Weighted Feature Pooling Network in Template-Based Recognition

Zekun Li, Yue Wu, Wael Abd-Almageed, and Prem Natarajan

Information Sciences Institute, USC, CA, USA

Abstract. Many computer vision tasks are template-based learning tasks in which multiple instances of a specific concept (e.g. multiple images of a subject's face) are available at once to the learning algorithm. The template structure of the input data provides an opportunity for generating a robust and discriminative unified *template-level* representation that effectively exploits the inherent diversity of feature-level information across instances within a template. In contrast to other feature aggregation methods, we propose a new technique to dynamically predict weights that consider factors such as noise and redundancy in assessing the importance of image-level features and use those weights to appropriately aggregate the features into a single template-level representation. We present extensive experimental results on the MNIST, CIFAR10, UCF101, IJB-A, IJB-B, and Janus CS4 datasets to show that the new technique outperforms statistical feature pooling methods as well as other neural-network-based aggregation mechanisms on a broad set of tasks.

Keywords: Template representation, feature pooling, attention network

1 Introduction

In many computer vision tasks, the input data comes in the form of grouped instances ("templates") of examples (images, videos) related to the same object or class. For example, in the case of face recognition where the task is to recognize the identity of a subject, many face recognition datasets provide multiple images of each subject, e.g., YouTube Faces (YTF) [1] and IARPA Janus Benchmarks [2] [3]. Template-based learning is a natural extension of image-based learning. When the template includes only one image, template-level recognition reduces to image-level recognition. Of course, even in the case of a single-image recognition, there may be scenarios in which it is advantageous to generate a pseudo-template by applying a sequence of augmentations to the base image using other sources of information (e.g., 3D face model) to introduce useful variability.

Template-based learning, like image-based learning, requires an appropriate mechanism to represent instances. Image-level representations or features are easy to obtain from popular deep neural networks such as VGG16 [4] or ResNet [5]. Two fundamental problems need to be overcome or accounted for in using these individual representations in template-based learning: variability across

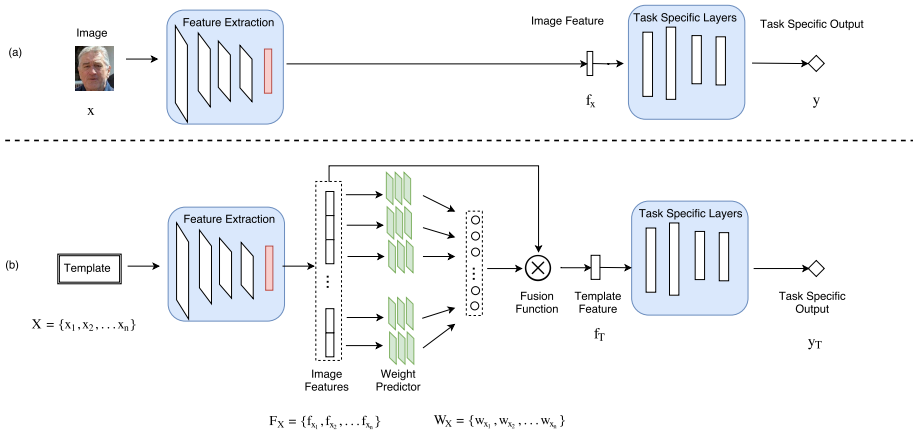


Fig. 1: WFPN can be constructed upon an arbitrary image-based learning network. (a) Image-based learning network (base model). (b) Template-based weighted feature pooling network (WFPN). The red block in feature extractor indicates split node (details in Section 3). **Weight predictor** network (green blocks) evaluates features in the same template and generates significance scores. The green blocks share the weights and have the same structure. This weight sharing technique enables WFPN to handle dynamic-sized templates. Fusion function fuses image-level features to template level feature according to the significance scores.

instances and varying numbers of instances from one template to the next. Different images in a given template may reveal different aspects of the underlying object. They may also carry redundant and noisy information [6], and images within the same template may exhibit significant diversity due to variances such as lighting conditions, alternative view angles, or different background scenes. Furthermore, the number of instances varies from one template to another; a natural way to deal with this variability is to generate a single, unified fixed-length representation which we call the "template-level representation".

