Neurocomputing ∎ (∎∎∎∎) ∎∎∎–∎∎∎

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

DeepFish: Accurate underwater live fish recognition with a deep architecture

Hongwei Qin [a,b], Xiu Li [a,b,*], Jian Liang [b], Yigang Peng [b], Changshui Zhang [b]

[a] Graduate School at Shenzhen, Tsinghua University, Shenzhen 518055, China
[b] Department of Automation, Tsinghua University, Beijing 100084, China

ARTICLE INFO

Article history:
Received 1 May 2015
Received in revised form
28 September 2015
Accepted 11 October 2015

Keywords:
Deep learning
Object recognition
Underwater
Cascaded network

ABSTRACT

Underwater object recognition is in great demand, while the research is far from enough. The unrestricted natural environment makes it a challenging task. We propose a framework to recognize fish from videos captured by underwater cameras deployed in the ocean observation network. First, we extract the foreground via sparse and low-rank matrix decomposition. Then, a deep architecture is used to extract features of the foreground fish images. In this architecture, principal component analysis (PCA) is used in two convolutional layers, followed by binary hashing in the non-linear layer and block-wise histograms in the feature pooling layer. Then spatial pyramid pooling (SPP) is used to extract information invariant to large poses. Finally, a linear SVM classifier is used for the classification. This deep network model can be trained efficiently. On a real-world fish recognition dataset, we achieve the state-of-the-art accuracy of 98.64%.

1. Introduction

Most recognition researches are focused on objects on the ground. However, underwater object recognition is in great demand. The past decade witnessed the fast development of ocean observation. On the one hand, seafloor cabled observatories like the Canada NEPTUNE and VENUS observatories[1] result in unprecedented volumes of underwater visual data. On the other hand, it imposes great challenges for automatic data processing. Fish species and population are among the important tasks of ocean observation, benefiting for academic researchers like ocean scientists and biologists [1], as well as commercial applications like fish farming [2].

However, fish recognition is a challenging research issue. Some special properties of underwater videos impose great challenges for fish recognition. For instance, the videos are usually of low quality because of the extreme conditions in the open sea. The imaging devices are designed with low resolution because of the huge amount of data. The ocean current may cause frequent luminosity change. Absorption and scattering may cause light attenuation, limiting the visibility, especially in the deeper sea. Furthermore, the fish moves erratically and fast in 3D space and against coral and sand. All of the above result in the variety of fish videos.

In this paper, we aim to find a solution to underwater object recognition. We propose a framework for underwater live fish recognition in unconstrained natural environment. The datasets are captured by underwater cameras in the open sea. A deep architecture is designed. The features are learned from the training data, so domain knowledge of fish is not needed. This architecture can be designed and learned efficiently compared to state-of-the-art deep learning architectures carefully learned (by DNNs). Code is available at https://github.com/qinhongwei/deepfish-release.

2. Related work

2.1. Fish recognition state-of-the-art

Prior fish recognition researches are mainly restricted to constrained environments. Lee et al. [3] used contour matching to recognize fish in the fish tanks. Strachan et al. [4–6] used color and shape descriptors to recognize fish transported along a conveyor belt underneath a digital camera. Larsen et al. [7] derived shape and texture features from an appearance model, classified with LDA, and tested on a dataset containing 108 images of three fish species under constrained condition (caught from the sea), achieving an accuracy of 76%. There are also researches carried out on underwater live fish. Spampinato et al. [8] combined texture features and shape features and tested on a database containing

* Corresponding author at: Graduate School at Shenzhen, Tsinghua University, Shenzhen 518055, China.
E-mail address: li.xiu@sz.tsinghua.edu.cn (X. Li).
[1] ⟨http://www.oceannetworks.ca/⟩.

http://dx.doi.org/10.1016/j.neucom.2015.10.122
0925-2312/

360 images of ten fish species, achieving an ave rage accuracy of 92%. Huang et al. [9] presented a Balance-Guaranteed Optimized Tree (BGOT) algorithm to control the error accumulation in hierarchical classification. They carried out an experiment on a dataset containing 3179 fish images of 10 species collected from underwater videos, and got an accuracy of 95%. Then, in [10], they further used GMM to improve the reject option in hierarchical classification. With BGOT+GMM they got an average precision of 65% on a larger dataset containing 24,150 images of 15 species.

Previous researches are not satisfying in four aspects. Firstly, most of them concentrate on fish images under constrained conditions. Secondly, they use small datasets containing a small number of fish and limited fish species. Thirdly, all the current fish recognition methods use hand-crafted features and these features are often combined to improve performance, so these methods are task-specific and are limited in generalization capability. Fourthly, the algorithms are not efficient on large datasets. Last but most importantly, the accuracy is not satisfying under constrained and unconstrained conditions. However, the fast increasing unprecedented volumes of underwater visual data demands much more. To meet that demand, we need accurate, efficient and robust recognition on large unconstrained datasets.

2.2. Large scale image classification and deep learning

Image classification is a quite challenging task. Conventional solutions for classification use manually designed low-level features. For example, SIFT and HOG features are used for object recognition, LBP and Gabor features are used for texture and face classification. The carefully hand-crafted low-level features do achieve good performance for some specific data and tasks. However, effective features require domain knowledge and most of them cannot simply apply to new conditions [11–13]. Besides, the generalization capability of many conventional machine learning tools like SVM, PCA and LDA, tend to saturate quickly as the volume of the training set grows significantly [14].

Hinton et al. [11] proposed a method to learn features through deep neural networks (DNNs), which significantly influences the machine learning field in recent years. Deep learning aims to learn multiple levels of representation of the data, from low-level to high-level, to make sense of data such as images, sound and text. High-level representation gives more information of the semantics of the data.

Deep and large networks have shown impressive results when used with large amounts of training data and scalable computation resources (thousands of CPU cores [15] and/or GPU computing [16]). Many conventional tasks have benefitted from this technical progress [17]. Most notably, Krizhevsky et al. [16] proved the effectiveness of deep convolutional neural networks trained on ImageNet [18] and achieved an excellent classification accuracy.

The success of deep learning in image classification lies in one key ingredient which is the use of convolutional architectures [19–21,16]. Generally, a deep neural network contains multiple trainable cascaded stages, followed by a supervised classifier. And each stage generally contains a convolutional filter bank layer, a nonlinear processing layer, and a feature pooling layer [13]. For the filter bank layer, RBM [21] and auto-encoders or their variations are used [12]. In general, Stochastic Gradient Descent (SGD) is used to learn such a network.

Though deep learning based methods have been used in many research fields [22,23], the most effective field is object recognition. Recently, there is a rapid development of object recognition algorithms. Before deep learning is widely used, linear spatial pyramid matching using sparse coding was used for image classification, achieving good performance on several image classification benchmarks by using a single type of descriptors [24]. Adaptive hypergraph learning and high-order distance-based multi-view stochastic learning were investigated for image classification tasks [25,26]. Fisher kernel was also very successfully used [27]. More recently, architectures like VGG [28,29], GoogLeNet [30] and PReLU-nets [31] achieved better and better, even surpassing human-Level performance on ImageNet classification task.

Learning a deep neural network for classification critically depends on expertise of parameter tuning. A simple deep learning baseline network for image classification named PCANet was proposed recently [13]. This network is designed to be easy to train and adaptable to different tasks. It is on par with the state-of-the-art features (prefixed, hand-crafted, or learned from DNNs) for some kinds of image classification tasks.

Inspired by the connection between general deep convolutional neural networks and PCANet, we try to find an effective and simple architecture to solve the underwater live fish recognition problem.

3. Proposed framework

3.1. Foreground extraction of underwater videos

Underwater stationary cameras deployed in the ocean observation networks are often faced with extreme conditions caused by various natural environment. Qin et al. [32] proposed a foreground extraction method for underwater videos based on sparse and low-rank matrix decomposition. Inspired by their method, we extract the foreground fish masks from the videos, thus the complex background can be eliminated. The preprocessing can make it easier for the fish recognition task.

