Nizhny Novgorod State University

Institute of Information Technologies, Mathematics and Mechanics

Department of Computer software and supercomputer technologies

Educational course
«Introduction to deep learning
using the Intel® neon™ Framework»

Transfer learning of deep neural networks

Supported by Intel

Valentina Kustikova,
Phd, lecturer, department of Computer software
and supercomputer technologies

Content

- The concept of transfer learning
- Application of transfer learning
- Transfer learning in the Intel® neon™ Framework

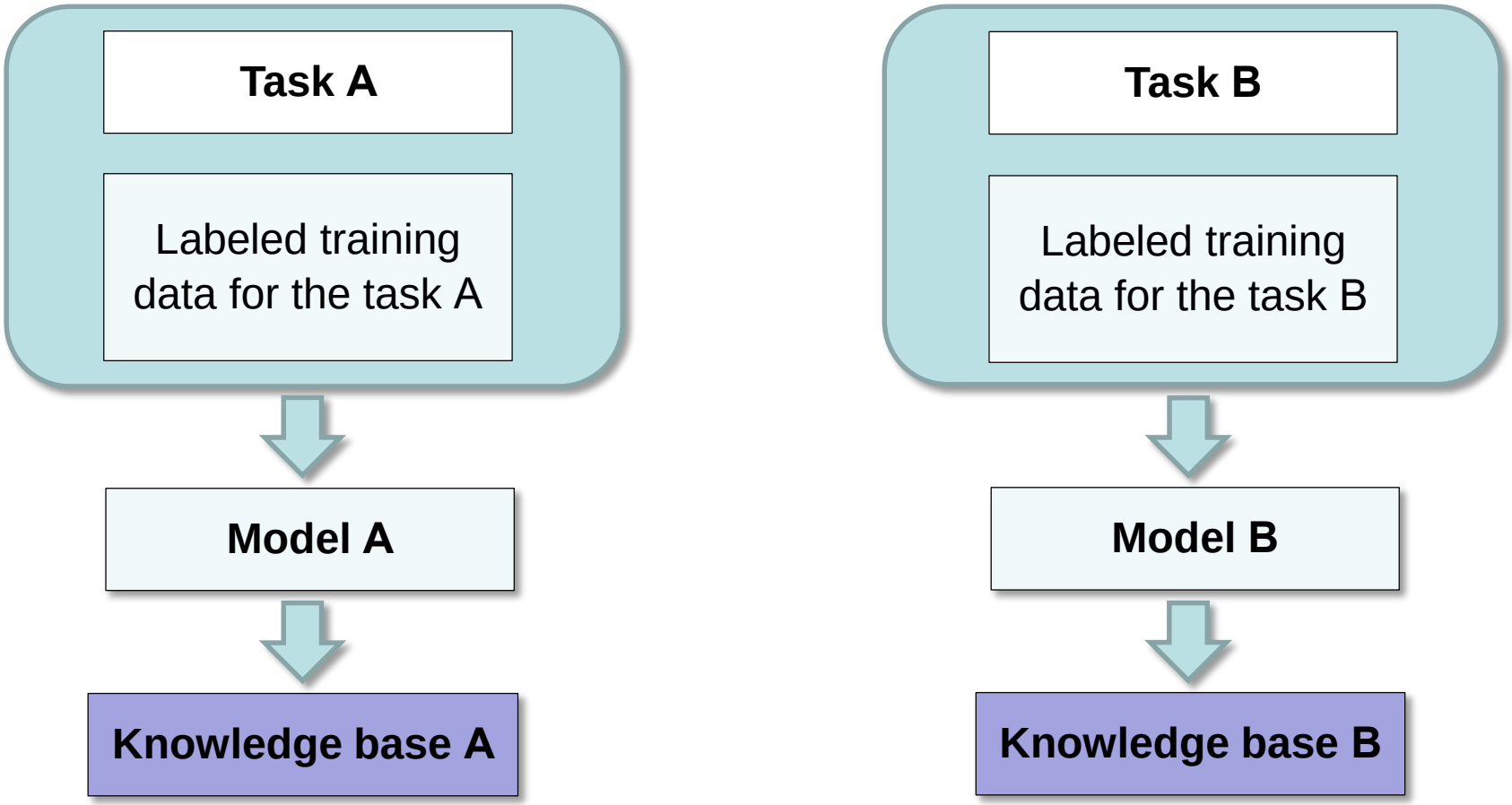
THE CONCEPT OF TRANSFER LEARNING

Introduction (1)

- Researchers are able to construct effective deep models for solving a wide range of tasks in the field of computer vision, image processing and natural language processing
- The training of such models requires a significant amount of labeled training data, since most of problems are solved using supervised learning

Introduction (2)

- “Task A” is an original task
- “Task B” is a target task



Introduction (3)

- If the original “Task A” and the target “Task B” are related in some way, can you somehow use the knowledge obtained in solving the original problem for the target one?

