate is to low, the process of training may last very long to achieve optimal values or never achieve it but on the other hand when the learning rate is too high, the average loss may increase which is opposite to our goal. Hence, the choice of learning rate and appropriate optimizer is crucial.

We have chosen two optimizers for our experiments: Stochastic Gradient Descent Optimizer (SGD) [13] and Adam Optimizer (Adam) [11] and used them in training process with various values of hyper-parameter learning rate $\alpha = 0.01$, $\alpha = 0.05$ and $\alpha = 0.1$. To choose the best configuration of these parameters, we were observing how the cross-entropy loss, training Top-1 accuracy and validation Top-1 accuracy behave during the training process (Fig.8-11).

When we look on the graph of cross-entropy loss in function of training steps (Fig.8*)*, three potential candidate configurations seem the most promising: the Stochastic Gradient Descent Optimizer with learning rate $\alpha = 0.1$, Adam Optimizer with $\alpha = 0.05$ and Adam Optimizer with $\alpha = 0.01$. The cross-entropy loss in these cases constantly goes down and within 3 epochs reaches the value of around 0.36. The cross-entropy loss for Stochastic Gradient Descent Optimizer with learning rate $\alpha = 0.01$ also goes down but much slower than in previous three configurations. The cross-entropy loss for Adam Optimizer with learning rate $\alpha = 0.1$ after 2000 training steps goes up, which is undesirable behavior and may suggest too large learning rate.

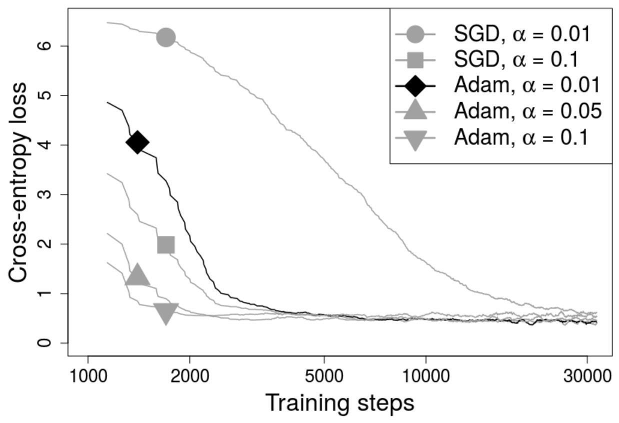


Fig. 8. Entropy in function of training steps (log scale) for Stochastic Gradient Descent Optimizer (SGD), Adam Optimizer (Adam) and various learning rates (α)

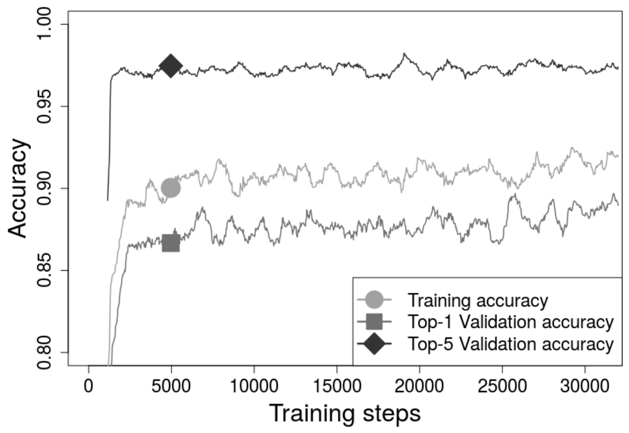


Fig. 9. Accuracy for Stochastic Gradient Descent Optimizer and learning rate α = 0.1

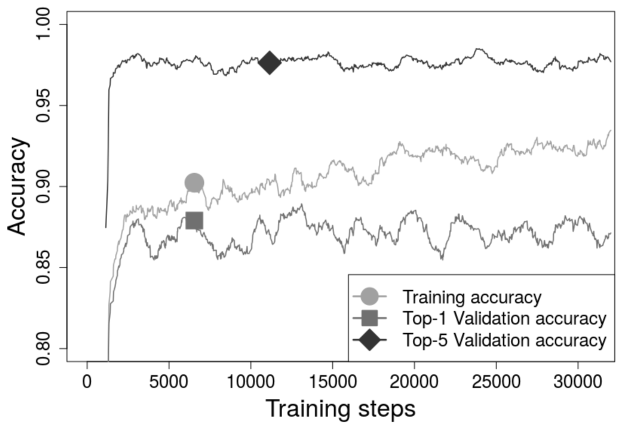


Fig. 10. Accuracy for Adam Optimizer and learning rate α = 0.05

The Top-5 Validation accuracy for all three chosen configurations achieve very quickly a desirable value of around 97% so we will focus on training accuracy and Top-1 validation accuracy when comparing the pointed solutions.