3.2. Fish recognition with deep architecture

Recently, in many object recognition systems, feature extraction stages are generally composed of a filter bank layer, a nonlinear transformation, and a feature pooling layer. For example, the recently widely noted convolutional neural networks (CNN) is such a system. A typical CNN consists of several layers, and can be regarded as a stage. A convolutional filter bank layer aims to extract local patterns. A nonlinear processing layer aims to form a non-linear complex model. A feature pooling layer aims to decrease feature maps' resolution. Deep CNN may contain several stages. In the network we use, the PCA filters are chosen for the filter bank layer, the binary hashing is chosen for the nonlinear layer, and the block-wise histograms of the binary codes are chosen for the feature pooling layer. Then spatial pyramid pooling (SPP) is used to extract information invariant to large poses. The final output is fed to an SVM classifier.

Fig. 1 illustrates the pipeline of the framework.

3.2.1. Input images

Suppose N input training images are given. First, we need to resize the images to the same size $m \times n$.

3.2.2. Layer size selection

We choose an architecture with two stages. The stage here is not completely the same as that in the DNNs. We regard the convolutional filter bank layer as a stage. So the two stages are actually two convolutional layers.

3.2.3. Convolutional layer 1

We set the patch size (i.e. 2D filter size) as $k_1 \times k_2$ at all the convolutional layers. Given an input image, we take a $k_1 \times k_2$ patch around each pixel. Then we subtract the patch mean from each

Please cite this article as: H. Qin, et al., DeepFish: Accurate underwater live fish recognition with a deep architecture, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.10.122

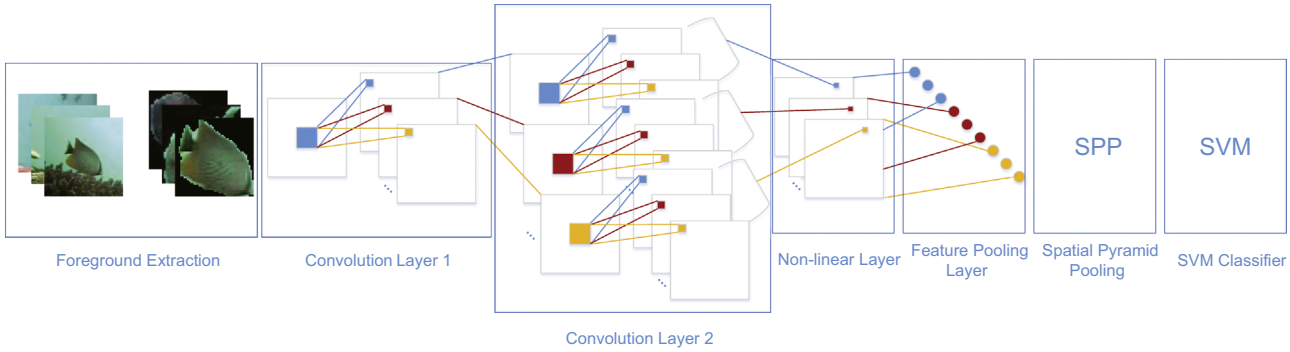


Fig. 1. The pipeline of the proposed framework. PCA filters are chosen for the filter bank layer, the binary hashing is chosen for the nonlinear layer, and the block-wise histograms of the binary codes are chosen for the feature pooling layer. Then spatial pyramid pooling (SPP) is used to extract information invariant to large poses. The final output is fed to a linear SVM classifier.

patch, and get the mean-removed patch. So, for a given color image X_i, for each channel c (c is r,g,b respectively), we get $\overline{X}_{c,i}=[\overline{x}_{c,i,1},\overline{x}_{c,i,2},\ldots,\overline{x}_{c,i,mn}]$, where $\overline{x}_{c,i,j}$ is a mean-removed patch. Repeat the process for each input image, and we get

$$X_c=[\overline{X}_{c,1},\overline{X}_{c,2},\ldots,\overline{X}_{c,N}]\in\mathbb{R}^{k_1k_2\times N_{mn}}. \tag{1}$$

Another parameter is the number of filters in each convolutional layer, denoted by L_i for the ith convolutional layer. We use PCA to minimize the reconstruction error within a set of orthogonal filters,

$$\min_{V\in\mathbf{R}^{3k_1k_2\times L_1}} \|X-VV^TX\|_F^2, \quad \text{s.t.}\quad V^TV=I_{L_1}, \tag{2}$$

where $X=[X_r^T,X_g^T,X_b^T]^T$, I_{L_1} is an identity matrix of size $L_1\times L_1$, V is a matrix consisting of a set of eigenvectors. Solve the optimization problem, and we get the L_1 principal eigenvectors of XX^T. Then the PCA filters of the first stage can be denoted as

$$W_l^1=\mathrm{mat}_{k_1,k_2,3}(q_l(XX^T))\in\mathbb{R}^{k_1\times k_2\times 3},\quad l=1,2,\ldots,L_1, \tag{3}$$

where $q_l(XX^T)$ is the lth principal eigenvector of XX^T, $\mathrm{mat}_{k_1,k_2,3}(v)$ is a function that maps $v\in\mathbb{R}^{3k_1k_2}$ to a matrix $W\in\mathbb{R}^{k_1\times k_2\times 3}$. The main variation of the mean-removed training patches can be expressed by the leading eigenvectors. Finally, the output of the first stage is

$$\mathbf{I}_i^l=\mathbf{I}_i*W_l^1,\ i=1,2,\ldots,N, \tag{4}$$

where $\mathbf{I}_i$ is the ith input image, and $W_l{}^1$ is the lth filter of the PCA filter bank in the first stage.

3.2.4. Convolution layer 2

This layer is basically the same as the last layer. The output of the last layer is used as the input of this one. The filter number is L_2. Besides this, the other processing are all the same. For each input from the last layer, we get L_1 outputs. Hence, the output of this layer is L_1L_2. Finally, for each input $\mathbf{I}_i^l$, the output of the second layer is

$$\mathbf{O}_i^l=\left\{\mathbf{I}_i^l*W_{l^*}^2\right\}_{l^*=1}^{L_2}. \tag{5}$$

where $W_{l^*}^2$ is the l^*th filter of the PCA filter bank.

If we want to build a deeper architecture, we can just add more stages (convolutional layers) to repeat the above process. The final number of the output filters would be $\prod_{i=1}^{nstage}L_i$, where *nstage* is the number of stages.

3.2.5. Nonlinear layer

After the second layer, for each of the L_1 inputs I_i^l, we get L_2 real-valued outputs $\left\{\mathbf{I}_i^l*W_{l^*}^2\right\}_{l^*=1}^{L_2}$. In this layer, these outputs are binarized to $\left\{H(\mathbf{I}_i^l*W_{l^*}^2)\right\}_{l^*=1}^{L_2}$, where $H(\cdot)$ is a Heaviside step function, whose output is one for positive entries and zero otherwise. This processing can also be called *hashing*. Then we convert the L_2 binary bits to a decimal number, denoted as

$$\mathbf{T}_i^l=\sum_{l^*=1}^{L_2}(\mathbf{I}_i^l*W_{l^*}^2). \tag{6}$$

The above process converts the L_2 outputs back into a single integer-valued *image*, whose pixel value ranges from 0 to $2^{L_2}-1$. As a result, the outputs number for now turn to L_1 again.

3.2.6. Feature pooling layer

After the nonlinear layer, we get L_1 outputs. In this layer, each of the L_1 outputs is partitioned into H blocks and the histogram of the decimal values in each block is computed. Then, we concatenate all the H histograms into one vector, denoted as $Hist(\mathbf{T}_i^l)$. Finally, the ultimate output is defined as

$$f_i=[Hist(\mathbf{T}_i^l),\ldots,Hist(\mathbf{T}_i^{L_1})]^T\in\mathbb{R}^{2^{L_2}L_1H}. \tag{7}$$

As can be seen, the output or the *feature* has a dimension of $L_12^{L_2}$ for now. The local blocks are overlapped, and the specific overlapping ratio is another parameter of the network, which can be set according to different tasks. Actually, the histogram offers some degree of translation invariance in the extracted features, just like in hand-crafted features such as Scale-invariant Feature Transform (SIFT) [33] or Histogram of Oriented Gradients (HOG) [34], and average or max pooling process in ConvNet [20,19,35,16,36].

