Deep Learning for Fixed Model Reuse[*]

Yang Yang, De-Chuan Zhan, Ying Fan, Yuan Jiang and Zhi-Hua Zhou
National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, 210023, China
Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing, 210023, China
{yangy, zhandc, fany, jiangy, zhouzh}@lamda.nju.edu.cn

Abstract

Model reuse attempts to construct a model by utilizing existing available models, mostly trained for other tasks, rather than building a model from scratch. It is helpful to reduce the time cost, data amount, and expertise required. Deep learning has achieved great success in various tasks involving images, voices and videos. There are several studies have the sense of model reuse, by trying to reuse pre-trained deep networks architectures or deep model features to train a new deep model. They, however, neglect the fact that there are many other fixed models or features available. In this paper, we propose a more thorough model reuse scheme, FMR (Fixed Model Reuse). FMR utilizes the learning power of deep models to implicitly grab the useful discriminative information from fixed model/features that have been widely used in general tasks. We firstly arrange the convolution layers of a deep network and the provided fixed model/features in parallel, fully connecting to the output layer nodes. Then, the dependencies between the output layer nodes and the fixed model/features are *knockdown* such that only the raw feature inputs are needed when the model is being used for testing, though the helpful information in the fixed model/features have already been incorporated into the model. On one hand, by the FMR scheme, the required amount of training data can be significantly reduced because of the reuse of fixed model/features. On the other hand, the fixed model/features are not explicitly used in testing, and thus, the scheme can be quite useful in applications where the fixed model/features are protected by patents or commercial secrets. Experiments on five real-world datasets validate the effectiveness of FMR compared with state-of-the-art deep methods.

Introduction

Machine learning methods including deep learning techniques have achieved great success in many fields. However, there are some obvious deficiencies, e.g., lots of demands on training consumptions including both computational expenses and labeled examples. Besides, lacking of adaptability in current learning techniques also narrows the range of application for learning techniques.

Model reuse (Zhou 2016) attempts to construct a model by utilizing existing available models, mostly trained for other tasks, rather than building a model from scratch. This offers a great potential to reduce the required amount of training examples and training time cost, because the exploitation of existing models may help set a good basis for the training of a new model. An example has been shown in (Li, Tsang, and Zhou 2013), where a model aims to optimizing AUC can be obtained with much less effort by reusing a model optimizing accuracy. Note that model reuse also reduces the requirement of expertise in training the models, because the user can start from a good model generated by experts, and thus, an expert-level new model can be obtained though the user him/herself is not an expert.

In the deep learning community, there are several pieces of studies trying to reuse the convolution layers in deep structures, e.g., by initializing a network with weights from pre-trained networks (Yosinski et al. 2014); by proposing a new network architecture to transfer features from pre-trained networks (Long and Wang 2015). These models are generally based on the strategies of re-training on dataset B with trained deep networks on dataset A, i.e., they mainly focus on the transfer of information of the latent weights in deep networks, neglecting the existence of fixed models which might not be deep models, and fixed features which might not be deep features.

In this paper, we propose the FMR (Fixed Model Reuse) approach. This is a more thorough model reuse scheme, which arranges the convolution layers and the model/features used in general tasks in parallel, and fully connecting to the output layer nodes. A *knockdown* strategy is also proposed for reducing the dependencies between the "fixed" model/features and the output layer nodes gradually and eventually makes the whole model independent to the original fixed model/features. As a consequence, only the raw features are required in testing phase, though the helpful information in the fixed model/features have already been incorporated into the deep structure.

The rest of this paper starts from introduction of related work. Then we propose our approach, followed by experiments and conclusion.

Related Work

Deep networks are able to learn nonlinear feature representations that nest high-level abstractions behind data and have achieved great success in many scenarios (Tian et al. 2014;

[*]This work was supported by NSFC (61333014, 61673201).

Kang, Li, and Tao 2016). There have been many researches on deep learning methods, e.g., Karpathy et al. (2014) studied multiple approaches that extends the connectivity of a CNN in time domain to take advantage of local spatiotemporal information; Wang et al. (2015) proposed the DCCAE (deep canonically correlated autoencoders) for multiple modal representation learning; Srivastava et al. (2014) randomly dropped units (along with their connections) from the neural network during training to deal with overfitting. However, these deep methods always need large volume of labeled training examples and expensive training processes.

Reusability has been emphasized by (Zhou 2016) as a crucial characteristics of the new concept of *learnware*. It would be ideal if models can be reused in scenarios that are very different from their original training scenarios. This is of course a big challenge, whereas reusing models in relatively similar scenarios have already been demonstrated well useful. Li, Tsang, and Zhou (2013) has shown that by starting from a trained model optimizing accuracy, it is easier to construct a model optimizing AUC. Transfer learning (Pan and Yang 2010) also provides possible techniques to model reuse, by emphasizing that there must exist some bridge connecting a source domain and target domain, though they generally do not explicitly reuse an existing model.