Introduction (4)

- If the original “Task A” and the target “Task B” are related in some way, can you somehow use the knowledge obtained in solving the original problem for the target one?

- Yes, but how?

Introduction (5)

- If the original “Task A” and the target “Task B” are related in some way, can you somehow use the knowledge obtained in solving the original problem for the target one?

- Yes, but how?
- Example:
  - The original “Task A” is a pedestrian detection in the daytime
  - The target “Task B” is a pedestrian detection at night
  - It is reasonable to use the model, trained to solve the original problem, for solving the target problem

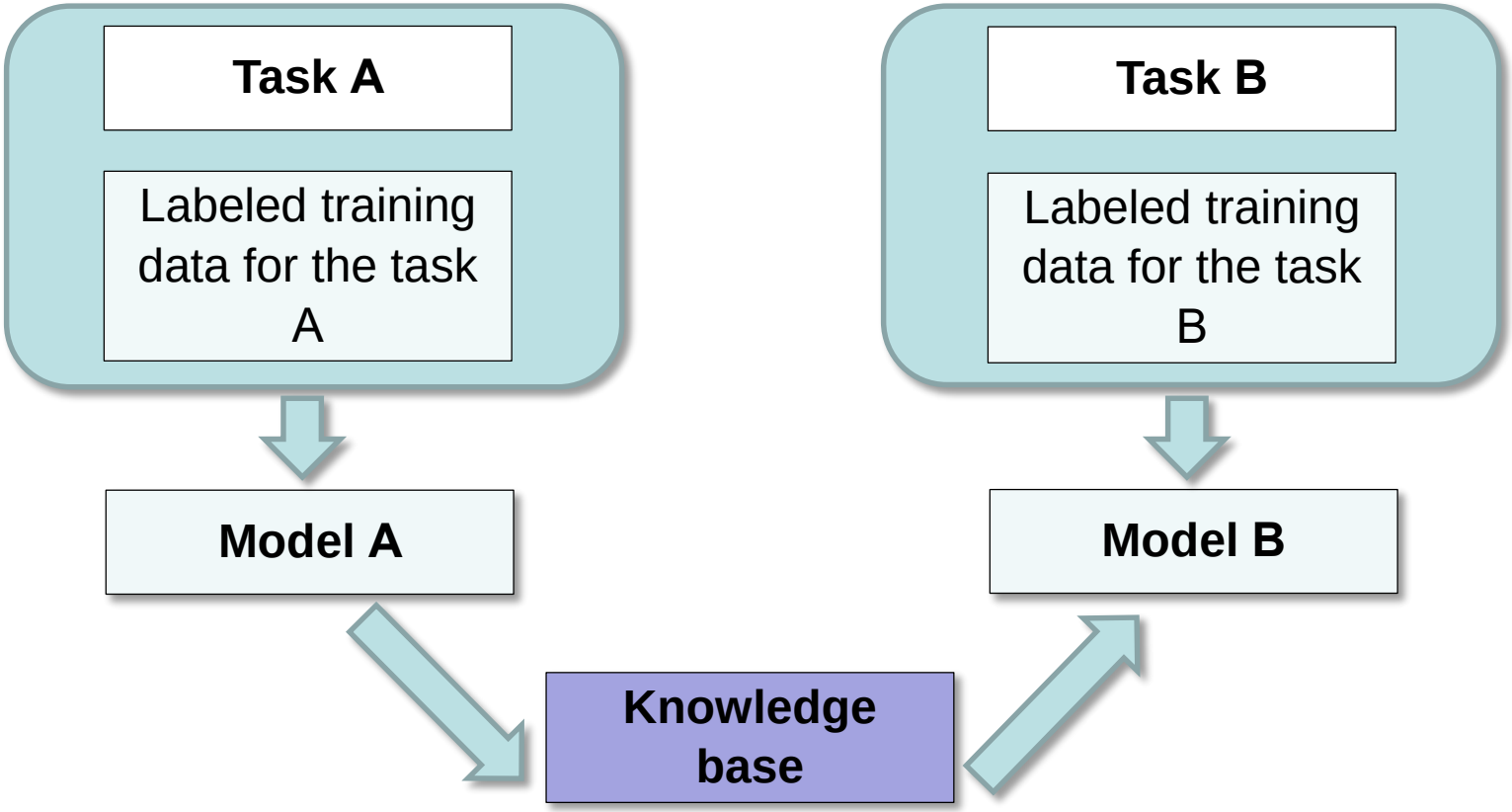
Introduction (6)

- If the original “Task A” and the target “Task B” are related in some way, can you somehow use the knowledge obtained in solving the original problem for the target one?