3.2.7. Spatial pyramid pooling (SPP)

As fish images consists of complex poses, we connect the Spatial Pyramid Pooling [37] process to the above output. This process can extract information invariant to large poses.

3.2.8. Classifier

We choose SVM classifier to complete the recognition. Generally, Softmax classifier is another widely adopted option. In the experiment section, we will discuss why we choose SVM.

As a conclusion, the parameters of the architecture include the filter size k_1,k_2, the number of stages (convolutional layers), the number of filters in each stage L_1,L_2, the block size for local histograms, the overlapping ratio between blocks and the pyramid vector.

4. Experiments and results

In this section, we carry out a series of experiments. We implement our architecture with Matlab, and use Liblinear [38] for

Please cite this article as: H. Qin, et al., DeepFish: Accurate underwater live fish recognition with a deep architecture, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.10.122

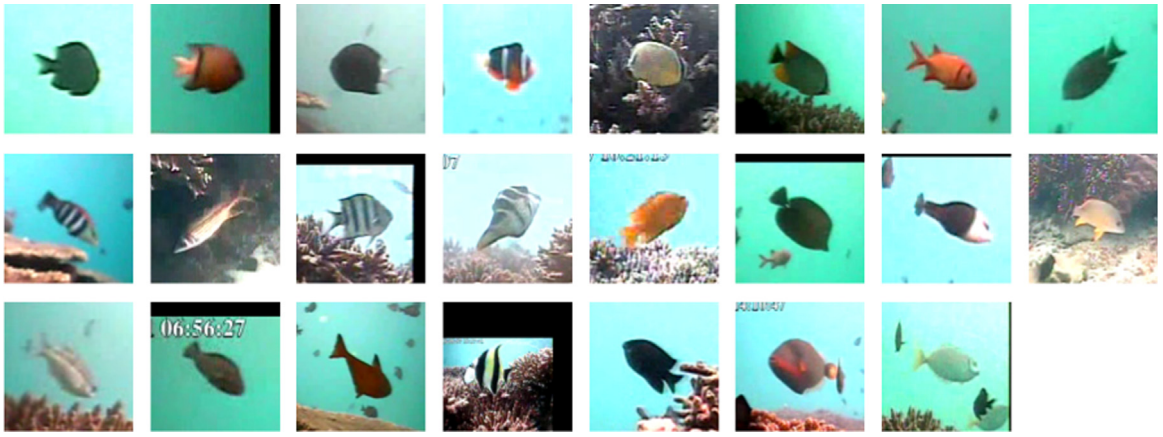


Fig. 2. From top left to bottom right, representative species of 23 clusters of fish. There are totally 27,370 images acquired from live videos captured from the open sea. The images shown are ideal images because many of the others in the dataset are of low quality, such as blur, various depth of focus, or complex background.

SVM classifier. Many data process and parameter setting details are discussed to improve the performance of our method. In the following section, we will compare our method using deep learning architecture with conventional methods.

4.1. Dataset

We evaluate the effectiveness of the deep architecture on the Fish Recognition Ground-Truth dataset made by the Fish4-Knowledge (F4K) project [39]. This underwater live fish dataset is acquired from a live video dataset captured from the open sea. There are totally 27,370 verified fish images of 23 clusters and each cluster is presented by a representative species. The fish species are manually labeled by following instructions from marine biologists. The fish images and masks are both given. Fig. 2 is an example of the 23 fish species. These RGB fish images have various sizes ranging from about 20×20 to about 200×200 pixels. The images shown in Fig. 2 are already resized to the same size. Besides, the number of different fish species is quite imbalanced. The number of the most frequent species is about 1000 times of the least one. So it is quite difficult to achieve a high accuracy over the whole dataset.

Images in the dataset vary significantly not only in fish position, scale and orientation within each class, but also in colors and even textures. With the fish masks given in the dataset, we can first eliminate the background of the fish image. To use the deep architecture, the images should be resized to the same size. Considering the average size of the images, we resize all the images to 47×47, which is neither too large to be very computation resource consuming, nor too small to lose too much information. The total images are divided into three subsets: 5/7 for training, 1/7 for validation, and 1/7 for test. The distribution of the fish species in the dataset is shown in Table 1. As the number of different fish species is quite imbalanced, each species is divided in the same proportion randomly.

Table 1
The fish species distribution.

ID	Species	Number
1	Dascyllus reticulatus	12,112
2	Plectroglyphidodon dickii	2683
3	Chromis chrysura	3593
4	Amphiprion clarkii	4049
5	Chaetodon lunulatus	2534
6	Chaetodon trifascialis	190
7	Myripristis kuntee	450
8	Acanthurus nigrofuscus	218
9	Hemigymnus fasciatus	241
10	Neoniphon sammara	299
11	Abudefduf vaigiensis	98
12	Canthigaster valentini	147
13	Pomacentrus moluccensis	181
14	Zebrasoma scopas	90
15	Hemigymnus melapterus	42
16	Lutjanus fulvus	206
17	Scolopsis bilineata	49
18	Scaridae	56
19	Pempheris vanicolensis	29
20	Zanclus cornutus	21
21	Neoglyphidodon nigroris	16
22	Balistapus undulatus	41
23	Siganus fuscescens	25
Total		22,370

4.2. Architecture parameters setting

We train of network with the following parameters, which are tuned according to the validation set. For the convolutional filter bank layer, the filter sizes are $k_1 = 5$, $k_2 = 13$ respectively, and the numbers of the filters are $L_1 = 32$, $L_2 = 6$ respectively. For the histogram layer, the block size is set as 8×8, and the overlapping ratio between blocks 0.6. Finally, SPP is connected to the output layer (histogram layer). The maximum response in each bin of block histograms is pooled in a pyramid of 4×4, 2×2, and 1×1 subregions. After that we get $21(=4 \times 4+2 \times 2+1 \times 1)$ pooled histogram features having a dimension of $L_1 2^{L_2}$. Then, a SVM classifier is adopted to perform the classification. To reduce the impact of data imbalance, we use data augmentation on part of the training data. Finally, we achieve an accuracy of 98.23% on the test set. We denote this method with DeepFish-SVM.

The parameters are optimized by linear search. In the experiments, we observe that (1) larger convolution kernel size (filter size) in the second stage brings performance improvement, while it cannot be too large because of the image size and memory limitation. (2) Larger L1 (number of first stage filters) brings performance improvement. (3) Generally more parameters bring better performance, while it is limited by many aspects, such as input image size, parameter matching and computer hardware.

4.3. Result after data augmentation on the training set

Data augmentation is widely used in deep learning architectures to enlarge the training data set to get better generalization [16]. Considering the imbalance of the data distribution, we

Please cite this article as: H. Qin, et al., DeepFish: Accurate underwater live fish recognition with a deep architecture, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.10.122

