Warsaw University of Technology

FACULTY OF
MATHEMATICS AND INFORMATION SCIENCE

Master's diploma thesis

in the field of study Computer Science
and specialisation Data Science

Transfer learning for time series classification

Paulina Pacyna

student record book number 290600

thesis supervisor
Agnieszka Jastrzębska, PhD

WARSAW 2023

Abstract

Transfer learning for time series classification

The task of classifying time series is an important problem in the field of data mining. Time series occur every time we want to measure some phenomenon that changes over time. A time series can describe for example the amplitude of a heartbeat sound, stock prices or hand movement along an axis when serving in tennis. Such time series can express different characteristics of the phenomenon. Those characteristics are called *classes*. For example, the heartbeat sound amplitude can represent (belong to) one of two classes: *healthy* and *unhealthy*. Time series classification attempts to learn the distinctive features of a time series and build a model that can distinguish between those classes.

The time series classification problem was initially solved using classical algorithms such as the k-nearest neighbor classifier with distance measures suited for time series, like Dynamic Time Warping. Still, the advantages of using deep learning algorithms in the context of time series classification have recently begun to be recognized. Neural networks are capable of detecting shapes that distinguish a class or understanding ordered temporal relationships.

Transfer learning practically is used when there is limited data to train. Transfer learning attempts to apply patterns learned from one dataset to improve learning when creating a model for another dataset. A common practice is to prepare a source classifier trained on a large, easily available amount of data for one task and then use this model or parts of it for a detailed task with a smaller amount of data. Models trained using this method often have a shorter training time, faster accuracy increase, and can generalize more easily on the test set.

Keywords: time series, classification, transfer learning, deep learning, ...

Streszczenie

Zastosowanie techniki transfer learning w zadaniu klasyfikacji szeregów czasowych

Zadanie klasyfikacji szeregów czasowych jest ważnym problemem w dziedzinie eksploracji danych. Szeregi czasowe występują za każdym razem, gdy chcemy zmierzyć jakieś zjawisko, które zmienia się w czasie. Szereg czasowy może opisywać np. amplitudę dźwięku bicia serca, ceny akcji czy ruch ręki wzdłuż osi podczas serwu w tenisie. Takie szeregi czasowe mogą wyrażać różne cechy zjawiska. Te cechy nazywane są *klasami*. Na przykład amplituda dźwięku bicia serca może reprezentować (należeć do) jednej z dwóch klas: *norma* i *choroba*. Klasyfikacja szeregów czasowych polega na rozpoznawaniu charakterystycznych cech szeregu czasowego i budowanie modelu, który potrafi rozróżniać klasy.

Problem klasyfikacji szeregów czasowych był początkowo rozwiązywany za pomocą klasycznych algorytmów, takich jak algorytm k-najbliższych sąsiadów w połączeniu z miarami podobieństwa dla szeregów, jak odległość DTW. W ostatnim czasie zaczęto dostrzegać zalety stosowania algorytmów głębokiego uczenia w kontekście klasyfikacji szeregów czasowych. Sieci neuronowe są w stanie wykryć kształty wyróżniające daną klasę lub zrozumieć relacje między obserwacjami w czasie.

Metoda transfer learning jest stosowane w przypadku ograniczonej ilości danych do tranowania modelu. Technika transfer learning próbuje zastosować wzorce wyuczone z jednego zbioru danych, aby poprawić uczenie podczas tworzenia modelu dla innego zbioru danych. Częstą praktyką jest przygotowanie źródłowego klasyfikatora wytrenowanego na dużej, łatwo dostępnej ilości danych dla jednego zadania, a następnie wykorzystanie tego modelu lub jego części do szczegółowego zadania z mniejszą ilością danych. Modele wytrenowane tą metodą często mają krótszy czas trenowania, szybszy wzrost dokładności i lepsze, ogólniejsze wyniki na zbiorze testowym.

Słowa kluczowe: szeregi czasowe, klasyfikacja, transfer learning ...