There are also some deep transfer learning approaches (Yosinski et al. 2014), e.g., hidden layers trained for fitting multiple domains (Ajakan et al. 2014); Maximum Mean Discrepancy measure incorporated as a regularization to reduce the distribution mismatch in the latent space (Ghifary, Kleijn, and Zhang 2014). However, these deep models always reuse the net structure and weights directly from source networks and can hardly utilize the existing pre-provided model/features.

In this paper, we propose a complete novel model reuse technique with deep structures, which directly substitute the sophisticated fixed model/features used in general tasks with a deep network rather than transferring with pre-trained weights or learning with source/target examples. A *knockdown* technique is developed for achieving such a purpose. The new operator *knockdown* is used for eliminating the collections between the sophisticated model/features and the deep structure, and it seemed similar to dropout yet is completely different in purpose and effects.

Dropout (Srivastava et al. 2014) is proposed to reduce the overfitting problem by randomly setting hidden unit activities to zero during training a deep network. The performance of deep networks on various tasks can be significantly improved with dropout technique and Gao and Zhou (2016) theoretically showed that dropout is able to exponentially reduce Rademacher complexity in deep neural networks. Nevertheless, different to the connections can be reset and updated with another trial of iteration in dropout, our *knockdown* strategy vanishes the dependencies between the output layer nodes and the fixed model/features. Once the dependencies is disconnected by a *knockdown* operation, the related features will not be functional forever. Thus, eventually after all connections of provided model/features are vanished, the discriminative abilities of the pre-obtained fixed model can be imported into the deep network. Consequently, users are only required to input the raw features instead of extracting those sophisticated features during the test phase. the scheme can be quite useful in applications where the fixed model/features are protected by patents or commercial secrets.

It is notable that biologists find that some early low-level neural reflections of healthy human babies will disappear when they grow up, only very limited parts of low-level neural reflections, such as "knee jerk reaction" are kept partly (Sembulingam and Sembulingam 2012). These corresponding functions are taken over or replaced by the center neural system, i.e., *human brains* (Interestingly, even the "knee jerk" is also partially managed by the brain, the "knee jerk" only takes place when attention loses focus). If we treated those sophisticated model/features which are widely used in various applications as low-level neural reflections, e.g., MIFS which is widely used in action recognition (Lan et al. 2015), and treated the powerful deep network as a "*machine brain*", it is interesting that our *knockdown* strategy and the whole model seems consistent with these biological understandings.

Proposed Method

Notations

In this paper, without any loss of generality, suppose we have N examples, denoted by $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)\}$, where each instance has d-dimensional raw inputs, i.e., $\mathbf{x}_i = [x_{i_1}, x_{i_2}, \cdots, x_{i_d}] \in \mathbb{R}^d$, and $y_i \in \{1, 2, , \cdots, c\}$ is the class label of $\mathbf{x}_i$. In the traditional deep learning training procedure, deep models generally compute the semantic feature representation of the input data by passing them through multiple layers of nonlinear transformations, assume that the parameters in network are represented as $\Theta = \{\theta_{l_1}, \theta_{l_2}, \cdots, \theta_{l_p}\}$, and the output of the q-th layer can be denoted as $\mathbf{x}_i^{l_q}$ given $\mathbf{x}_i$ as the input instance. The output l_o layer has c units. Moreover, in our deep transfer learning scenarios, there are additional d_f-dimensional sophisticated features for each instance, i.e., $\mathbf{z}_i = [z_{i_1}, z_{i_2}, \cdots, z_{i_{d_f}}] \in \mathbb{R}^{d_f}$. It is expected that with these additional pre-fixed features, deep network can transfer more information from these features while reducing the dependencies to large amounts of training examples. In another aspect, it is expected that deep transfer learning can achieve better generalization performance than ordinary deep networks given the same number of training examples with the existences of additional task specific fixed features. Eventually, in our configurations, training data can be denoted as $\{\mathbf{x}_i, \mathbf{z}_i, y_i\}_{i=1}^N$.

Fixed Model Reuse (FMR) Approach

In this section, we mainly introduce the concrete steps on how to transfer from fixed model/features to a deep structure network. There can be several different setting:

- **Target replacing with sequence training strategies**:

A straight forward procedure could directly treat the fixed features as the outputs of a deep network as Fig. 1 shows. After training with sufficient examples, the activities/weights

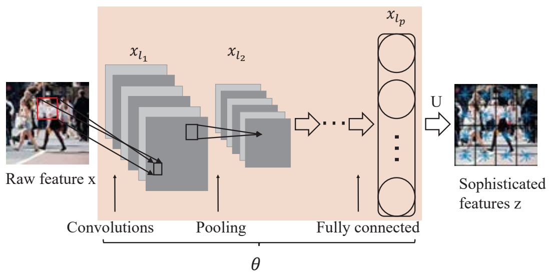


Figure 1: Target replacing with sequence training strategies. $\mathbf{x}$ is the raw feature inputs; $\mathbf{z}$ is the fixed features; deep network are composed with convolution layers, pooling layers and fully connected output layers.