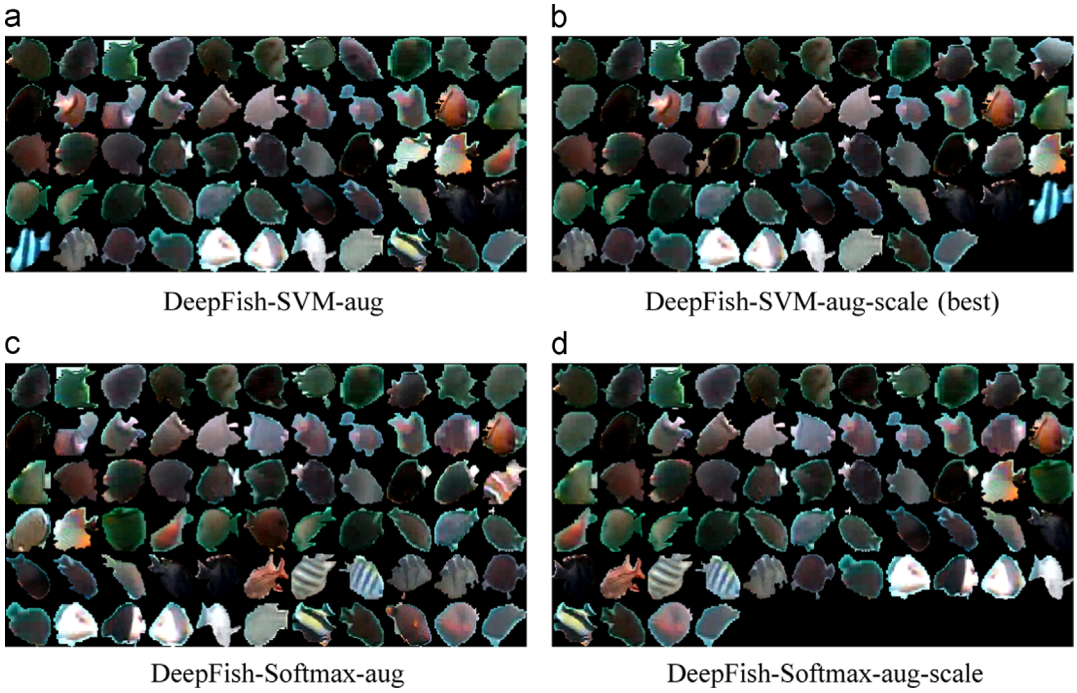


Fig. 3. Error instances. (a) The 55 falsely classified fish images out of 3908 test images with DeepFish-SVM-aug. (b) The 53 falsely classified fish images out of 3908 test images with DeepFish-SVM-aug-scale. (c) The 66 falsely classified fish images out of 3908 test images with DeepFish-Softmax-aug. (d) The 59 falsely classified fish images out of 3908 test images with DeepFish-Softmax-aug-scale.

enlarge the training set for the species whose image number is less than 300. Specifically speaking, the images are randomly rotated with an angle between -10° and 10°, and added to the original training set. The process is repeated for 5 times. Then we use the same parameters tuned above to train the network. On the test set, we get a higher accuracy of 98.59%. We denote this method with DeepFish-SVM-aug. The 55 error instances of the 3908 test images are shown in Fig. 3(a).

We only perform rotation on the training data. Because the foreground extraction procedure and foreground image resizing make the fish images unified in size, the translation and zoom would not be necessary.

4.4. Final classifier: SVM vs Softmax

After we extract the features using our architecture, the features and labels are fed into an SVM classifier. Various classifiers are used at the last stages in modern deep architectures. Softmax classifier is frequently used. In our experiment, we also train a Softmax classifier with the features of the training dataset. The accuracy on the test dataset is 92.55%. We denote this method with DeepFish-Softmax-aug. However, we scale the feature to 100 times larger, and the accuracy rises to 98.31%. The falsely classified fish image number is 66, as shown in Fig. 3(c). We further tune the weight decay and termination tolerance parameters of Softmax, and the accuracy rises to 98.49%. The falsely classified fish image number is 59, as shown in Fig. 3(d). We denote this method with DeepFish-Softmax-aug-scale.

Inspired by this improvement, we scale the features to 100 times larger and train another SVM classifier model. The accuracy is 98.64%, which gets a minor improvement compared with before. We denote this method with DeepFish-SVM-aug-scale. The falsely classified fish image number is 53, as shown in Fig. 3(b).

Table 2
Comparison of fish recognition accuracy (%) of various methods on the test set. *Scale* means scaling the feature vector value to 100 times larger. *Aug* means part data augmentation on the training set. Deep-CNN is our CNN architecture. The bold means accuracy is higher than 98%.

Method	Accuracy (%)
LDA + SVM	80.14
Raw-pixel SVM	82.92
Raw-pixel Softmax	87.56
Raw-pixel Nearest Neighbor	89.79
VLFeat Dense-SIFT	93.58
DeepFish-SVM (our)	**98.23**
DeepFish-SVM-aug (our)	**98.59**
DeepFish-SVM-aug-scale (our)	**98.64**
DeepFish-Softmax-aug (our)	92.55
DeepFish-Softmax-aug-scale (our)	**98.49**
Deep-CNN (our)	**98.57**

4.5. Results analysis

4.5.1. Comparisons

To make comparison, we also carry out experiments using conventional machine learning tools as baseline methods. With a widely used baseline method LDA (to extract features) + SVM (classifier) [38], using foreground fish images (each one reshaped as a vector) as input, the accuracy is 80.14%. With nearest neighbor method, the accuracy is 89.79%. As baseline methods, we train an SVM classifier on the raw pixels. With this classifier, we obtain an accuracy of 82.98%. We also train a Softmax classifier, with which we obtain an accuracy of 87.56%. The popular VLFeat [40] library is also used as one of the baseline methods. We use PHOW features (dense multi-scale SIFT descriptors), Elkan k-means for fast visual word dictionary construction, spatial histograms as image descriptors, a homogeneous kernel map to transform a Chi2 support vector machine (SVM) into a linear one, and SVM classifiers. The results are listed in Table 2.

Please cite this article as: H. Qin, et al., DeepFish: Accurate underwater live fish recognition with a deep architecture, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.10.122

Table 3
When train:val:test = 5:1:1. The distribution of the 55 falsely classified images. Total number means the total number of that fish species. False number means the number of the falsely classified images.

ID	Species	False number	Total number	Accuracy (%)
1	Dascyllus reticulatus	12	1730	99.31
2	Plectroglyphidodon dickii	11	383	97.13
3	Chromis chrysura	7	513	98.64
4	Amphiprion clarkii	0	578	100.00
5	Chaetodon lunulatus	0	362	100.00
6	Chaetodon trifascialis	2	27	92.59
7	Myripristis kuntee	1	64	98.44
8	Acanthurus nigrofuscus	11	31	64.52
9	Hemigymnus fasciatus	0	34	100.00
10	Neoniphon sammara	0	43	100.00
11	Abudefduf vaigiensis	1	14	92.86
12	Canthigaster valentini	1	21	95.24
13	Pomacentrus moluccensis	0	26	100.00
14	Zebrasoma scopas	2	13	84.62
15	Hemigymnus melapterus	2	6	66.67
16	Lutjanus fulvus	1	29	96.55
17	Scolopsis bilineata	1	7	85.71
18	Scaridae	0	8	100.00
19	Pempheris vanicolensis	0	4	100.00
20	Zanclus cornutus	1	3	66.67
21	Neoglyphidodon nigroris	1	2	50.00
22	Balistapus undulatus	1	6	83.33
23	Siganus fuscescens	0	4	100.00
Total		55	3908	98.59

Table 4
When train:val:test = 5:1:1. The distribution of the 219 falsely classified images. Total number means the total number of that fish species. False number means the number of the falsely classified images.

ID	Species	False number	Total number	Accuracy (%)
1	Dascyllus reticulatus	32	5190	99.38
2	Plectroglyphidodon dickii	25	1149	97.82
3	Chromis chrysura	35	1539	97.73
4	Amphiprion clarkii	7	1734	99.60
5	Chaetodon lunulatus	1	1086	99.91
6	Chaetodon trifascialis	10	81	87.65
7	Myripristis kuntee	12	192	93.75
8	Acanthurus nigrofuscus	21	93	77.42
9	Hemigymnus fasciatus	3	102	97.06
10	Neoniphon sammara	4	129	96.90
11	Abudefduf vaigiensis	9	42	78.57
12	Canthigaster valentini	12	63	80.95
13	Pomacentrus moluccensis	1	78	98.72
14	Zebrasoma scopas	12	39	69.23
15	Hemigymnus melapterus	8	18	55.56
16	Lutjanus fulvus	5	87	94.25
17	Scolopsis bilineata	5	21	76.19
18	Scaridae	2	24	91.67
19	Pempheris vanicolensis	0	12	100.00
20	Zanclus cornutus	2	9	77.78
21	Neoglyphidodon nigroris	2	6	66.67
22	Balistapus undulatus	8	18	55.56
23	Siganus fuscescens	3	12	75.00
Total		219	11,724	98.13

The deep network itself is proved effective. And the higher accuracy brought by the data augmentation process may result from the fact that the fish poses and orientations are various and slightly and randomly rotating the images reduces that influence. As we can see in Fig. 3(b). Errors mainly result from wrong annotations of foreground mask, confusing colors and low resolution.