Contents

Introduction

Time series classification was initially approached using classical machine learning algorithms. The paper [1] categorizes the commonly used algorithms into several categories. The first category are time domain distance based algorithms. Those algorithms use various distance measures adjusted for time series domain to capture similarity between pairs of time series. The distance measure is then combined with a distance-based classifier. A flagship example of an algorithm belonging to this class of algorithms is Dynamic Time Warping distance with k-Nearest Neighbour classifier, often used as a benchmark classifier. Another category of classifiers mentioned in [1] are dictionary, shapelet and interval based classifiers. All of them try to extract features distinctive for the class of the time series. The dictionary based classifiers encode the time series into a dictionary of *words* representing the time series. Shapelet based algorithms focus on subseries of the time series that are discriminatory of class membership. Interval based algorithms extract features from selected intevals of the time series. With the rise of popularity of deep learning algorithms researchers made an attempt to replace the former, hand-extracted features with deep learning classifiers. A few years after the article [3] was published. It contains a review of deep learning algorithms applied to the task of time series classification. The usage of deep learning algorithms enabled the possibility to utilize transfer learning.

Transfer learning is widely used in image recognition and natural language processing. The authors of paper [3] extended their previous finding by a study on application of transfer learning. The authors examine knowledge transerability on 85 dataset from UCR archive, by pre-training a model on one dataset and fine-tuning on another. The authors examine if the accuracy improved for all pairs of datasets in UCR archive.

Transfer learning for deep learning is usually done by training the network on a big, diverse dataset and utilizing first layers of the network when training on another dataset. In image recognition or natural language processing it is common to pre-train the source network on ImageNet dataset or Wikipedia dataset respectively. Such diverse, dataset with labels does not exists for time series. In this thesis, we will attempt to create such dataset from 85 smaller datasets available on the UCR archive. We will try to mimic good properties of a transfer learning source dataset by preprocessing, upsampling and augmenting the dataset. We will also

study it transfer learning on a diverse helps to generalize to new training data and improve the training process on the target dataset. Similarly as in [4], we will experiment to find which features of the source dataset (classes diversity, dataset size, augmentation, preprocessing) are fundamental for the transfer learning process.

1. Related works

In this chapter, we describe several algorithms used in time series classification. We will also recall theoretical definitions used to describe transfer learning.

1.1. Time series classification

A time series is an ordered collection of observations indexed by time.

$$X = (x_t)_{t \in T} = (x_1, ..., x_T),\ x_t \in \mathbb{R}$$

The time index T can represent any collection with the natural order. We assume that indices are spaced evenly in the set T. The realization or observation x_t in the times series is a numerical value describing the phenomena we observe, for example, the amplitude of a sound, stock price, or y-coordinate. Time series classification is a problem of finding the optimal mapping between a set of time series and corresponding classes.

1.1.1. Dynamic Time Warping with k-Nearest Neighbour

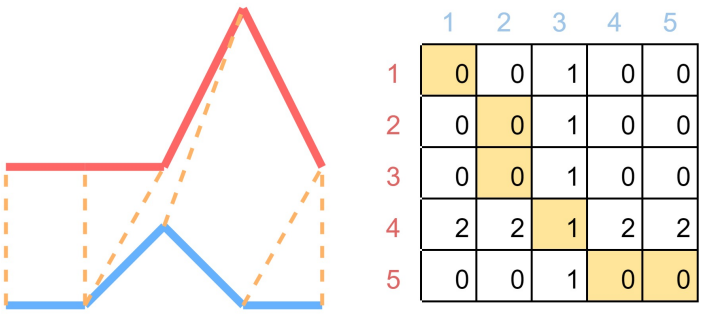
The Dynamic Time Warping [1] with k-Nearest Neighbour classifier uses a distance-based algorithm with a specific distance measure. A DWT distance between time series X^1, X^2 of equal lengths is defined as follows:

$$DTW(X^1, X^2) = \min\{\sum_{i=1}^{S} dist(x^1_{e_i}, x^2_{f_i}) : (e_i)_{i=1}^S, (f_i)_{i=1}^S \in 2^T\}$$

subject to:

- $e_1 = 1, f_1 = 1, e_S = N, f_S = N$
- $|e_{i+1} - e_i| \leq 1$, $|f_{i+1} - f_i| \leq 1$

The measure defined above, used in the k-Nearest Neighbour classifier, is often used as a benchmark classifier. The list of indices $[(e_1, f_1), \ldots, (e_S, f_S)]$ is called the warping path. An illustration of this measure is displayed on figure 1.1



	1	2	3	4	5
1	0	0	1	0	0
2	0	0	1	0	0
3	0	0	1	0	0
4	2	2	1	2	2
5	0	0	1	0	0

Figure 1.1: The Dynamic Time warping distance between the series above is equal to 1. We show the warping path in the distance matrix and connections indicated by the dynamic warping path.