- Yes, but how?
- Example:
  - The original “Task A” is a pedestrian detection in the daytime
  - The target “Task B” is a cyclist detection
  - The original model can not be directly used to solve the target problem
  - Despite the fact that in both cases the person detection is carried out, the persons’ positions are different

The goal of transfer learning

- ***The goal of transfer learning*** is to accumulate the knowledge necessary to solve the original “Task A”, and use this knowledge base to solve a related target “Task B”



Formalization of the transfer learning goal. Domain

- ***Domain*** is defined by a pair

$$D = \{\mathrm{X}, P(\mathrm{x})\},$$

where X is a feature space,

$P(\mathrm{x})$ is a probability distribution of a random variable

$$\mathrm{x} = \{x_1, x_2, \dots, x_n\} \in \mathrm{X}$$

- Two domains are mismatched $D_1 \neq D_2$, if the feature spaces $\mathrm{X}_1 \neq \mathrm{X}_2$ or distributions $P(\mathrm{x}_1) \neq P(\mathrm{x}_2)$ are different

Formalization of the transfer learning goal. Task

- ***Task*** is defined by a pair

$$T = \{\mathrm{Y}, f(\cdot)\},$$

where Y is a space of labels, $f(\cdot)$ is a prediction function

- The task is not observed in explicit form, it is trained on the training dataset of pairs $(x_i, y_i), x_i \in \mathrm{X}, y_i \in \mathrm{Y}$
- The prediction function is the distribution of the conditional probability that, for a given feature description there is a certain label $f(x) = P(y|x)$
- Two tasks are mismatched $T_1 \neq T_2$ if the label spaces $\mathrm{Y}_1 \neq \mathrm{Y}_2$ or the probability distributions $P(Y_1|\mathrm{x}_1) \neq P(Y_2|\mathrm{x}_2)$ are different

Formalization of the transfer learning goal

- Given the original task T_S in the domain D_S and the target task T_T in the domain D_T, ***the goal of the transfer learning*** is to improve the quality of the prediction function $f_T(\cdot)$ of the target task in the domain D_T using the knowledge obtained by learning the task T_S in the domain D_S, where $D_T \neq D_S$ or $T_T \neq T_s$ (domains or tasks do not match)

Main problems of transfer learning

- ***"What to transfer?"***
  - Assumes the selection of part of the task or domain that should be transferred
  - There are general knowledge for domains that are valid for many domains simultaneously, but there are domain-specific knowledge
- ***"How to transfer?"***
  - If there is information about "what to transfer", the question "how to transfer?" is solved in a natural way
- ***"When to transfer?"***
  - The answer allows us to assess situations when application of the transfer learning is justified
  - If the domains are not related to each other, the transfer learning may be ineffective

Types of environment in the transfer learning implementation (1)

- ***Inductive Transfer Learning*** when tasks are mismatched $T_T \neq T_S$
- Domains can be either the same or different
- To create an objective prediction function, you need to label the data within the target domain

Types of environment in the transfer learning implementation (2)

- Inductive transfer learning is divided in two classes:
  - The presence of labeled training data for the original task
    - Inductive transfer learning = ***multitask learning***
    - The goal of transfer learning is to improve the model quality for the target task
    - The goal of multitask learning is to train the original and target tasks simultaneously
  - Absence of labeled training data in the original problem
    - Inductive learning transfer = ***self-taught learning***
    - The label spaces in the domains of the original and target tasks can be different, which means that there is no possibility of directly using the side information of the original domain

Types of environment in the transfer learning implementation (3)

- ***Transductive Transfer Learning*** when tasks are the same $T_T = T_S$, and the domains are different $D_T \neq D_S$
- There are no labeled training data in the target domain, but there is a large amount of labeled training data in the original domain

- There are two situations:
  - The feature space of the original and target domains are mismatched $\mathrm{X}_S \neq \mathrm{X}_T$
  - The feature space of the original and target domains are the same $\mathrm{X}_S = \mathrm{X}_T$, but the probability distributions are different $P(\mathrm{X}_S) \neq P(\mathrm{X}_T)$

Types of environment in the transfer learning implementation (4)

- ***Unsupervised Transfer Learning*** when tasks are mismatched $T_T \neq T_S$, but the goal is to train tasks in unsupervised manner (clustering, decreasing the dimension of feature space and so on) within the domain of the target task D_T

Types of transfer learning (1)