Besides, training the network is efficient, because the filter learning does not involve regularized parameters and does not require numerical optimization solver. With above parameters and data augmentation, the training procedure takes about 3 hours on an Intel (R) Xeon(R) CPU E5-2650 2.60 GHz computer (only one core is used).

As for larger-sized fish, we think that the architecture also applies. The generalization of deep architectures are widely proved in recent deep learning research. We just need to vary the hyper parameters.

4.5.2. Partition of datasets

To make sure that the dataset partition is appropriate, we also carry out another experiment, in which we decrease the training set to 3/7, and increase the test set to 3/7. With the same training settings before, the accuracy is 98.13%, which is only a tiny decline, indicating that our framework performs well.

In the first experiment, the total images are divided into three subsets: 5/7 for training, 1/7 for validation, and 1/7 for test. The distribution of the falsely classified images are shown in Table 3.

In the second experiment, the total images are divided into three subsets: 3/7 for training, 1/7 for validation, and 3/7 for test. The distribution of the falsely classified images are shown in Table 4

From Tables 3 and 4, we can see that for each species, the larger image number brings better accuracy in general.

4.5.3. The difference of Softmax and SVM

In our deep architecture, we adopted a linear SVM layer instead of commonly used Softmax layer as classifier. According to the experiments in [41] that such switching is effective in that the objective function of hinge loss has higher positive correlation with test error. Our results with both kinds of classifier types demonstrate increasing performance of this switching. Meanwhile, hinge loss brings the convenience of screening the samples outside the margin, which makes it easier to accelerate training [17]. That is why we use a linear SVM classifier layer at the end of our architecture.

5. Visualizing and understanding the architecture

To make a better understanding of the architecture, in this section, we visualize the convolutional layer filters and outputs of each convolutional layer.

5.1. Filter visualization

In the first PCA stage, there are 32 convolutional kernels of size $5 \times 5 \times 3$, which is shown in Fig. 4(a). In the second PCA stage, there are 6 convolutional kernels of size 13×13, which is shown in Fig. 4(b). The first stage filters are very nice and smooth, indicating that they are nicely learned. The color and grayscale features are clustered to some extent. The grayscale filters extract high-frequency features, while the color filters extract low-frequency features. The second stage filters are also very smooth, well-formed and free of noisy patterns. We can infer from the visualizations that the PCA stages are effective.

5.2. Feature visualization

The outputs of the first PCA stage are 32 images of size 47×47, which are shown in Fig. 5(a). The outputs of by the second PCA stage are 192 images of size 47×47, which are shown in Fig. 5(b). We can see that the first PCA stage outputs are relatively dense and the second are more sparse and localized. In conventional methods, which features to extract are usually carefully decided by experts. While in our architecture, filters are learned and feature extraction are fully automatic without domain knowledge.

Please cite this article as: H. Qin, et al., DeepFish: Accurate underwater live fish recognition with a deep architecture, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.10.122

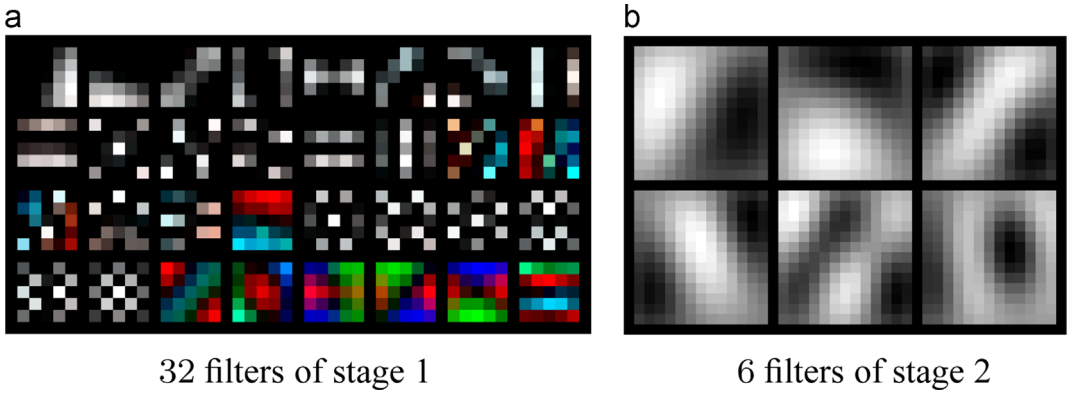


Fig. 4. Convolutional layer filters visualization of our Deepfish-SVM-aug architecture. (a) The 32 filters (each of size $5 \times 5 \times 3$) learned by the first PCA stage. (b) The 6 filters (each of size 13×13) learned by the second PCA stage. Notice that the first stage filters are very nice and smooth, indicating that they are nicely learned. The grayscale filters extract high-frequency features, while the color filters extract low-frequency features. The second stage filters are also very smooth, well-formed and free of noisy patterns.

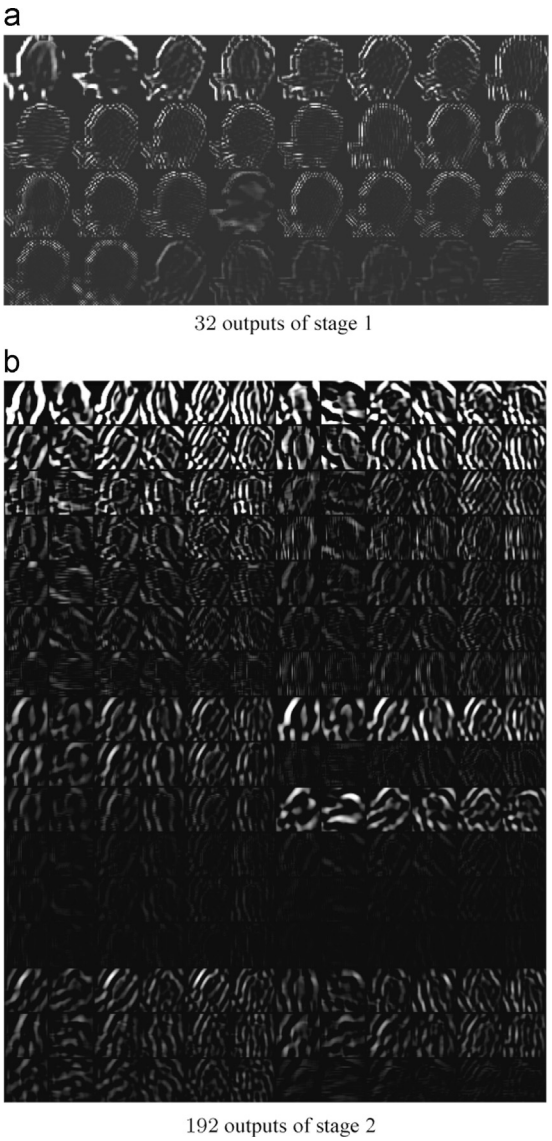


Fig. 5. Typical convolutional layer features visualization of our Deepfish-SVM-aug architecture. (a) The 32 output images (each of size 47×47) of the first PCA stage. (b) The 192 output images (each of size 47×47) of the second PCA stage filters. Notice that the first PCA stage outputs are relatively dense and the second are more sparse and localized.

6. CNN architecture for fish recognition

Convolutional Neural Networks are widely used in object recognition tasks [19,16]. To explore a best architecture for underwater object recognition, we design a CNN architecture to make comparison with the previous architecture.

The overall architecture of our CNN is depicted in Fig. 6.

The net contains six layers with weights, of which the first three are convolutional and the remaining three are fully connected. The output of the last fully-connected layer is fed to a 23-way Softmax which produces a distribution over the 23 class labels.