For Stochastic Gradient Descent Optimizer with learning rate α = 0.1 (Fig.9), we can observe a big gap between the training accuracy and Top-1 validation accuracy.

For Adam Optimizer with learning rate α = 0.05 (Fig. 10), the situation is a bit worse: the gap between training accuracy and Top-1 validation accuracy is not only big but it is even increasing.

The big gaps between training accuracy and Top-1 Validation accuracy for both configurations: Stochastic Gradient Descent Optimizer with learning rate α = 0.1 and Adam Optimizer with learning rate α = 0.05 suggest that the models might be over fitted.

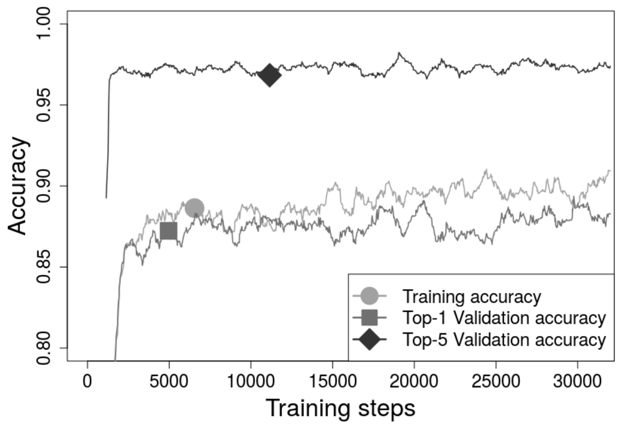


Fig. 11. Accuracy for Adam Optimizer and learning rate α = 0.01

The last candidate: Adam Optimizer with learning rate α = 0.01 (Fig.11) gives both acceptable values for Top-1 validation accuracy of around 0.87 and acceptable gap between the training accuracy and Top-1 validation accuracy. What is more, the figure of cross-entropy loss in this case also seem the best: it reaches the lowest values after 10000 training steps and constantly goes down. Hence, this configuration has been chosen as a final setup for Transfer Learning method.

Summarizing, chosen configuration of parameters for Transfer Learning method is as follows: training steps= 10000, Adam Optimizer with learning rate α=0.01 and epsilon=0.1 (a small constant for numerical stability), train batch size=100.

VII. Performance benchmarking

Next, we have performed a series of experiments to benchmark hardware configurations available in our lab (see section V B) as platforms for re-training image classifier models. As discussed in section III, we can distinguish several phases in the process of Transfer Learning method and so the performance benchmarks were performed separately for each one of them.

A. *"Bottleneck" pre-calculation phase*

First computationally intensive step is to determine the "bottleneck" values, that is an output of pre-last layer of neural network, given a single image at the input. Measured time of performing that operation, averaged over 50 test runs for different images, is presented on Fig.12. Not surprisingly, the Odroid platform is the slowest one, with 2s to calculate a single bottleneck. The PC platform with GPU card is a clear winner, with 0.08s time. Remark that the bottleneck calculation time includes the time to read the image file from disk. On all studied platforms this time was below 0.004s, so we consider it negligible.

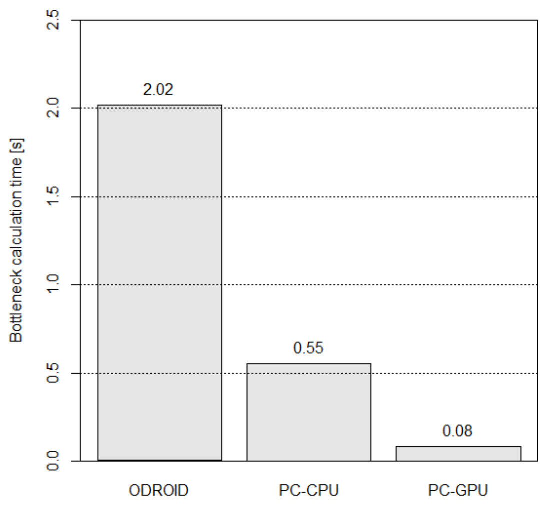


Fig. 12. Calculation time of "bottleneck" value for 1 image

The bottleneck calculation operation has to be done once for every image in the training corpus. The result can be saved in a file, for quick access during the re-training phase later. Taking that into account, Table II presents estimated total time to calculate bottlenecks for all images in the training set, for different number of target classes, and assuming approximately 1000 images per class. We can see that for a 1000-class model, it takes about 1 day on our fastest platform to prepare "bottleneck" values for entire image corpus. Fortunately, this operation is done just once, with results saved on disk for quick access during re-training phase.