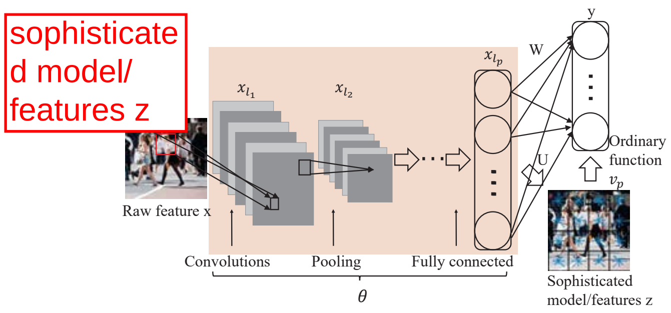


Figure 2: Parallel structure with *knockdown* strategies. $\mathbf{x}$ is the raw feature inputs; $\mathbf{z}$ is the fixed features; $\mathbf{z}$ and V_p compose the sophisticated model; in the parallel structure, the deep network are also composed with convolution layers, pooling layers and fully connected output layers.

learned in the first p layers (shown in yellow shadows) are kept, and then replace the targets with classification labels. This style of deep transfer is generally following the existing deep transfer learning methods (Yosinski et al. 2014; Ajakan et al. 2014).

However, this strategy has obvious shortcomings, e.g., different from transferring between sources and targets with different example distributions, this style of transferring is managing different tasks in this two-step training phases, one for features embedding and learning, the other is the original classification. Besides, this style of transfer learning between features and deep networks obviously divided into two phases, which definitely will induce more chances of error accumulations. Third, the second step of this style of transferring can sweep away the weights in layer $l_1, \cdots, l_p$, which are contributed by the first step with high probabilities, unless there is substantial "theory" guaranteeing there are relationships between fixed features and class labels. This weakness is referred by almost all methods which are learning with two phases.

- **Parallel structure with *Knockdown* strategies**:

To overcome the disadvantage of sequence targets replacing design. We put forward a novel configuration of the structure together with a new learning strategy which pushes the "information" from fixed model/features to the deep networks in parallel, as shown in Fig. 2. Different from the first setting which divided the training strategy into two steps, we directly arrange the deep network and pre-obtained model/features in parallel to the output layer nodes.

Specifically, the raw features $\mathbf{x}$ can be calculated among several layers and can be finally represented as $\mathbf{x}^{l_p}$, and eventually fully connected with the output labels y_s. All convolution facts are denoted as $\Theta = \{\theta_{l_1}, \theta_{l_2}, \cdots, \theta_{l_{p-1}}\}$ and the fully connected weights can be organized as a linear mapping matrix W together with a nonlinear softmax function. Besides, in this parallel structure, there are also linear connections between the provided features and the convolution network layer l_p, which is also a full connection structure, these weights can be denoted as U. The provided features and the labels are connected with linear weights V_p as well. It is obvious that the provided fixed features and V_p compose a traditional linear prediction model which could be useful in conventional application.

Knockdown based Training

In many applications, e.g., sophisticated features are protected, only limited example features can be obtained for training. It becomes a desire for seeking a substitution for these fixed features. In this paper, we propose a novel *knockdown* strategy for facilitating the "information transfer" of fixed features $\mathbf{z}$ to the deep network, and eventually ensure the deep network possessing the representation and discriminative information of the fixed features $\mathbf{z}$. The whole training procedure of FMR can be largely divided into several iterative steps, that is

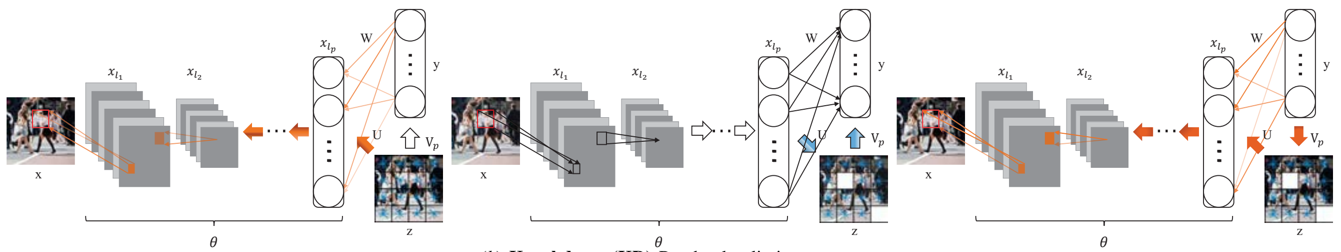
Weight Propagation (WP) As general CNN training, the WP step focuses on reducing the errors made in the current status of the network. Considering there are two lines in parallel in the network as shown in Fig. 2. Without any loss of generalities, the loss function implied in the parallel network structure is:

$$L(\Theta, W, U, V_p) = \sum_{i=1}^{N} \ell(\mathbf{x}_i, \mathbf{z}_i, y_i) + \lambda L_{reg}, \qquad (1)$$