1.1.2. Multi Layer Perceptron

The Multi Layer Perceptron (MLP) is the first artificial neural network architecture proposed in [3] and can be used for time series classification task. Formally, the MLP network can be defined as a composition of *layer* functions. The network returns a vector that usually represents the probability distribution over the set of classes.

$$MLP(X; \theta_1, \ldots, \theta_M, \beta_1, \ldots, \beta_M) = L_M(\ldots L_2(L_1(X; \theta_1, \beta_1); \theta_2, \beta_2); \theta_M, \beta_M)$$

Each layer $L_i : \mathbb{R}^M \to \mathbb{R}^N$ is a function that depends on the parameters $\theta \in r^{M \times N}, \beta \in \mathbb{R}^N$

$$L_i(X; \theta_i, \beta_i) = f_i(X\theta_i + \beta_i)$$

The function $f_i : \mathbb{R}^N \to \mathbb{R}^N$ is an arbitrarily chosen non-linear function. The number of layers and dimension of weights in hidden layers are also arbitrary. The weights in the first and last layer have to match the dimensionality of input data (e.g. the length of the time series) and the number of classes. The output of the last layer is interpreted as a probability distribution over the set of classes.

The disadvantage of using Multi Layer Perceptrons for time series classification is that the input size is fixed. All time series is the training data must have the same length. In transfer learning, this means that if we want to reuse the source network (or a set of first layers from the network), the time series in the target dataset must have the same length as in the source dataset.

The MLP architecture fails at understanding the temporal dependencies [3]. Each input value in the time series is treated separately because it is multiplied by a separate row in the weight matrix.

1.1.3. Convolutional Neural Networks

Convolutional Neural Networks are widely used in image recognition. A convolution applied for a time series can be interpreted as sliding a filter over the time series. A convolutional layer is a set of functions called convolutions or filters. The filter is applied at a given point, taking into account the values that surround the point.

To define the convolution operation, let's assume the input is a matrix $X \in \mathbb{R}^{(N_1,\ldots,N_K)}$. In the case of images, the number of dimensions K is often equal to 3 (height, width, channels), for univariate time series we can assume just one dimension, and for multivariate time series we need two dimensions - (feature, time). The filter consists of a matrix of weights $M \in \mathbb{R}^{(P_1,\ldots,P_K)}$. Usually, P_l are odd numbers, so that we can index the matrix with symmetrical numbers: $(\frac{-P_l+1}{2}, \frac{-P_l+3}{2}, \ldots, 0, \ldots, \frac{P_l-1}{2})$. The 0 index marks the center of the matrix.

Finally the convolution $*$ is defined as follows:

$$(X * M)_{i_1,\ldots,i_K} = \sum_{l_1=\frac{-P_1+1}{2}}^{\frac{P_1-1}{2}} \cdots \sum_{l_K=\frac{-P_K+1}{2}}^{\frac{P_K-1}{2}} M_{l_1,\ldots,l_K} X_{i_1+l_1,\ldots,i_K+l_K}$$

The result of the convolution is passed elementwise to a nonlinear function. The nonlinear function together with the convolution operation will be called a filter.

In the case of univariate time series, the first layer of the convolutional neural network is one-dimensional. The output of the first layer has dimensions (length of time series - the length of the filter + 1, number of filters). Below we define the value of the output for filter i

$$y_{t,i} = f_i([\theta^i_{\frac{-M+1}{2}}, \ldots, \theta^i_{\frac{M-1}{2}}] \cdot [X_{t_+\frac{-M+1}{2}}, \ldots, X_{t+\frac{M-1}{2}}]),$$

where $\cdot$ is a dot product and $t \in T$ is the time index.

The weights θ^i are different for each filter. The same filter is applied over the whole length of the time series. This is called *weight sharing* and it enables the network to learn patterns regardless of the position in the time series.

The architecture of the convolutional layer is not dependent on the size of the input data. Regardless of the size of the input data, the number of filters and size of filters remain the same, only the output sizes depend on the input size. Therefore, if the convolutional layer is succeeded by layers with the same property, like other convolutional layers or Global Pooling with Dense Layer (see section 1.1.4), the whole network may be invariant to the input sizes [3]. Such networks may be interesting in terms of transfer learning, as the sizes of time series in the source task and in the target task do not have to match.