TABLE II. ESTIMATED BOTTLENECK CALCULATION TIME FOR WHOLE IMAGE CORPUS

Number of classes	**ODROID**	**PC-CPU**	**PC-GPU**
100	56h	15.3h	2.2h
250	140h	38.3h	5.7h
1000	23d 8h	6d 9h	22.7h

B. Re-training phase

Now, the final classification layer of neural network is being trained with training images. The process follows a Stochastic Gradient Descent algorithm [2] with "mini-batch". In each step, the values of loss function, gradients and weight updates are calculated for a batch of images, in our tests equal to 100, 1000 and 10000. Measured average time of training step, normalized for 1 training image (i.e. divided by batch size) is presented in Fig.13. We can distinguish the following time components:

- **File access time** is the time to read the "bottleneck" value from an a-priori saved file, as discussed in section VIIA. Remark that this time component is highly related with performance of disk file system, rather than processor computing power. It is a little bit surprising that it has the biggest impact on total training time. Remark that at each training step, a large number of small files (equal to batch size: 100, 1000, 10000) is read from disk and then processed in-memory. We may presume that optimization of disk access, e.g. by using faster disks, or pre-caching all training data in memory could bring significant reduction of this phase (a single "bottleneck" file has about 18kB size, which means that for entire image corpus we need about 18GB - unfortunately for current experiments we didn't have a machine with sufficient amount of RAM).
  Still, we can see that the time to access pre-cached bottleneck data is still much smaller than the time to produce the data by doing full calculations, as measured previously (about 0.015s to read from disk, vs. 0.08s of calculations on fastest machine). This result confirms that it indeed makes sense to pre-calculate and save "bottleneck" data for later usage multiple times in re-training.
- **Training time** is the actual time to perform all mathematical calculations related with completing the training steps. This time appears to be negligible comparing with the time of accessing training data from disk.
- **Validation time**. Once every 500 batches, the validation step is performed, i.e. the value of top-1 and top-5 accuracy metric is calculated for a given validation set. This operation is done for the purpose of monitoring and logging the learning process, and has no impact on final result. But since it is usually done, we report it also in time benchmarks.

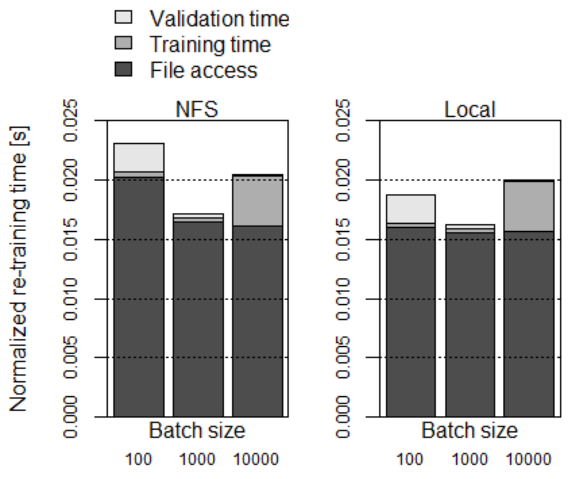


Fig. 13 Analysis of re-training time, normalized for 1 training example

The results presented in Fig.13 correspond to processing a single training image. For completeness, Table III presents measured total time of re-training with a corpus of images appropriate for a given number of classes (recall that we have around 1000 images per class). The reported time corresponds to 1 "epoch", that is training once with a full set of training data. Due to the fact that file access time has greatest impact on training time in this phase, the differences between hardware platforms are not so big. Surprisingly, the less powerful device (Odroid) performs best for this task. The explanation is that this device has a fast flash disk which beats in terms of file access time (which, as we saw has great impact on performance) the magnetic disc of our lab PC.

TABLE III. MEASURED TRAINING TIME OF 1 EPOCH (AVERAGE AND STANDARD DEVIATION)

Number of classes	ODROID	PC-CPU	PC-GPU
100	8.57 ± 0.57 m	14.24 ± 0.08m	14.57 ± 0.57m
250	22.91 ± 0.69m	36.55m ± 1.58m	36.66 ± 1.53m
1000	2h7m ± 3.18m	3h39m ± 16.35m	3h30m ± 8.14m

C. *Test phase*

Finally, test phase is when we want to actually obtain a classification result for an arbitrary photo. Fig.13 presents averaged measurement of time to calculate a final result. The following phases are distinguished:

- **Session run time**: actual time of running the calculations with image pixel values as input, and scores assigned to each class (there were 1000 classes in tested model), as output.
- **Result processing time**: time needed to prepare the result, i.e. sort the result table to extract 5 best scores, lookup the labels table to retrieve human-readable names of classes, and prepare the final result as json structure.