This net maps an input image x_i, via a series of layers, to a probability vector $\hat{y}_i$ over the 23 different classes. Each layer consists of the following operations: (i) convolution of the previous layer output (or, in the case of the 1st layer, the input image) with a set of learned filters (optimized weights); (ii) passing the responses through a rectified linear function $relu(x) = \max(x, 0)$; (iii) max pooling over local neighborhoods and (iv) a local contrast operation that normalizes the responses across feature maps.

6.1. Architecture details

As demonstrated in Fig. 6, the first convolutional layer filters the $47 \times 47 \times 3$ input image with 70 kernels of size $5 \times 5 \times 3$ with a stride of 1 pixel. The outputs are max pooled with kernel size 3×3, fed to ReLU layer and normalized. The second and the third convolutional layers repeat the above process, besides that the filters kernel size are both adjusted to 3×3. In detail, the second convolutional layer filters the (pooled, rectified, and normalized) outputs of the first convolutional layer with 110 kernels of size $3 \times 3 \times 70$, and the third layer has 180 kernels of size $3 \times 3 \times 110$ connected to the outputs of the second layer. The fully-connected layers have 200, 22, 23 neurons in turn. In total, the network has about 1 million parameters. We denote this method with Deep-CNN.

6.2. Training details

The model is trained on the same dataset as before. The preprocess is also the same. We implement the architecture with Caffe [42] framework. Stochastic gradient descent with a mini-batch size of 64 is used to update the parameters. The base learning rate is set to 10^{-2}, in conjunction with a momentum term of 0.9 and weight decay of 0.0005. An equal learning rate for all layers is used and adjusted manually throughout training. When the validation error comes to plateaus, the learning rate is decreased by a factor of 10.

Please cite this article as: H. Qin, et al., DeepFish: Accurate underwater live fish recognition with a deep architecture, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.10.122

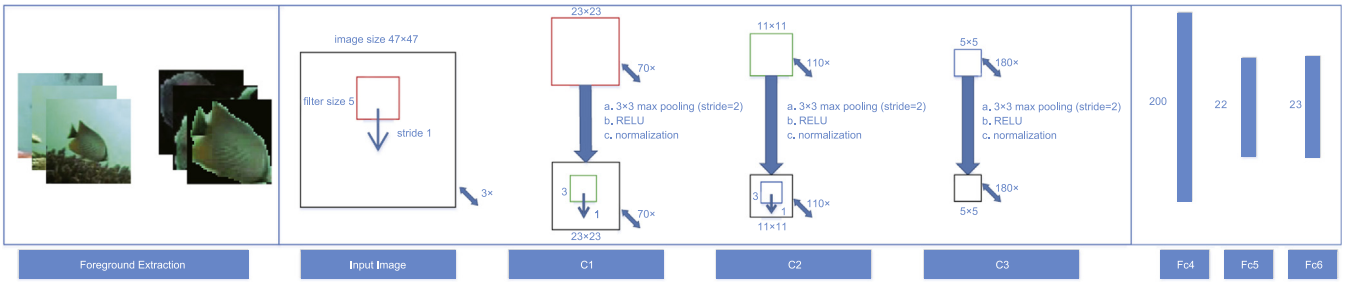


Fig. 6. The architecture of our CNN, showing the definition of different layers. The network's input is 6627-dimensional, and the number of neurons of each layer is given by 37030-13310-4500-200-22-23.

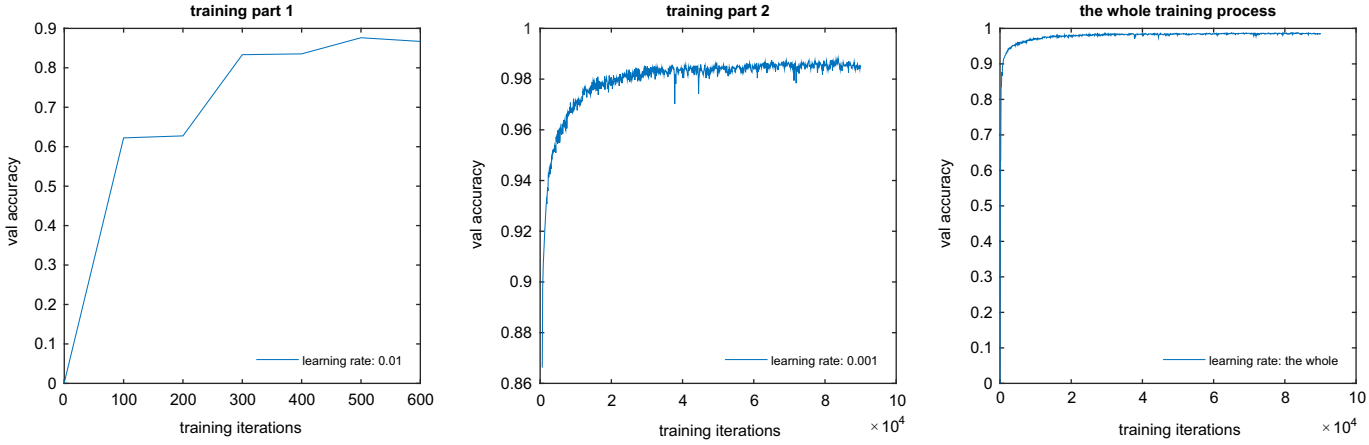


Fig. 7. Validation accuracy vs. training iterations: the training process contains two parts, the first of which has a fixed learning rate of 0.01, and the second 0.001, as shown in the left and middle columns. The right column shows the validation accuracy change in the whole training process.

Dropout [43] is used in all the fully-connected layers but the last one with a rate of 0.5. The weights in each convolutional layer are initialized from a zero-mean Gaussian distribution with standard deviation of 0.01. For the weights in each fully-connected layer, the standard deviation is set to 0.005. The biases in the second convolutional layer and the last fully-connected layer are initialized with the constant 1. The remaining layers are initialized with the constant 0.

6.3. Results

After 65,000 iterations, we get a validation accuracy of 98.75% (49 wrong instances). The test accuracy is 98.57% (56 wrong instances). The validation accuracy vs. training iterations is shown in Fig. 7. The result is comparable with the previous architecture, but has much more parameters (learned filters) and requires optimization at the filter learning stage. Furthermore, this network is carefully tuned to get satisfying convergence. In the following section, we will look into the model by visualization.

6.4. Feature and filter visualization

Now we have a close look at the features and weights by visualizing them.

The activations of the three convolutional layers during the forward pass are shown in Fig. 8. The corresponding convolutional filters (weights) are illustrated in Fig. 9.

The first layer activations look relatively blobby and dense. In the following layers the activations become more sparse and localized. This is quite interpretable because we can infer the class-specific features. While in conventional hand-crafted methods, these features are often discovered and decided by experts with domain knowledge, for example, biologists.

As for the convolutional filters, the first layer is looking directly at the raw pixel data, so the filters looks relatively nice and smooth. The following two layers' weights are not as interpretable, but still well-formed. However, compared with the filters of DeepFish-SVM-aug in Fig. 4, the filters are not as nice, but outperform them in number. They both can extract appropriate features for the following network layers and eventually recognition.

7. Conclusion

In this paper, we aim to find a solution to accurate underwater object recognition. We propose an effective underwater live fish recognition framework based on a simple cascaded deep network, whose performance is comparable with our carefully designed and tuned deep CNN architecture. The features are learned from the training data, so no domain knowledge of fish is required.

Experiments on an underwater live fish dataset demonstrate that the recognition accuracy achieves the state-of-the-art. With the proposed framework, we expect to advance underwater live fish recognition research, explore underwater object recognition issues, and benefit the ocean biologists, ecologists, as well as commercial applications like fish farming. The framework we use can be easily extended to other recognition tasks. The method with which we choose net parameters and classifiers can serve as a reference. As we did not use complicated networks or any special tricks like very deep network with much more parameters, tens of times data augmentation on the whole dataset, or ensemble of models, although they usually can achieve better results. (Code is available at https://github.com/qinhongwei/deepfish-release.)