where

$$\begin{aligned}
\ell(\mathbf{x}_i, \mathbf{z}_i, y_i) &= \widetilde{\ell}(\mathbf{x}_i^{l_p}, \mathbf{z}_i, y_i) + \hat{\ell}(\mathbf{x}_i^{l_p}, \mathbf{z}_i);\\
\widetilde{\ell}(\mathbf{x}_i^{l_p}, \mathbf{z}_i, y_i) &= y_i \log h(f(\mathbf{x}_i^{l_p}) + g(\mathbf{z}_i));\\
\hat{\ell}(\mathbf{x}_i^{l_p}, \mathbf{z}_i) &= \frac{1}{2}\|\mathbf{z}_i - \mathbf{x}_i^{l_p} U\|_F^2.
\end{aligned}$$

Here $\mathbf{x}_i^{l_p}$ is the output feature of convolution network layer l_p with raw input example $\mathbf{x}_i$, $\widetilde{\ell}(\mathbf{x}_i^{l_p}, \mathbf{z}_i, y_i)$ is the label prediction loss function ($\widetilde{\ell}$ actually can be with any convex loss functions), in which the $f(\mathbf{x}_i^{l_p})$ is the prediction of $\mathbf{x}_i^{l_p}$, we define as linear function $\mathbf{x}_i^{l_p} W + b_{\mathbf{x}^{l_p}}$ for simplicity here, $b_{\mathbf{x}^{l_p}}$ is the bias for predictors of $\mathbf{x}^{l_p}$, h is a soft-max operator and $g(\mathbf{z}_i)$ is the prediction of the provided features $\mathbf{z}$, we also define as linear function $\mathbf{z}V_p + b_{\mathbf{z}}$, $b_{\mathbf{z}}$ is the bias of fixed features $\mathbf{z}$. L_{reg} can be any convex regularization, while in order to facilitate the WP step, in this paper, we



(a) **Weight Propagation (WP)** Update weights Θ, U, W backwards (marked with yellow left-wards arrows) to reduce errors on y, $\mathbf{z}$

(b) **Knockdown (KD)** Randomly eliminate some features $\mathbf{z}$ together with changing corresponding: V_p (the fixed model) and U (both are marked with blue arrows)

(a') **Weight Propagation after *Knockdown* (WP/K)** Update Θ, U, V_p (marked with yellow left-wards arrows) with remaining $\mathbf{z}$ on changed network

Figure 3: Training approach for FMR model: WP/K$\rightarrow$ KD iteration. It is notable that in KD step, corresponding connections $V_{p_{i,*}}$ and $U_{*,i}$ in weight matrix V_p, U are set to zeros while feature $\mathbf{z}_i$ are eliminated $\big($marked with white blanks in plot (b)$\big)$.

choose L_{reg} as: $\|W\|_2^2 + \|U\|_2^2 + \|V_p\|_2^2$. The parameter λ controls the trade-off between the loss and regularization. In WP step, the derivatives are taken to portion parameters, i.e., Θ, W, U, with the help of Back Propagation technique as shown in Fig. 3(a).

***Knockdown* (KD)** In order to eliminate the influence of the fixed features $\mathbf{z}$ during the training procedure, we need to remove those connected parts corresponding to features $\mathbf{z}$ gradually and finally vanish all related components. Besides, in each adjustment on removing parts of those components, it is required additional steps for making the whole deep structure self-consistent which can further reduce the prediction errors. In this paper, we propose the *knockdown* and consequence procedures, i.e., Weight Propagation after *knockdown* for the purpose above in Fig. 3(b), we randomly remove several components of features saying $\mathbf{z}_{i,j_1}, \mathbf{z}_{i,j_2}, \cdots$, where $j_1, j_2 \in [1, d_f]$, and consequently, the corresponding connections, i.e., $U_{\cdot,j_1}, U_{\cdot,j_2}, \cdots$ in matrix U, $V_{p_{j_1,\cdot}}, V_{p_{j_2,\cdot}}, \cdots$ in matrix V_p are restricted to zero as well. These KD steps will act for several trials in the whole iterations of the Fig. 3 steps. In each iteration, the KD step randomly eliminates components without replacement, and finally will cause all the fixed features $\mathbf{z}$ and the fixed prediction models V_p disconnected with the deep networks. Note that in Fig. 2 and Fig. 3, the deep networks are shown as convolution networks, while as a matter of fact, the networks can be with any deep structure. After all features or the whole predictor models removed, the trained model is the same structure as a traditional deep network. Thus, in the test phase, only raw features $\mathbf{x}$ are required as inputs for the deep model. No sophisticated features are further desired.