As expected, total test time is longest on the Odroid (session run time is around 6s). On the other hand, on a PC with GPU this processing is time is reduced to less than 2s.

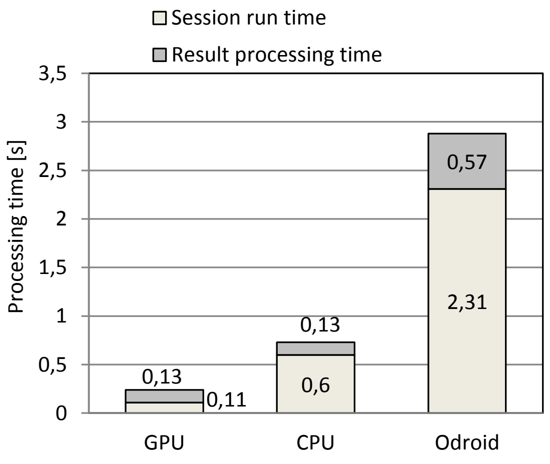


Fig. 14 Analysis of inference time (image testing) on different hardware platforms

VIII. SUMMARY AND FUTURE WORK

Transfer Learning method has been studied as efficient approach to training neural network models for image classification. Practical guidelines for setting configuration parameters of re-training process were given in the paper, which includes: number of training steps for achieving sufficient accuracy of re-trained model, type of optimization algorithm, and value of learning rate. Performance benchmarks for training image classification models on typical low- and middle-level hardware platforms were given. The measurements show significant advantage of using GPU card for computationally demanding operations, and, a little surprisingly, advantage of low capacity device equipped with ultra-fast disk, in the case of training phases where lots of training data must be accessed in short time. In future work we plan to extend the benchmarking experiments to cover wider range of hardware platforms, including machines with different CPUs, RAM size, and different GPU types.

It was shown that a good quality image classification model with our own set of classes can be obtained in several hours instead of weeks, by applying Transfer Learning method, that is re-training an existing neural network model downloaded from publicly available source and re-using most of the parameter values of original model. In future work we plan to study transfer learning with different sets of images than basic ImageNet-1000 corpus, with a goal to improve realism of our scenarios and transferring learned features to completely different classification task.

REFERENCES

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, “Deep Learning”, Book in preparation for MIT Press, 2016, on-line version available at:http://www.deeplearningbook.org

[2] Michael A.Nielsen, “Neural Networks and Deep Learning”, Determination Press, 2015, on-line version of the book available at: http://neuralnetworksanddeeplearning.com/index.html

[3] LeCun, Y., Jackel, L. D., Boser, B., Denker, J. S., Graf, H. P., Guyon, I., Henderson, D.,Howard, R. E., and Hubbard, W.. Handwritten digit recognition: Applications of neural network chips and automatic learning. IEEE Communications Magazine, 27(11), 1989

[4] Ch.Szegedy et al, “Going deeper with convolutions”, http://arxiv.org/abs/1409.4842

[5] ImageNet database of computer images: http://image-net.org/

[6] Yosinski J, Clune J, Bengio Y, and Lipson H. How transferable are features in deep neural networks? In Advances in Neural Information Processing Systems 27 (NIPS ’14), NIPS Foundation, 2014

[7] Caffe Model Zoo web page: https://github.com/BVLC/caffe/wiki/Model-Zoo

[8] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo,

[9] Z. Chen, et al., TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[10] Ch.Szegedy et al., “Rethinking the Inception Architecture for Computer Vision”, http://arxiv.org/abs/1512.00567

[11] D.Kingma, J.Ba, “Adam: A Method for Stochastic Optimization”, http://arxiv.org/abs/1412.6980

[12] ODROID-XU4 hardware : http://www.hardkernel.com/main/products/prdt_info.php?g_code=G143452239825

[13] Y. LeCun, L. Bottou, G. Orr and K. Muller: Efficient BackProp, in Orr, G. and Muller K. (Eds), Neural Networks: Tricks of the trade, Springer, 1998

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In NIPS 2012, Neural Information Processing Systems, Nevada, 2012