- ***Instance-based transfer learning***
  - Some parts of the training set from the original domain are used during the training of the target task
- ***Feature representation transfer***
  - The purpose of this transfer learning is to train a “good” feature representation of the target domain
  - The knowledge used for transmission between domains is encoded in the representation of the learning function

Types of transfer learning (2)

- ***Parameter transfer***
  - The source and target tasks share some parameters or a priori distributions of model hyperparameters
  - Knowledge is transferred through tasks
- ***Transfer learning for relational domains***
  - Some relationships between data in the original and target domains are similar
  - In this case, the transferred knowledge is the relationship between the data

Applying different types of transfer learning in different environments

	Inductive transfer learning $T_T \neq T_S$	Transductive transfer learning $T_T = T_S, D_T \neq D_S$	Unsupervised transfer learning $T_T \neq T_S$
Instance-based transfer learning	+	+	
Feature representation transfer	+	+	+
Parameter transfer	+		
Transfer learning for relational domains	+		

* Pan S.J., Yang Q. A Survey on Transfer Learning // IEEE Transactions on Knowledge and Data Engineering. – 2010. – Vol. 22, Issue 10. – P.1345-1359.

APPLICATION OF TRANSFER LEARNING

Application of transfer learning

- The application of transfer learning is reduced to carrying out various types of experiments

- The problem statement:
  - Let us assume that a deep model to solve the original “Task A” has been trained
  - It is necessary to obtain a model for solving the target “Task B”. Assumed that the tasks are related in meaning

Examples

- “Task A” is an image classification problem, “Task B” is a vehicle classification (on images) problem
- “Task A” is an object detection problem, “Task B” is a vehicle detection problem

Experiment types (1)

- ***Direct use of the model trained to solve the original "Task A" for solving the target "Task B"***
  - This scheme allows us to evaluate the quality of the model constructed for solving the general problem, with respect to the particular problem
  - The trained deep model is used without any changes
  - The experiment implements the parameter transfer

Experiment types (2)

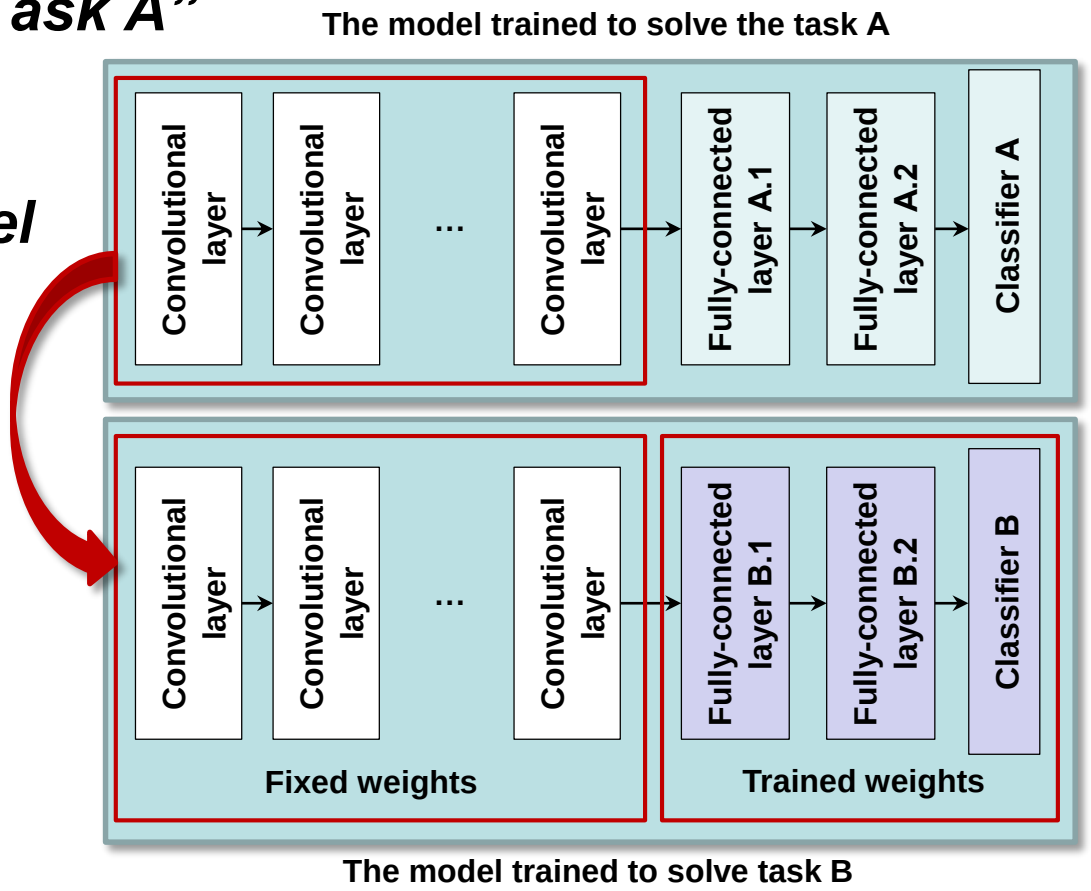
- ***Using the structure of a deep model constructed to solve the original "Task A", with the aim of training a similar model for solving "Task B"***
  - Assumed the model constructed for solving the original problem is trained on the data prepared for the target problem
  - In this case, the model weights are initialized randomly
  - The experiment implements the transfer learning for relational domains