As pointed out by Bengio and LeCun in [44], end-to-end learning systems that have many parameters and few built-in

Please cite this article as: H. Qin, et al., DeepFish: Accurate underwater live fish recognition with a deep architecture, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.10.122

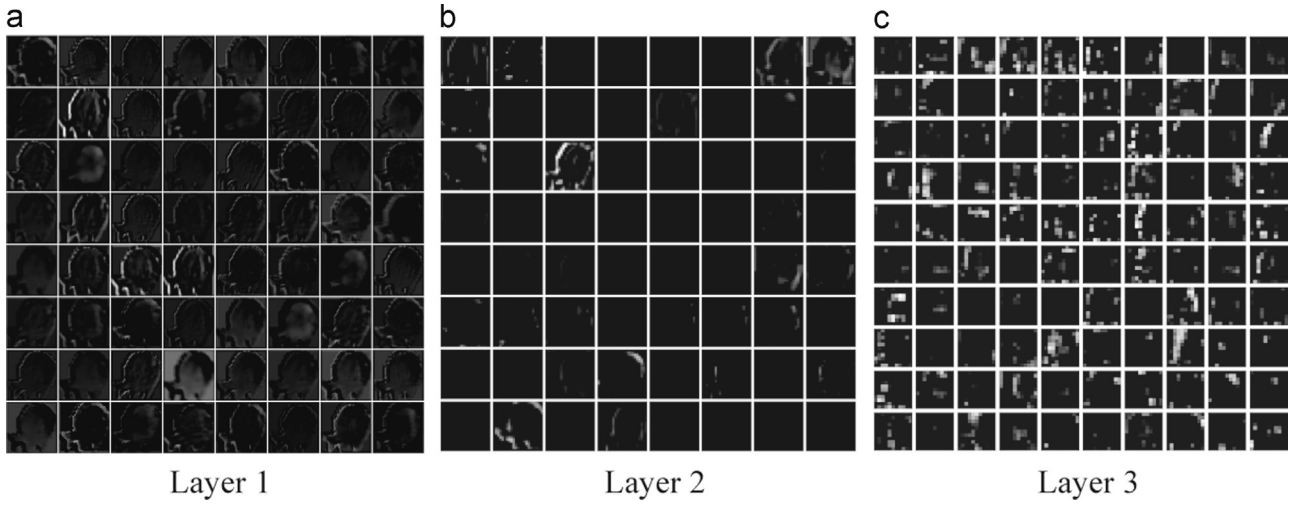


Fig. 8. A typical visualization of the CNN architecture convolutional layer activations (rectified by ReLU) given an input image of fish. Each box shows an activation map corresponding to some filter. From left to right: (a) 64 outputs of size 23×23 by the first convolutional layer (64 out of 70 are shown). (b) 64 outputs of size 11×11 by the second convolutional layer (64 out of 110 are shown). (c) 100 outputs of size 5×5 by the second convolutional layer (100 out of 180 are shown). Notice that the last two layers' activations are mostly sparse (shown in black) and localized.

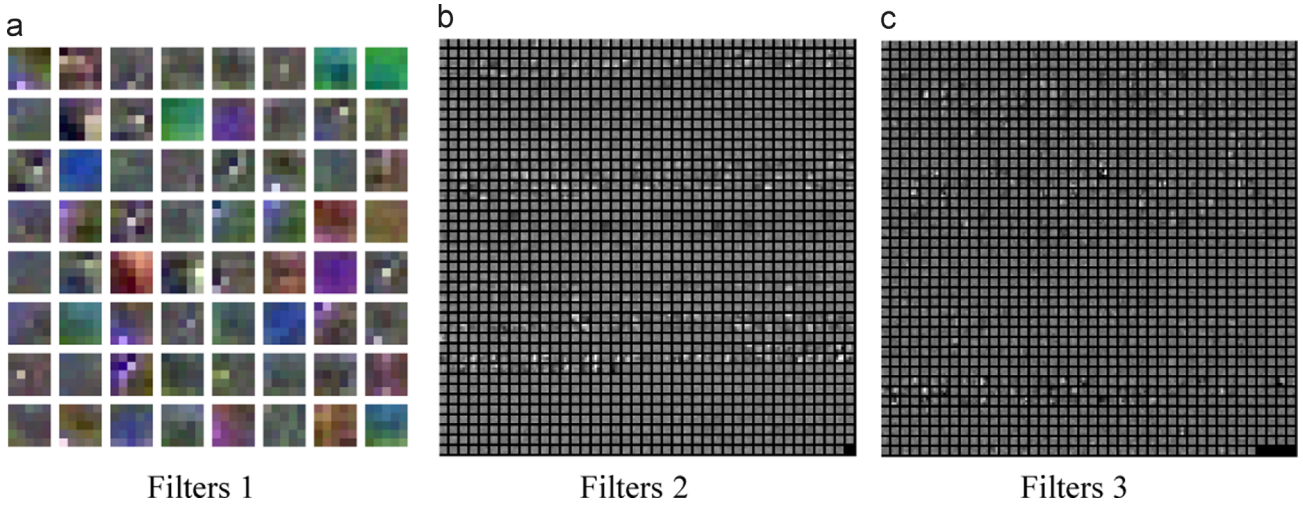


Fig. 9. Visualization of the CNN architecture convolutional layer weights. Each box shows a filter. From left to right: (a) 64 convolutional kernels of size $5 \times 5 \times 3$ learned by the first convolutional layer on the $47 \times 47 \times 3$ input images (64 out of 70 are shown). (b) 24 convolutional kernels of size $3 \times 3 \times 70$ learned by the second convolutional layer on the outputs of the previous layer (24 out of 110 are shown). (c) 16 convolutional kernels of size $3 \times 3 \times 110$ learned by the second convolutional layer on the outputs of the previous layer (16 out of 180 are shown). Notice that the kernels in (b) and (c) are reshaped for visualization.

assumptions can make use of more data and more compute power far more easily than systems that require hand-engineering of domain-specific knowledge. What we design in this paper is exactly such a system. With large and well labeled datasets, deep architectures can be powerful and effective.

In the future, we would like to figure out a way to reduce CPU memory consuming for the filter learning procedure and SVM classifier model training procedure. We also plan to write a GPU package for the architecture to further reduce time consuming for larger-scale recognition tasks.

Acknowledgements

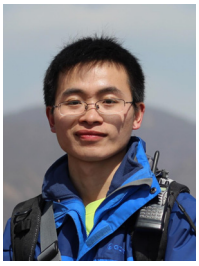
This work is supported by National Natural Science Foundation of China (Grant No. 71171121/61033005) and National 863 High Technology Research and Development Program of China (Grant No. 2012AA09A408).

References

[1] M.R. Heithaus, L.M. Dill, Food availability and tiger shark predation risk influence bottlenose dolphin habitat use, Ecology 83 (2) (2002) 480–491.
[2] A. Rova, G. Mori, L.M. Dill, One fish, two fish, butterfish, trumpeter: recognizing fish in underwater video, in: MVA, 2007, pp. 404–407.
[3] D.-J. Lee, R.B. Schoenberger, D. Shiozawa, X. Xu, P. Zhan, Contour matching for a fish recognition and migration-monitoring system, in: Optics East, International Society for Optics and Photonics, 2004, pp. 37–48.
[4] N. Strachan, P. Nesvadba, A.R. Allen, Fish species recognition by shape analysis of images, Pattern Recognit. 23 (5) (1990) 539–544.
[5] N. Strachan, Recognition of fish species by colour and shape, Image Vis. Comput. 11 (1) (1993) 2–10.
[6] D. White, C. Svellingen, N. Strachan, Automated measurement of species and length of fish by computer vision, Fish. Res. 80 (2) (2006) 203–210.
[7] R. Larsen, H. Olafsdottir, B.K. Ersbøll, Shape and texture based classification of fish species, Image Anal., 2009, 745–749.
[8] C. Spampinato, D. Giordano, R. Di Salvo, Y.-H.J. Chen-Burger, R.B. Fisher, G. Nadarajan, Automatic fish classification for underwater species behavior understanding, in: Proceedings of the First ACM International Workshop on Analysis and Retrieval of Tracked Events and Motion in Imagery Streams, ACM, Firenze, Italy, 2010, pp. 45–50.