Weight Propagation after *Knockdown* (WP/K) This step is generally the same as WP step excepting for updating V_p. However, we here emphasize that the KD step could break the structure of the originally consistent network, therefore after the *knockdown*, this is an additional task for the success weight propagation step, i.e., harmonize the weights and make the network re-consistent, finally reducing the errors. As to the configurations, for deep weights propagation in our implementation, we follow the methods in (Chatfield et al. 2014). Specifically, the training procedure follows (Krizhevsky, Sutskever, and Hinton 2012), using gradient descent with momentum. The hyperparameters are the same as used by (Krizhevsky, Sutskever, and Hinton 2012): momentum 0.9; weight decay $5 \cdot 10^{-4}$; initial learning rate 10^{-2}, which is decreased by a factor of 10. To prevent the internal covariance shift phenomenon, we normalize each channel of the feature map averaging over spatial locations and batch instances as in (Ioffe and Szegedy 2015), in addition, we use weight sharing technique in our network, each filter $h(\mathbf{x}_i)$ is replicated across the entire visual field and these replicated units share the same parameterization (weight vector and bias), which aims to increase learning efficiency by greatly reducing the number of free parameters being learnt. It is notable that in the WP/K step, the parameters Θ, U, V_p are depended only on those fixed features remaining on the network changed by the KD step as shown in Fig. 3(a'). The detail training procedure is shown in Alg. 1.

Algorithm 1 Training Algorithm For FMR

```
Require: limited examples together with fixed features $\hat{D} = \{\mathbf{x}_i, \mathbf{z}_i, y_i\}_{i=1}^N$;
    eliminate number of features in each iteration: $m$; elements in each batch: $n$;
    max-iter: k.
1: repeat
2:     Create Batch: Randomly pick up $n$ examples from $\hat{D}$ without replacement
3:     Calculate the loss $L$
4:     WP/K step (weight consist): Obtain the derivative $\partial L/\partial W$, $\partial L/\partial U$,
       $\partial L/\partial V_p$, $\partial L/\partial \Theta$. Update parameters $W, V_p, U, \Theta$;
5:     KD step (model transfer): Randomly eliminate $m$ fixed features
       without replacement and set corresponding connections to zero, i.e.,
       $U_{\cdot,j_1}, U_{\cdot,j_2}, \cdots = 0, V_{p_{j_1,\cdot}}, V_{p_{j_2,\cdot}}, \cdots = 0$;
6: until converge or reach the max-iter
```

Experiments

FMR can reuse sophisticated model/features and eventually replace the fixed models, i.e., in the test phase, only the raw features $\mathbf{x}$ are required as inputs rather than the features $\mathbf{z}$ extracted by mature extractors. In this section, we will demonstrate that FMR can replace the fixed models based on features $\mathbf{z}$ with a deep structure given raw features as inputs and provide highly-competitive results. In particular, we demonstrate these phenomenons on two real tasks, i.e.,

Table 1: The performance (avg. accuracy) of compared image classification methods. The best performance is marked in bold. Deep learning methods are marked with *.

Compared methods	WIKI	Flickr8k
Drift (Kobayashi 2014)	22.5	51.1
LLC (Wang et al. 2010)	29.2	59.9
ScSPM (Yang et al. 2009)	31.1	60.7
DCM (Ngiam et al. 2011)*	30.3	57.8
CNN (Vedaldi and Lenc 2015)*	42.6	69.9
UDAB (Ganin and Lempitsky 2015)*	42.6	66.0
SQR (Sequence Reuse)*	42.7	72.6
FMR	**43.3 ± 0.5**	**74.8 ± 0.7**

image classification and action recognition. In image classification task, two real-world datasets are tested and fixed models built on surrounding texts of images are acted as sophisticated models. The action recognition task is to recognize the human actions in short clips of videos and there are three real-world datasets in this task.

During the training phase, FMR contains a network with deep architecture which is in parallel with the fixed models. The deep network is implemented the same as MatConvNet (Vedaldi and Lenc 2015), which comprises 8 learnable layers, 5 of which are convolutional layers, and the last 3 are fully-connected. All input images are normalized into the size of 224×224. Fast processing is ensured by the 4 pixels stride in the first convolutional layer. We run the following experiments with the implementation of an environment on NVIDIA K80 GPUs server and our model can be trained about 290 images per second with a single K80 GPGPU.

In image classification task, image pixels are treated as raw features and a linear model built on surrounding texts is considered as the sophisticated fixed model. In action recognition task, we follow (Lan et al. 2015), i.e., randomly extract 10 frames in each clip, and treat the extracted images as the raw inputs, while the extracted MIFS features are treated as the sophisticated features. In classification tasks, 66% instances are chosen as training set and the remains are test set. While in action recognition, training and test splits are provided by (Lan et al. 2015). We repeat experiments 30 times on each dataset, the average accuracies are recorded and evaluated. The parameter λ in the training phase is tuned in $\{0.1, 0.2, \cdots, 0.9\}$. When the variations between the objective value of Eq.1 is less than 10^{-5} in iterations, we consider FMR converges.