Experiment types (3.1)

- ***Using a model constructed to solve the original "Task A" as a fixed method of feature extraction in constructing a model that solves "Task B"***
  - The idea of this approach is to remove from the deep model the classifier (the last fully-connected layers) and to consider the initial part of the network as a method of feature extraction
  - Instead of the old classifier, you can place a new classifier (for example, another set of fully-connected layers or a support vector machine) and train it on the features constructed using the initial part of the network
  - The experiment implements the feature representation transfer

Experiment types (3.2)

- ***Using a model constructed to solve the original "Task A" as a fixed method of feature extraction in constructing a model that solves "Task B"***

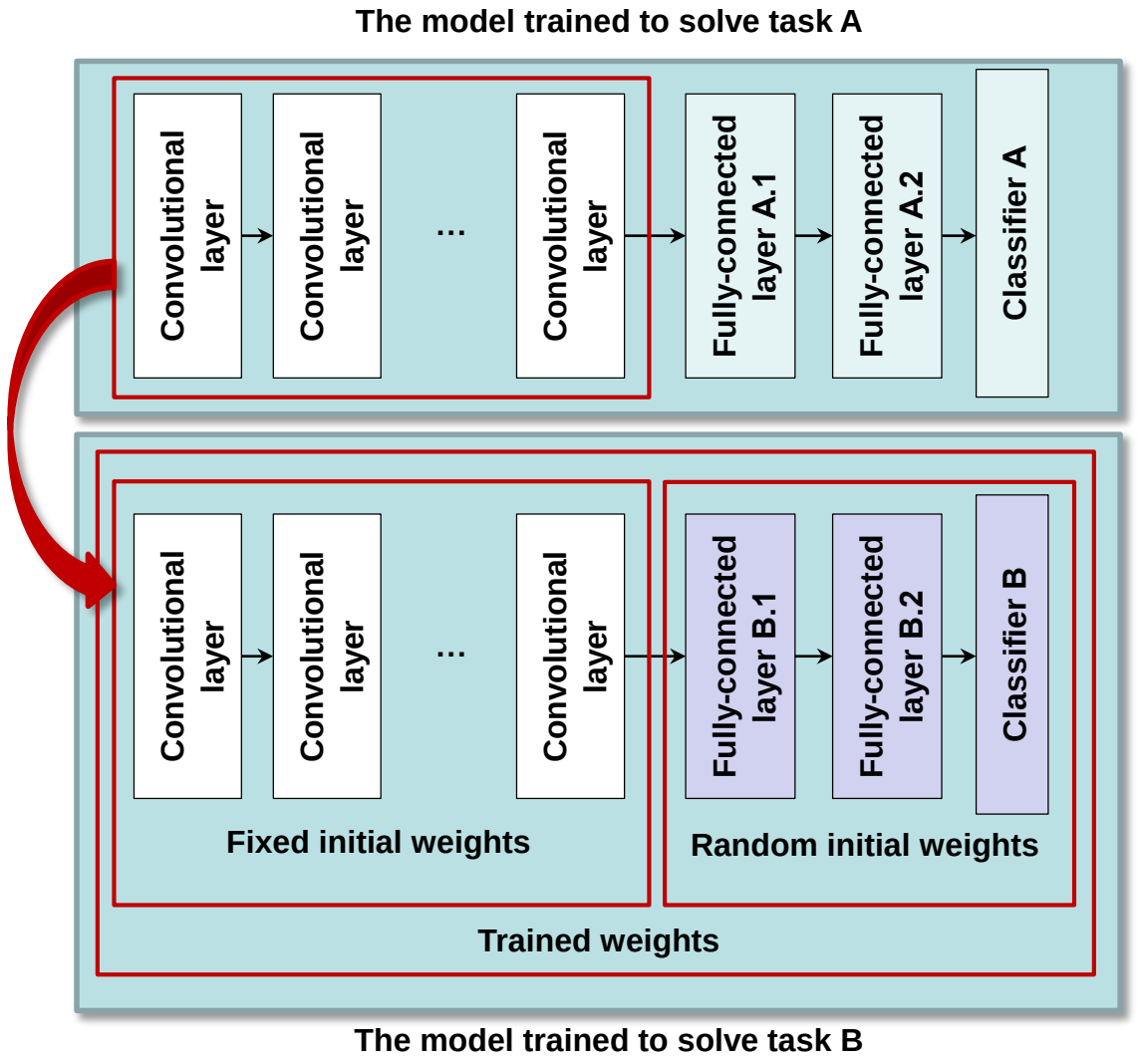


Experiment types (4.1)