1.1.4. Fully Convolutional Networks

Fully Convolutional Networks are convolutional networks used in time series classification. A sample architecture of a Fully Convolutional Network proposed is in [3]. The first layers in the network are 3 blocks of convolutional layers with ReLU activation function followed by batch normalization layers. The output of the last block is passed to a global pooling layer. The global pooling layer averages the output through the time axis, resulting in a vector of length equal to the number of feature maps in the last convolutional layer. The averaged vector is passed to a block of 2 fully connected layers. Figure 1.2 shows a visualisation of the network.

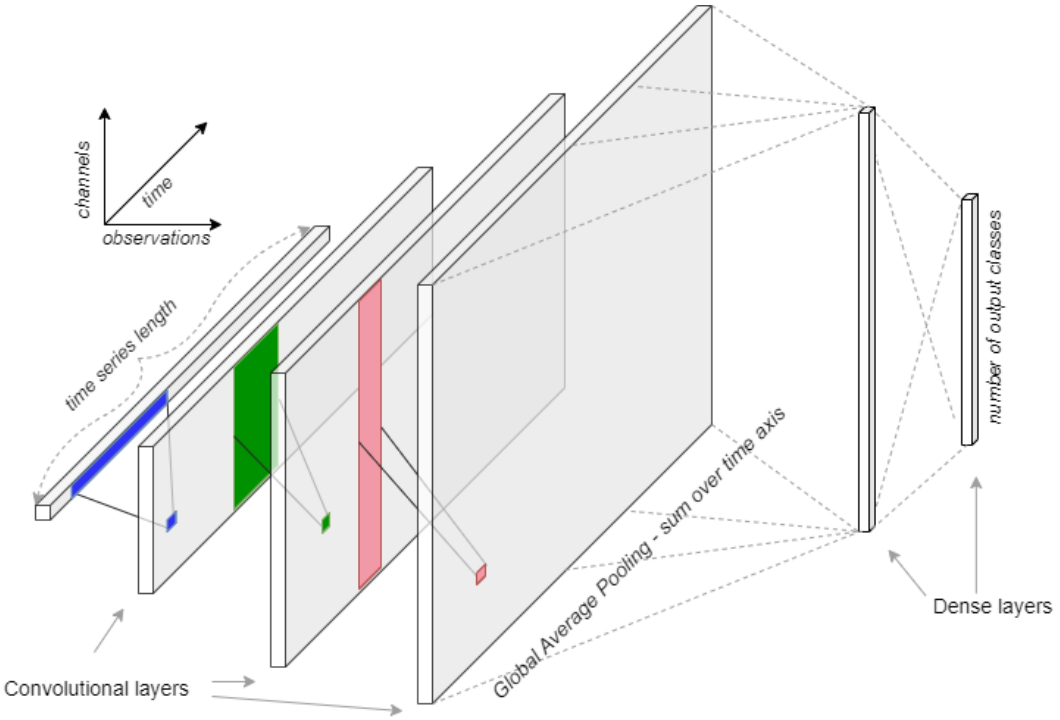


Figure 1.2: Architecture of a Fully Convolutional Network. Source: [3]

Because the architecture of convolutional layers does not depend on the size of input data and the convolutional layers are followed by pooling over the time axis, the whole network is capable of processing data of variable lengths.

1.1.5. Residual Network

The Residual Network was first proposed in [**?**]. The Residual Network is a relatively deep architecture compared to other neural networks used for time series classification.

A residual connection addresses the vanishing gradient problem occuring in networks composed of many layers [**?**]. The vanishing gradient occurs in the backpropagation process. When

the gradient is passes from the last layer to the first, it may be decreasing towards zero. This causes the first layers of the network to learn slowly. The residual connection between layers passes the input directly from one layer to another, skipping a few layers. This way if the network is struggling with vanishing gradient, this shortcut connection may help the network to converge.

The architecture of the Residual Network is conceptually similar to the FCN network. Instead of three convolutional layers, the Residual Network begins with three blocks of convolutional layers, connected with residual connections. Each block consists of three convolutional layers. The three blocks, which consist of nine convolutional layers and three residual connections in total, are followed by a Global Average Pooling layer and a Dense layer. The networks' diagram is shown below (Figure 1.3).