Image Classification

Datasets and Compared Methods Two real world datasets are: the WIKI (Rasiwasia et al. 2010) is a rich-text web document dataset with images, which has 2,866 documents extracted from Wikipedia. Each document is accompanied by an image and is labeled with one of the ten semantic classes. We represent the text information by 7343-dimensional vectors based on TF-IDF (Salton and Buckley 1988); Flickr8K (Hodosh, Young, and Hockenmaier 2013) consists of 8,000 images that are each paired with captions which provide clear descriptions in 4 categories (Srivastava and Salakhutdinov 2012). The texts are represented by 3000-dimensional vectors based on TF-IDF as well. FMR is firstly compared to 3 image classification algorithms with traditional image features, i.e., LLC, ScSPM and Drift. Since there is a deep network in FMR, CNN is also compared in the experiments, performance of DCM is also listed since in FMR both the images and surrounding texts are used for training, yet note that only image raw pixels are needed for classification in FMR.

- **LLC**: locality preserving feature projection; max pooling is used for generating the final features; linear classifier is used (Wang et al. 2010);
- **ScSPM**: generalized vector quantization for sparse coding; followed by multi-scale spatial max pooling; linear SPM kernel based on SIFT descriptor for classification (Yang et al. 2009);
- **Drift**: Dirichlet Fisher kernel for new histogram feature representation; linear classifier for classification (Kobayashi 2014);
- **DCM**: Multi-modal subspace feature extraction; linear classifier for classification (Ngiam et al. 2011);
- **CNN**: standard pre-trained CNN with imageNet; only use the image as input for both training and test phase (Vedaldi and Lenc 2015);
- **UDAB**: Deep transfer learning approach; linear classifier for classification (Ganin and Lempitsky 2015);
- **SQR**: Target replacing with sequence training strategies; as mentioned in section 2

From Table 1 it reveals that FMR has achieved the best classification performance, i.e., the mean accuracy, on two datasets compared to all other compared methods.

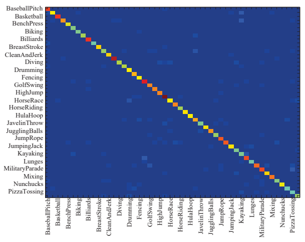
Action Recognition

Datasets and Compared Methods Three most widely used datasets are selected as the action recognition benchmarks, i.e., the HMDB51 dataset (Kuehne et al. 2011) has 51 action classes and 6766 video clips extracted from digitized movies and YouTube (Kuehne et al. 2011). We use original videos in this paper and standard splits. Mean accuracies are used for evaluation of the performance; The UCF101 dataset (Soomro, Zamir, and Shah 2012) has 101 action classes spanning over 13320 YouTube videos clips. We use the standard splits with training and test videos provided by (Soomro, Zamir, and Shah 2012) and mean accuracies are reported as well; The UCF50 dataset (Reddy and Shah 2013) has 50 action classes spanning over 6618 YouTube videos clips and mean accuracies over all classes are also reported. The fixed model/features in action recognitions are the state-of-the-art MIFS features.

We list several most recent features and models of action recognition for comparison, i.e., BoF (Sapienza, Cuzzolin, and Torr 2014), CFS (Oneata, Verbeek, and Schmid 2013), SFV (Peng et al. 2014), FHNN (Karpathy et al. 2014), TSCN (Simonyan and Zisserman 2014), IT (Wang and Schmid 2013a), MIFS (Lan et al. 2015), SQR.

Table 2: The performance (avg. accuracy) of compared action recognition methods. The best performance is marked in bold. Deep learning methods are marked with *.

HMDB51		UCF101		UCF50	
(Oneata, Verbeek, and Schmid 2013)	54.8	(Karpathy et al. 2014)*	65.4	(Narayan and Ramakrishnan 2014)	89.4
(Wang and Schmid 2013a)	57.2	(Sapienza, Cuzzolin, and Torr 2014)	82.3	(Ciptadi, Goodwin, and Rehg 2014)	90.0
(Simonyan and Zisserman 2014)*	59.4	(Wang and Schmid 2013b)	85.9	(Oneata, Verbeek, and Schmid 2013)	90.0
(Peng et al. 2016)	61.1	(Peng et al. 2016)	87.9	(Wang and Schmid 2013a)	91.2
(Peng et al. 2014)	66.8	(Simonyan and Zisserman 2014)*	88.0	(Peng et al. 2016)	92.3
(Lan et al. 2015)	65.1	(Lan et al. 2015)	89.1	(Lan et al. 2015)	**94.4**
SQR (Sequence Reuse)*	62.9	SQR (Sequence Reuse)*	85.3	SQR (Sequence Reuse)*	91.1
FMR	**68.9** $\pm$ **2.3**	FMR	**91.6** $\pm$ **1.2**	FMR	92.4 $\pm$ 1.3