- ***Fine-tuning the parameters of the model constructed to solve the original "Task A", with the aim of solving "Task B"***
  - The last layers of the deep model corresponding to the classifier that solves "Task A" are replaced by a new classifier (for example, a set of fully-connected layers with a different number of outputs)
  - The obtained model is trained as a unified system
  - The experiment implements the instance-based transfer learning

Experiment types (4.2)

- ***Fine-tuning the parameters of the model constructed to solve the original "Task A", with the aim of solving "Task B"***



When and what type of transfer learning to use?

- Main factors:
  - Size of the target task training dataset
  - The similarity of the training data for the models of the target and the original tasks

Recommendations on the use of different types of transfer learning (1)

- ***The target dataset is small, and it is similar to the dataset of the original task***

Recommendations on the use of different types of transfer learning (1)

- ***The target dataset is small, and it is similar to the dataset of the original task***
  - Since the dataset is small, it makes no sense to perform a fine-tuning due to the possibility of overfitting
  - The most correct way is to use the initial part of the original network to extract features and to train the simplest classifier on these features

Recommendations on the use of different types of transfer learning (2)

- ***The dataset of the target task is large, and it is similar to the dataset of the original task***

Recommendations on the use of different types of transfer learning (2)

- ***The dataset of the target task is large, and it is similar to the dataset of the original task***
  - You can use fine-tuning the network parameters, while expecting an increase the network performance on the target task

Recommendations on the use of different types of transfer learning (3)

- ***The target dataset is small, and it differs from the original task dataset***

Recommendations on the use of different types of transfer learning (3)

- ***The target dataset is small, and it differs from the original task dataset***
  - The target dataset is small, so you should train the simplest classifier on the features constructed using the earliest layers of the original network
  - The early layers of the network contain more transforms that describe the characteristics of the dataset
  - It make sense the complete network training should be carried out, i.e. training datasets are fundamentally different from each other

Recommendations on the use of different types of transfer learning (4)

- ***The target dataset is large, and it is fundamentally different from the dataset of the original task***

Recommendations on the use of different types of transfer learning (4)

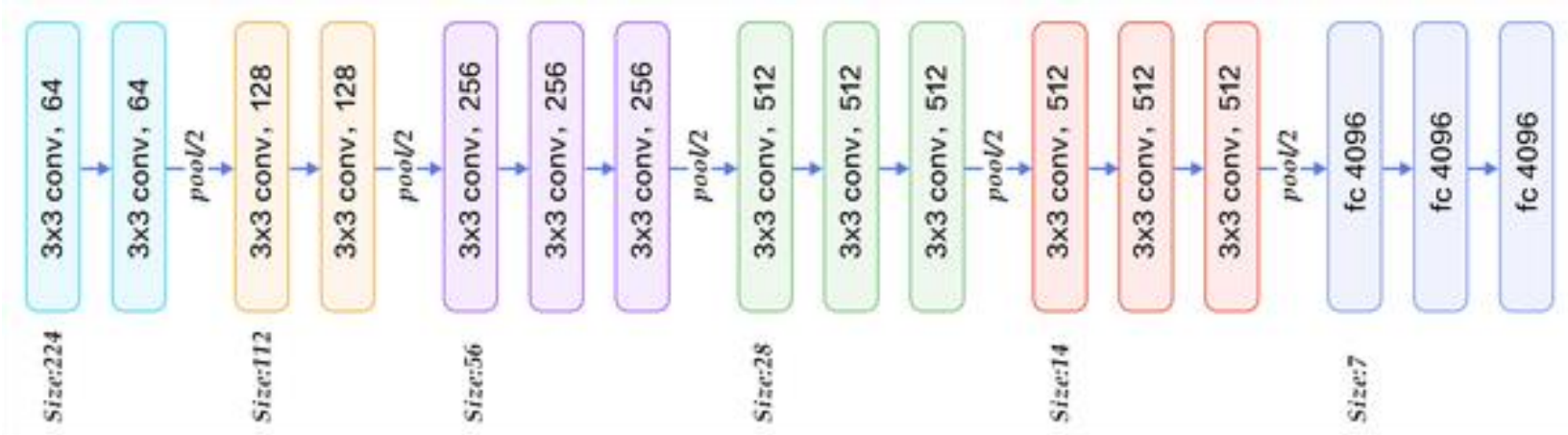
- ***The target dataset is large, and it is fundamentally different from the dataset of the original task***
  - The training dataset for the target task is large enough to train a model whose structure is analogous to the model of the original task
  - In practice, it often turns out that the weight initialization by the values obtained during the training of the original task allows you to more accurately configure the network parameters