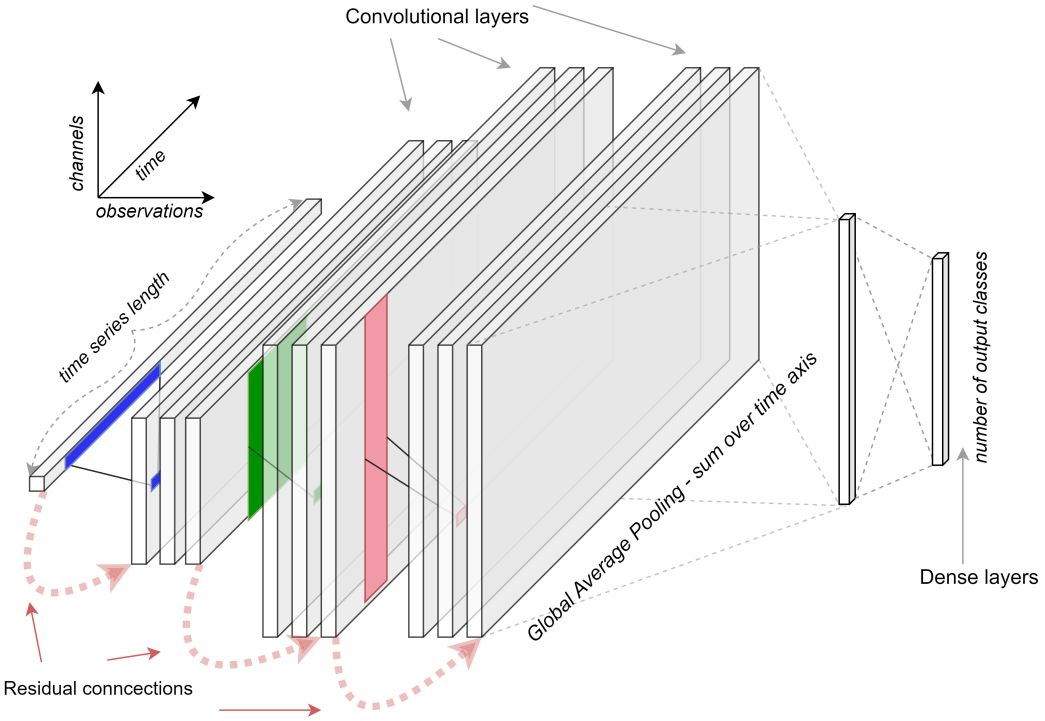


Figure 1.3: Architecture of a Residual Network. Source: [3]

1.1.6. Encoder Network

The Encoder Network is a deep convolutional network proposed by [5]. The first layers of the network are similar to the FCN architecture. The network consists of three convolutional layer followed by an attention layer instead of the Global Average Pooling layer. Another novelty introduced in this network comparing with the FCN architecture is adding a DropOut leayer and replacing the ReLU activation function with PReLU (Parametrized ReLU, thus adding another parameter to the network. The network ends with a Dense layer predicting the distribution over

the classes. The architecture is shown below (Figure 1.4).