(*a*) UCF50

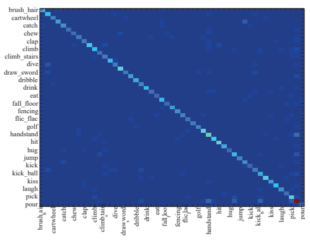

(*b*) HMDB51

Figure 4: Confusion Matrix of action recognition datasets

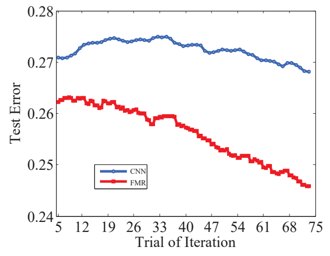


Figure 5: Test error in WP/K-KD iterations on UCF101

Result Note that on these three action recognition benchmarks in Table 2 different methods are compared. This is simply because we only compared FMR with the *six best approaches* for each dataset in *past 5 years open reported publications*. From Table 2, it reveals on HMDB51 and UCF101, FMR outperforms all six best compared approaches obviously, and on UCF50, FMR achieves the runner-up and makes a high-competitive result, i.e., only MIFS method performs better than FMR. The confusion matrix of 50 classes on UCF50 and 51 categories on HMDB51 are also plotted in Fig. 4, where the numbers in the confusion matrix are normalized and replaced by color (from blue to yellow, and white represents 83% while yellow represents 100% in quantity). From Fig. 4, it can be clearly found that on UCF50, the confusion matrix of FMR concentrated on the main diagonal which means FMR performs well on most classes. Similarly, FMR achieves 70% accuracy on most categories according to the confusion matrix on HMDB51.

In order to investigate the convergence behavior for FMR, we conduct more experiments and record the test error curve during iterations in Fig. 5, where it clearly shows the proposed FMR converges faster than ordinary CNN with the help of *knockdown* strategy, and successfully utilizes the fixed model information for accelerating the training. Besides, we also investigated the affections on the number of drooped features in each iteration, and find FMR achieves 91.6 on UCF101 when dropping 52 features in each iteration, while with dropping 104 features iteratively, the accuracy only changes slightly to 92.0.

Conclusion

Reusable model design becomes a desire with the rapid expansion of machine learning applications and reusability is emphasized as a crucial characteristics of *learnware*. Model reuse is helpful to reduce the resource consumptions and avoid building learners from scratch. In this paper, we follow the learnware principle and propose more thorough model reuse approach: FMR. Different from the existing reusable deep learners which are mainly reuse the pre-trained network weights, our approach exploits those mature model or sophisticated features which have been widely used in various applications. The discriminative information behind these mature model/features are further taken over by the powerful deep network structures in FMR. More specifically, FMR arranges the deep network and the sophisticated model/features in parallel initially, and we also propose a novel training strategy "*knockdown*", which can gradually eliminate the influences from fixed model/features and eventually provide a deep learner relying on raw features only. Thus, this scheme guarantees no more requirements on mature model or expensive features in FMR test phase and can be quite useful in applications where the mature model/features are protected by patents or commercial secrets. Experiments on five real-world datasets validate the effectiveness of our methods compared with other state-of-the-art methods. It is interesting to discuss the possibilities of integrating multiple abilities rather than the discriminativity only into one deep network with the help of multiple mature model in various applications, therefore, a further investigation on transferring various models in different applications into a single deep model by the FMR framework can be an interesting future work.

References

Ajakan, H.; Germain, P.; Larochelle, H.; Laviolette, F.; and Marchand, M. 2014. Domain-Adversarial Neural Networks. *arXiv:1412.4446*.

Chatfield, K.; Simonyan, K.; Vedaldi, A.; and Zisserman, A. 2014. Return of the Devil in the Details: Delving Deep into Convolutional Nets. In *BMVC*, Ariticle No. 54.

Ciptadi, A.; Goodwin, M. S.; and Rehg, J. M. 2014. Movement Pattern Histogram for Action Recognition and Retrieval. In *ECCV*, 695–710.

Ganin, Y., and Lempitsky, V. 2015. Unsupervised Domain Adaption by Backpropagation. In *ICML*, 1180–1189.

Gao, W., and Zhou, Z.-H. 2016. Dropout Rademacher Complexity of Deep Neural Networks. *Science China: Information Sci.* 59(12):072–104.

Ghifary, M.; Kleijn, W. B.; and Zhang, M. 2014. Domain Adaptive Neural Networks for Object Recognition. In *PRICAI*, 898–904.

Hodosh, M.; Young, P.; and Hockenmaier, J. 2013. Framing Image Description as A Ranking Task: Data, Models and Evaluation Metrics. *JAIR* 47:853–899.

Ioffe, S., and Szegedy, C. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *ICML*, 448–456.

Kang, G.; Li, J.; and Tao, D. 2016. Shakeout: A New Regularized Deep Neural Network Training Scheme. In *AAAI*, 1751–1757.

Karpathy, A.; Toderici, G.; Shetty, S.; Leung, T.; Sukthankar, R.; and Fei-Fei, L. 2014. Large-Scale Video Classification with Convolutional Neural Networks. In *CVPR*, 1725–1732.

Kobayashi, T. 2014. Dirichlet-based Histogram Feature Transform for Image Classification. In *CVPR*, 3278–3285.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet Classification with Deep Convolutional Neural Networks. In *NIPS*, 1097–1105.