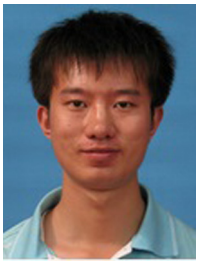
Please cite this article as: H. Qin, et al., DeepFish: Accurate underwater live fish recognition with a deep architecture, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.10.122

[9] P.X. Huang, B.J. Boom, R.B. Fisher, Underwater live fish recognition using a balance-guaranteed optimized tree, in: Computer Vision–ACCV 2012, Springer, Daejeon, Korea, 2013, pp. 422–433.
[10] P.X. Huang, B.J. Boom, R.B. Fisher, Gmm improves the reject option in hierarchical classification for fish recognition, in: 2014 IEEE Winter Conference on Applications of Computer Vision (WACV), IEEE, Steamboat Springs CO., USA, 2014, pp. 371–376.
[11] G. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, Neural Comput. 18 (7) (2006) 1527–1554.
[12] Y. Bengio, A. Courville, P. Vincent, Representation learning: a review and new perspectives, IEEE Trans. Pattern Anal. Mach. Intell. 35 (8) (2013) 1798–1828.
[13] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, Y. Ma, PCANet: a simple deep learning baseline for image classification?, arXiv preprint arXiv:1404.3606.
[14] Y. Taigman, M. Yang, M. Ranzato, L. Wolf, Deepface: closing the gap to human-level performance in face verification, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2013, pp. 1701–1708.
[15] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q.V. Le, et al., Large scale distributed deep networks, in: Advances in Neural Information Processing Systems, 2012, pp. 1223–1231.
[16] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, Stateline NV, USA, 2012, pp. 1097–1105.
[17] J. Jin, K. Fu, C. Zhang, Traffic sign recognition with hinge loss trained convolutional neural networks, IEEE Trans. Intell. Transp. Syst. 15 (5) (2014) 1991–2000. http://dx.doi.org/10.1109/TITS.2014.2308281.
[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: a large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2009, IEEE, Miami, FL, USA, 2009, pp. 248–255.
[19] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[20] I.J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, Y. Bengio, Maxout networks, arXiv preprint arXiv:1302.4389.
[21] H. Lee, R. Grosse, R. Ranganath, A.Y. Ng, Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations, in: Proceedings of the 26th Annual International Conference on Machine Learning, ACM, Montreal, Canada, 2009, pp. 609–616.
[22] D. Tao, X. Lin, L. Jin, X. Li, Principal component 2-d long short-term memory for font recognition on single Chinese characters, IEEE Trans. Cybern.
[23] J. Bai, Y. Wu, J. Zhang, F. Chen, Subset based deep learning for rgb-d object recognition, Neurocomputing (2015) 280–C292.
[24] J. Yang, K. Yu, Y. Gong, T. Huang, Linear spatial pyramid matching using sparse coding for image classification, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2009, IEEE, Miami, FL, USA, 2009, pp. 1794–1801.
[25] J. Yu, D. Tao, M. Wang, Adaptive hypergraph learning and its application in image classification, in: IEEE Trans. Image Process. 21 (7) (2012) 3262–3272.
[26] J. Yu, Y. Rui, Y.Y. Tang, D. Tao, High-order distance-based multiview stochastic learning in image classification, IEEE Trans. Cybern. 44 (12) (2014) 2431–2442.
[27] F. Perronnin, J. Sánchez, T. Mensink, Improving the fisher kernel for large-scale image classification, in: Computer Vision–ECCV 2010, Springer, Heraklion, Crete, Greece, 2010, pp. 143–156.
[28] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556.
[29] K. Chatfield, K. Simonyan, A. Vedaldi, A. Zisserman, Return of the devil in the details: delving deep into convolutional nets, arXiv preprint arXiv:1405.3531.
[30] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, arXiv preprint arXiv:1409.4842.
[31] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: surpassing human-level performance on imagenet classification, arXiv preprint arXiv:1502.01852.
[32] H. Qin, Y. Peng, X. Li, Foreground extraction of underwater videos via sparse and low-rank matrix decomposition, in: 2014 ICPR Workshop on Computer Vision for Analysis of Underwater Imagery (CVAUI), IEEE, Stockholm, Sweden, 2014, pp. 65–72.
[33] D.G. Lowe, Distinctive image features from scale-invariant keypoints, Int. J. Comput. Vis. 60 (2) (2004) 91–110.
[34] N. Dalal, B. Triggs, Histograms of oriented gradients for human detection, in: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005, vol. 1, IEEE, San Diego, CA, USA, 2005, pp. 886–893.
[35] K. Jarrett, K. Kavukcuoglu, M. Ranzato, Y. LeCun, What is the best multi-stage architecture for object recognition?, in: 2009 IEEE 12th International Conference on Computer Vision, IEEE, Kyoto, Japan, 2009, pp. 2146–2153.
[36] K. Kavukcuoglu, P. Sermanet, Y.-L. Boureau, K. Gregor, M. Mathieu, Y.L. Cun, Learning convolutional feature hierarchies for visual recognition, in: Advances in Neural Information Processing Systems, Vancouver, Canada, 2010, pp. 1090–1098.
[37] K. Grauman, T. Darrell, The pyramid match kernel: discriminative classification with sets of image features, in: 2005 Tenth IEEE International Conference on Computer Vision, ICCV 2005, vol. 2, IEEE, Beijing, China, 2005, pp. 1458–1465.
[38] C.-C. Chang, C.-J. Lin, Libsvm: a library for support vector machines, ACM Trans. Intell. Syst. Technol. (TIST) 2 (3) (2011) 27.
[39] B.J. Boom, P.X. Huang, J. He, R.B. Fisher, Supporting ground-truth annotation of image datasets using clustering, in: 2012 21st International Conference on Pattern Recognition (ICPR), IEEE, Tsukuba, Japan, 2012, pp. 1542–1545.
[40] A. Vedaldi, B. Fulkerson, VLFeat: an open and portable library of computer vision algorithms, in: Proceedings of the International Conference on Multimedia, ACM, Firenze, Italy, 2010, pp. 1469–1472.
[41] Y. Tang, Deep learning using linear support vector machines, arXiv preprint arXiv:1306.0239.
[42] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: Convolutional architecture for fast feature embedding, in: Proceedings of the ACM International Conference on Multimedia, ACM, Orlando, Florida, USA, 2014, pp. 675–678.
[43] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1), 2014, 1929–1958.
[44] Y. Bengio, Y. LeCun, et al., Scaling learning algorithms towards ai, Larg.-Scale Kernel Mach. 34 (5) (2007).



Hongwei Qin received his B.S. degree in Automation from Tsinghua University, Beijing, China, in 2012, where he is currently working toward the Ph.D. degree in the Department of Automation. His research interests are in the areas of deep learning, computer vision and image processing.

Xiu Li received her Ph.D. degree in computer integrated manufacturing in 2000. Since then, she has been working in Tsinghua University. Her research interests include data mining, business intelligence systems, knowledge management systems and decision support systems.



Jian Liang received his B.S. degree in Automation from Huazhong University of Science and Technology, Wuhan, China, in 2012. He is currently working toward the Ph.D. degree in the Department of Automation, Tsinghua University, Beijing. His research interests are in the areas of deep learning and pattern recognition.

Yigang Peng received the bachelor's degree from Beijing University of Posts and Telecommunications, Beijing, China, in 2007, and the Ph.D. degree from Tsinghua University, Beijing, in 2012. His research interests include computer vision, image processing, and computer network.

Changshui Zhang received his B.S. degree from the Peking University, Beijing, China, in 1986, and Ph.D. degree from Tsinghua University, Beijing, China, in 1992. He is currently a Professor of the Department of Automation, Tsinghua University. He is an Editorial Board Member of Pattern Recognition. His interests include artificial intelligence, image processing, pattern recognition, machine learning and evolutionary computation.

Please cite this article as: H. Qin, et al., DeepFish: Accurate underwater live fish recognition with a deep architecture, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.10.122