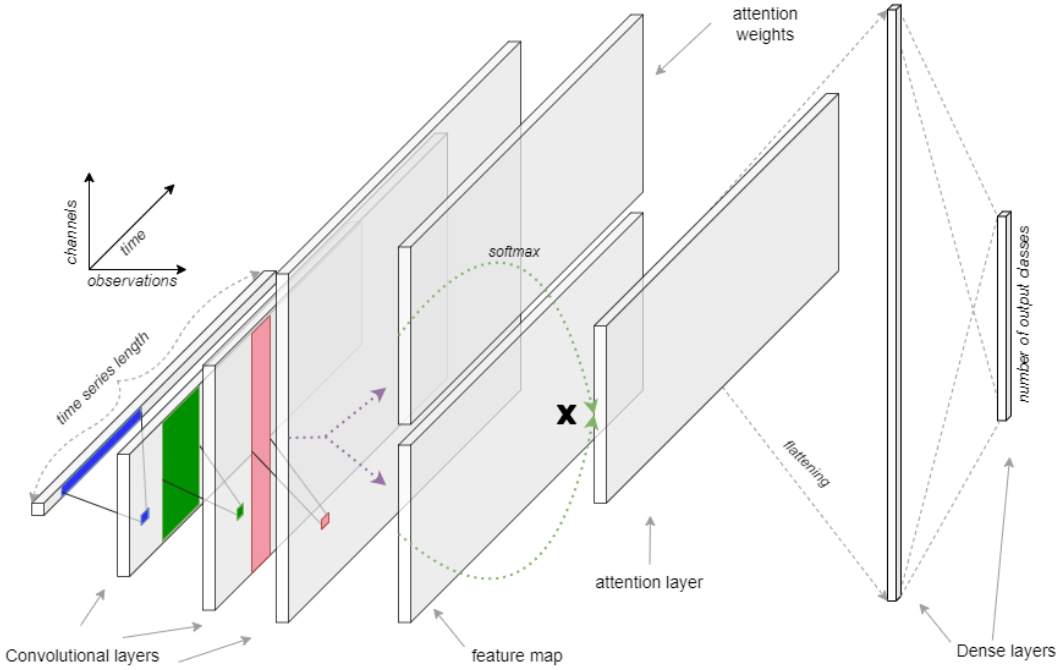


Figure 1.4: Architecture of an Encoder Network. Source: [3]

The attention layer, first proposed in [2], assigns weights to each element of the input representing the importance for the prediciton. The weights representing the attention are normalized using softmax function and then applied on the input data. The weigths are learned by the network. The outputs of the attention layer can be interpreted as a universal representation of the input time series, that will be able to adapt to and represent unseen data [5].

As opposed to the FCN network architecture, the Encoder network is not invariant to the input size. The Dense layer parameters depend on the output size of the attention layer and implicilitely on the output size of convolutional layers. Similarly as former architecture, the convolutional layers enable weight sharing thus learning shapelets independently of their position in the time series.

The authors decribe two approaches of training the network: an end-to-end approach and a transfer learning approach.

1.2. Transfer learning

Transfer learning is a technique that attempts to apply knowledge learned while solving one task to enhance the learning process for another task. Formally, the problem can be described using the notions of tasks and domains [7, 8]. A *Domain* is a pair $\mathcal{D} = (\mathcal{X}, P(\mathcal{X}))$, where $\mathcal{X}$

is the feature space (e.g. the time series observations, and $P(\mathcal{X})$ is the probability distribution over the feature space. A *Task* is a pair of label space $\mathcal{Y}$ and the decision function f, $\mathcal{T} = (\mathcal{Y}, f)$. The decision function f is learned from $\mathcal{X}, P(\mathcal{X}), \mathcal{Y}$ in the learning process.

Transfer learning attempts to utilize knowledge domain/domains and task/tasks. Formally, given $S \in \mathbb{N}$ source domains and source tasks $(\{(\mathcal{D}_i^S, \mathcal{T}_i^S) : i = 1, \ldots, S\})$ and $T \in \mathbb{N}$ target domains and target tasks $(\{(\mathcal{D}_i^T, \mathcal{T}_i^T) : i = 1, \ldots, S\})$ transfer learning utilizes knowledge learned from source domains and tasks to improve the learning process of decision functions in target tasks $\mathcal{T}_i^T$

1.2.1. Transfer learning categorization

Transfer learning can be categorized from different points of view [8]. One of the ways in which we can divide transfer learning algorithms is based on the availability of the labels:

- **inductive** transfer learning - labels are available for both source and target datasets
- **transductive** transfer learning - labels are available only in the domain dataset
- **unsupervised** transfer learning - no labels available in either dataset

Another way of dividing transfer learning methods is by comparing the distribution of feature space and labels. If the source and training dataset consists of the same features, belonging to a similar distribution $(\mathcal{D}^{\mathcal{S}} = (\mathcal{X}, \mathcal{P}(\mathcal{X}))^{\mathcal{S}} = (\mathcal{X}, \mathcal{P}(\mathcal{X}))^{\mathcal{T}} = \mathcal{D}^{\mathcal{T}}))$ and the label spaces are equal $(\mathcal{Y}^{\mathcal{S}} = \mathcal{Y}^{\mathcal{T}})$, then the task can be described as homogeneous transfer learning. If at least one of the former assumptions does not hold, the transfer learning setting is called heterogeneous.

Transfer learning can be categorized based on the algorithm/approach used. There are four main categories [6] used in deep learning:

- **instance** based transfer learning - target dataset is enriched by instance from the source dataset belonging to the target distribution.
- **mapping** based transfer learning - source and target dataset are mapped to one domain. The distribution is adjusted in both datasets.
- **network** based transfer learning - target classifier uses parts of the network from the source classifier f.
- **adversarial** based transfer learning - an adversarial network tries to distinguish between the two datasets. If the network fails at this task, then it means that the datasets are similar and the netowork extracted features similar between source and target datasets.