Example

TRANSFER LEARNING IN THE INTEL® NEON™ FRAMEWORK

Architecture of transferred neural network

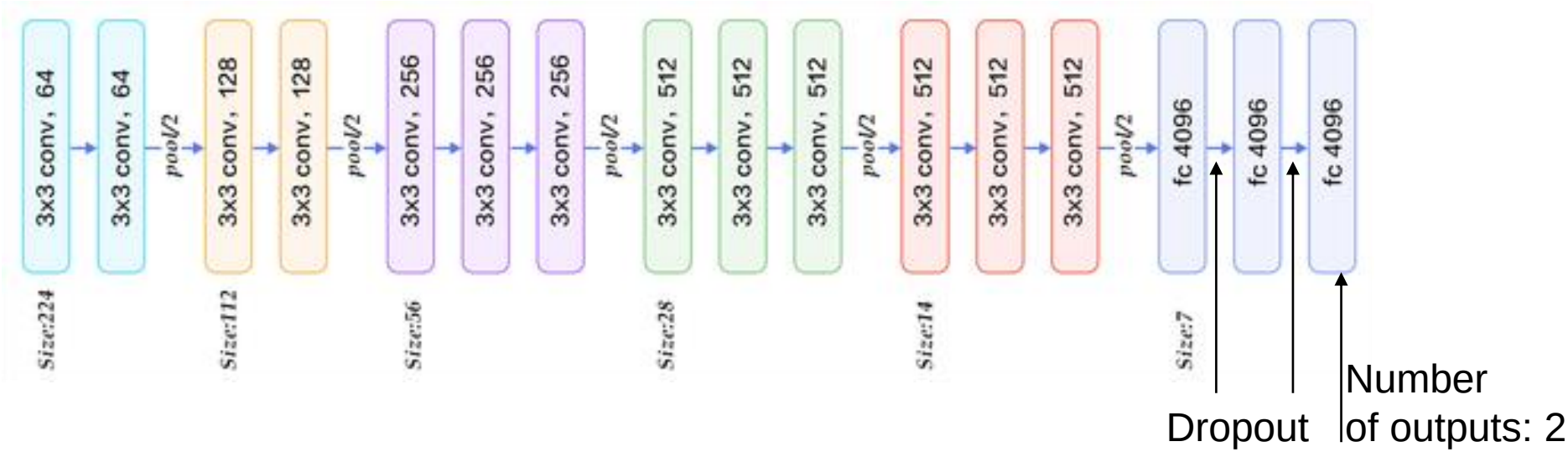
- VGG-16



* Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. – 2015. – [https://arxiv.org/pdf/1409.1556.pdf].

** What is the VGG neural network? [https://www.quora.com/What-is-the-VGG-neural-network].

Replace the classifier in the transferred neural network



* Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. – 2015. – [https://arxiv.org/pdf/1409.1556.pdf].

** What is the VGG neural network? [https://www.quora.com/What-is-the-VGG-neural-network].

Transfer learning in the Intel® neon™ Framework (1)

```
def generate_vgg_16_transfer_model():
    init1 = Xavier(local=True)
    initfc = GlorotUniform()
    relu = Rectlin()
    conv_params = {'init': init1,'strides': 1,'padding': 1}
    # Set up the model layers
    layers = []

    # set up 3x3 conv stacks with different map sizes
    for i, nofm in enumerate([64, 128, 256, 512, 512]):
        i = i + 1
        layers.append(Conv((3, 3, nofm), **conv_params,
                           name='conv%d_1' % i))
```

Transfer learning in the Intel® neon™ Framework (2)

```
layers.append(Bias(init=Constant(0),
                   name='conv%d_1_bias' % i))
layers.append(Activation(Rectlin()))
layers.append(Conv((3, 3, nofm), **conv_params,
                   name='conv%d_2' % i))
layers.append(Bias(init=Constant(0),
                   name='conv%d_2_bias' % i))
layers.append(Activation(Rectlin()))
if nofm > 128:
    layers.append(Conv((3, 3, nofm), **conv_params,
                       name='conv%d_3' % i))
    layers.append(Bias(init=Constant(0),
                       name='conv%d_3_bias' % i))
```