Kuehne, H.; Jhuang, H.; Garrote, E.; Poggio, T.; and Serre, T. 2011. HMDB: A Large Video Database for Human Motion Recognition. In *CVPR*, 2556–2563.

Lan, Z.; Lin, M.; Li, X.; Hauptmann, A. G.; and Raj, B. 2015. Beyond Gaussian Pyramid: Multi-Skip Feature Stacking for Action Recognition. In *CVPR*, 204–212.

Li, N.; Tsang, I. W.; and Zhou, Z.-H. 2013. Efficient Optimization of Performance Measures by Classifier Adaptation. *IEEE TPAMI* 35(6):1370–1382.

Long, M., and Wang, J. 2015. Learning Transferable Features with Deep Adaptation Networks. In *ICML*, 97–105.

Narayan, S., and Ramakrishnan, K. 2014. A Cause and Effect Analysis of Motion Trajectories for Modeling Actions. In *CVPR*, 2633–2640.

Ngiam, J.; Khosla, A.; Kim, M.; Nam, J.; Lee, H.; and Ng, A. Y. 2011. Multimodal Deep Learning. In *ICML*, 689–696.

Oneata, D.; Verbeek, J.; and Schmid, C. 2013. Action and Event Recognition with Fisher Vectors on A Compact Feature Set. In *ICCV*, 1817–1824.

Pan, S. J., and Yang, Q. 2010. A Survey on Transfer Learning. *IEEE TKDE* 22(10):1345 – 1359.

Peng, X.; Zou, C.; Qiao, Y.; and Peng, Q. 2014. Action Recognition with Stacked Fisher Vectors. In *ECCV*, 581–595.

Peng, X.; Wang, L.; Wang, X.; and Qiao, Y. 2016. Bag of Visual Words and Fusion Methods for Action Recognition: Comprehensive Study and Good Practice. *CVIU* 15(1):109–125.

Rasiwasia, N.; Pereira, J. C.; Coviello, E.; Doyle, G.; Lanckriet, G. R.; Levy, R.; and Vasconcelos, N. 2010. A New Approach to Cross-Modal Multimedia Retrieval. In *ACM MM*, 251–260.

Reddy, K. K., and Shah, M. 2013. Recognizing 50 Human Action Categories of Web Videos. *Mach. Visi. Appli.* 24(5):971–981.

Salton, G., and Buckley, C. 1988. Term-Weighting Approaches in Automatic Text Retrieval. *Inform. Proce. Manage.* 24(5):513–523.

Sapienza, M.; Cuzzolin, F.; and Torr, P. H. 2014. Feature Sampling and Partitioning for Visual Vocabulary Generation on Large Action Classification Datasets. *arXiv:1405.7545*.

Sembulingam, K., and Sembulingam, P. 2012. *Essentials of Medical Physiology*. London, UK.: JP. Medical Ltd.

Simonyan, K., and Zisserman, A. 2014. Two-Stream Convolutional Networks for Action Recognition in Videos. In *NIPS*, 568–576.

Soomro, K.; Zamir, A. R.; and Shah, M. 2012. UCF101: A Dataset of 101 Human Actions Classes from Videos in the Wild. *arXiv:1212.0402*.

Srivastava, N., and Salakhutdinov, R. 2012. Multimodal Learning with Deep Boltzmann Machines. In *NIPS*, 2222–2230.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *JMLR* 15(1):1929–1958.

Tian, F.; Gao, B.; Cui, Q.; Chen, E.; and Liu, T.-Y. 2014. Learning Deep Representations for Graph Clustering. In *AAAI*, 1293–1299.

Vedaldi, A., and Lenc, K. 2015. MatConvNet: Convolutional Neural Networks for Matlab. In *ACM MM*, 689–692.

Wang, H., and Schmid, C. 2013a. Action Recognition with Improved Trajectories. In *ICCV*, 3551–3558.

Wang, H., and Schmid, C. 2013b. LEAR-INRIA Submission for the THUMOS Workshop. In *ICCV THUMOS Challenge*, 1–3.

Wang, J.; Yang, J.; Yu, K.; Lv, F.; Huang, T.; and Gong, Y. 2010. Locality-Constrained Linear Coding for Image Classification. In *CVPR*, 3360–3367.

Wang, W.; Arora, R.; Livescu, K.; and Bilmes, J. 2015. On Deep Multi-View Representation Learning. In *ICML*, 1033–1039.

Yang, J.; Yu, K.; Gong, Y.; and Huang, T. 2009. Linear Spatial Pyramid Matching Using Sparse Coding for Image Classification. In *CVPR*, 1794–1801.

Yosinski, J.; Clune, J.; Bengio, Y.; and Lipson, H. 2014. How Transferable are Features in Deep Neural Networks? In *NIPS*, 3320–3328.

Zhou, Z.-H. 2016. Learnware: On the Future of Machine Learning. *Frontiers of Comp. Sci.* 10(4):589–590.