1.2.2. Characteristics of a good source domain

In the field of image processing, it is very common to use convolutional neural networks pre-trained with the ImageNet dataset [4]. ImageNet is a large dataset of human-annotated images. It contains 1 million labeled images of 1000 classes. The label space consists of fine-grained classes such as breeds of dogs and cats, but also coarse-grained classes like *red wine* and *traffic light*. Example pictures from this dataset are shown below Às transfer learning based on this dataset became more popular and successful, a question arose: Which features of this dataset make it so good for this task?

Figure 1.5: Sample images from the ImageNet dataset, with examples of very similar (fine-grained) classes like two birds of different breeds and coarse-grained classes, like a lamp and a bird. Source: url

A study conducted in [4] attempts to answer this question. The first hypothesis is that the volume of the dataset is relevant to train accurate, general classifiers. The authors compared models pretrained on the original dataset and models based on sampled subsets (reduced 2, 4 8 and 20 times). The results show that the more training examples, the better results. The accuracy of the initial classifier occurred to be more dependent on the size of dataset than the accuracy of classifiers fine-tuned from the base classifier. It is natural, that the base classifier's accuracy will depend on the dataset size. Still, the classifier fine-tuned from the base classifier seems to cope with the reduced dataset and the impact on the accuracy is less detrimental. This is visible on the figure 1.6

Next experiments examine the label space. The authors study if the granularity of the label space is essential for the problem. To compare the results, the label space is clustered and 127 classes are derived from the initial 1000 classes. Pre-training with the reduced label space has a minimal negative impact on the accuracy of classifiers fine-tuned from this classifier. This suggests that such a fine division may not be needed.

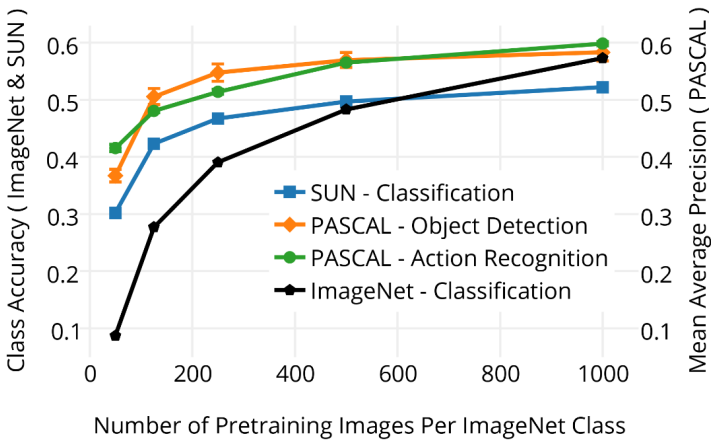


Figure 1.6: Accuracy of the base classifier (black) and classifiers fine-tuned from the base classifier. Image source: [4]

Finally, the last question is if we train the classifier on the reduced label space with 127 classes, will it be able to distinguish between the fine-grained classes? To examine that, the authors extracted features from the first layers of the networks trained on reduced label space. Then, the authors performed classification with 1-NN and 5-NN models on the extracted feature space, but with 1000 classes. The findings are that the k-NN classifier performs 15% worse on reduced dataset vs normal dataset. This shows that CNNs are capable of implicitly learning representative features distinctive between similar classes even when trained on coarse-grained classes.

While the article [4] does not conclude which single feature of ImageNet dataset makes it so efficient as a source dataset for transfer learning, it is clear that all properties of this dataset are important for the accuracy of the classifier.

Bibliography

[1] Anthony Bagnall, Aaron Bostrom, James Large, and Jason Lines. *The Great Time Series Classification Bake Off: An Experimental Evaluation of Recently Proposed Algorithms. Extended Version.* arXiv, 2016.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014.

[3] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, mar 2019.

[4] Minyoung Huh, Pulkit Agrawal, and Alexei A. Efros. *What makes ImageNet good for transfer learning?* arXiv, 2016.

[5] Joan Serrà, Santiago Pascual, and Alexandros Karatzoglou. Towards a universal neural network encoder for time series, 2018.

[6] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning, 2018.

[7] Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang. *A survey of transfer learning.* Journal of Big Data, 2016.

[8] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. *A Comprehensive Survey on Transfer Learning.* arXiv, 2019.