Transfer learning in the Intel® neon™ Framework (3)

```
        layers.append(Activation(Rectlin()))
        layers.append(Pooling(2, strides=2))

    layers.append(Affine(nout=4096, init=initfc,
                         bias=Constant(0), activation=Rectlin(),
                         name='fc6_new'))
    layers.append(Dropout(keep=0.5))
    layers.append(Affine(nout=4096, init=initfc,
                         bias=Constant(0), activation=Rectlin(),
                         name='fc7_new'))
    layers.append(Dropout(keep=0.5))
    layers.append(Affine(nout=2, init=initfc,
                         bias=Constant(0), activation=Softmax(),
                         name='cls_imdb_wiki_face'))
```

Transfer learning in the Intel® neon™ Framework (4)

```
model = Model(layers=layers)
cost = GeneralizedCost(costfunc=CrossEntropyMulti())
return (model, cost)
```

Transfer learning in the Intel® neon™ Framework (5)

```
if (__name__ == '__main__'):
    args = parse_args(__doc__)
    pretrained_model_filepath = args.source_model
    ...
    target_model, cost = target_model_generator()
    pretrained_model = load_obj(pretrained_model_filepath)
    imported_layers =
      import_matching_layers(pretrained_model, target_model)

    # Make optimizer for transfer learning:
    base_lr = 0.001
    default_optimizer = GradientDescentMomentum(
       base_lr, momentum_coef=0.9,
       stochastic_round=args.rounding, wdecay=0.0005)
```

Transfer learning in the Intel® neon™ Framework (6)

```
optimizers_mapping = {'default': default_optimizer}

# Make small learning rates for imported layers
imported_layers_optimizer_params = \
  default_optimizer.get_description().copy()['config']
imported_layers_optimizer_params['learning_rate'] = 0
imported_layers_optimizer = \
   GradientDescentMomentum(
        **imported_layers_optimizer_params)
optimizers_mapping.update({l: imported_layers_optimizer
   for l in imported_layers})

optimizer = MultiOptimizer(optimizers_mapping)
```

Infrastructure

- CPU: Intel® Xeon® CPU E5-2660 0 @ 2.20GHz
- GPU: Tesla K40s 11Gb
- OS: Ubuntu 16.04.4 LTS
- Frameworks:
  - Intel® neon™ Framework 2.6.0
  - CUDA 8.0
  - Python 3.5.2
  - Intel® Math Kernel Library 2017 (Intel® MKL)

Experiments

Network id	Experiment	Training parameters	Accuracy, %	Training time, s
TL-1	Learning the weights of the last layers	num_epochs = 90, base_lr = 0.001, lr = 0	85.6	119975
TL-2	Fine-tuning of all weights	num_epochs = 90, base_lr = 0.001	85.3	119989
TL-3	Complete network training from random initial weights	num_epochs = 30, base_lr = 0.001	**86.3**	39282

Summary results

Network id	Accuracy, %	Training time, s
FCNN-1	71.2	932
FCNN-2	73.5	977
FCNN-3	**77.7**	1013
CNN-1	79.3	1582
CNN-2	**83.5**	2030
ResNet-18 (90 epochs)	**81.3**	15127
ResNet-50 (30 epochs)	80.9	11849
TL-1	85.6	119975
TL-2	85.3	119989
TL-3	**86.3**	39282

Conclusion

- The transfer learning makes it possible to quickly obtain an initial approximation of the problem solution quality
- For many tasks, using a deep model structure demonstrates better quality than fine-tuning the network parameters
- As a rule, time of complete network training from random initial weights is less than fine-tuning parameters

Literature

- Haykin S. Neural Networks: A Comprehensive Foundation. – Prentice Hall PTR Upper Saddle River, NJ, USA. – 1998.
- Osovsky S. Neural networks for information processing. – 2002.
- Goodfellow I., Bengio Y., Courville A. Deep Learning. – MIT Press. – 2016. – [http://www.deeplearningbook.org].

Authors

- **Kustikova Valentina Dmitrievna**
  Phd, lecturer, department of Computer software and supercomputer technologies, Institute of Information Technologies, Mathematics and Mechanics, Nizhny Novgorod State University
  valentina.kustikova@itmm.unn.ru
- **Zhiltsov Maxim Sergeevich**
  master of the 1st year training, Institute of Information Technology, Mathematics and Mechanics, Nizhny Novgorod State University
  zhiltsov.max35@gmail.com
- **Zolotykh Nikolai Yurievich**
  D.Sc., Prof., department of Algebra, geometry and discrete mathematics, Institute of Information Technologies, Mathematics and Mechanics, Nizhny Novgorod State University
  nikolai.zolotykh@itmm.unn.ru