The most straightforward way to generate a template-level representation is to use the average pooling (or other order statistics) over image-level features, which has been used extensively in recent works [7, 8]. Such pooling, however, ignores factors such as noise and redundancy. Importantly, the essence of average pooling is that each feature carries equal weight, which is not true in most cases. For example, certain features from high-resolution images might be more important than those from low-resolution images while other features may be equally important across both conditions. Other neural-network based approaches [9] considered the difference of features by applying convolutional masks on features, but the masking is always constant. The work that aligns most to ours [10] proposed a cascade attention network, while we used a feed-forward structure that evaluates the features only once, and we incorporated template level information. Ideally, feature pooling methods should be designed such that (1) the complementary information from different instances/images can be exploited

and irrelevant details are attenuated, and (2) the resulting representation is of a fixed-size. It is based on this observation that we propose a new network design that incorporates a dynamic weight predictor to automatically assign weights according to the importance of each feature being aggregated.

The goal of this work is to design a network that produces a discriminative, compact fixed-length representation for any given template of arbitrary size, which can be subsequently used in downstream classification or recognition tasks. We propose a generic weighted feature pooling network (WFPN) that exploits discriminative information in the templates and generates robust representations. As shown in Figure 1(b), our weighted feature pooling network directly builds upon and seamlessly integrates with the underlying image-based learning network, i.e., the base model. It is important to note that WFPN takes dynamic-sized templates as input and produces task-specific outputs. Each image inside the template goes through a feature extraction block to generate image-level features. A weight predictor module then assigns weights to the image features according to their importance, where less informative features are assigned lower weights. Image features and weights are then fed into the fusion layer to compute template representations. While we have used linear functions in this work, the fusion layer can use either linear or nonlinear aggregation functions. In our design, the predicted weights should be constrained to sum up to one. WFPN can be trained with the same loss function as in the base model.

The main contribution of this work is the introduction of a new network module (weight predictor) that explicitly and dynamically predicts the importance of image-level features for templates of arbitrary sizes. The proposed WFPN is the integration of weight predictor and the underlying base model. It can deal with various template sizes and generate a discriminative feature that represents the entire template. We quantitatively compared our network with traditional pooling methods as well as neural-network-based approaches, and experiments show superior performance of our WFPN. Furthermore, WFPN is a general end-to-end framework that can be easily extended to solve a broad class of recognition and classification problems. From a computational perspective, compared to its corresponding base models, WFPN provides a significant performance boost while being parsimonious in its addition of parameters.

The remainder of this paper is organized as follows: In Section 2 we discuss other feature pooling methods and briefly review related work. Section 3 describes the components of the weighted feature pooling network (WFPN) and show how the weight predictor network is constructed. In Section 4 training schemes and evaluation baselines are outlined. In Section 5 we evaluate WFPN on multiple datasets. More specifically, with MNIST and CIFAR-10, we show that image-based recognition problems can be easily extended to template-based problems. With IJB-A, IJB-B and Janus CS4, we show that WFPN can deal with dynamic template size on face recognition tasks. With UCF-101, we show WFPN can be applied to action recognition problems. We experimented with various depth and structures of WFPN and observed consistent improvement over statistical pooling and other attention networks.

2 Related Work

The most relevant feature pooling methods can be broadly categorized into two families—transformation-based pooling and neural network-based methods.

Average pooling and max-pooling are the most commonly used statistical pooling methods to generate template representations. Despite their simplicity, these two pooling methods have been remarkably effective in face recognition problems. Hassner *et al.* [7] used average pooling over images and features that grouped by image quality and head pose to generate template representations. Kawulok *et al.* [8] applied max pooling over feature descriptors to obtain a single representation from a collection of face images. In the above methods, pooling is performed after feature extraction, and the choice of pooling operations is essentially ad hoc. Some other works [11] [12] formulate the pooling problem as finding orbit of a set of images, where the variations inside the set can be modeled as unitary transforms. This approach is capable of capturing translation, in-plane rotation, out-of-plane rotation, and illumination changes.

Attention networks also have an aggregation process when generating context vector. Given a fixed target output, it loops over all the source hidden states and compares with the current target state to generate scores. Then it uses softmax to normalize all scores and generate attention alignment vector. Then source-side context vector can be computed from the weighted sum of source hidden states according to attention alignment vector. In visual attention networks [13–15], spatial-attention and channel-attention mechanisms are widely used for object recognition and detection, where an attention mask is learned to focus on different regions of image.

In contrast to the approaches mentioned above, our proposed network *explicitly* predicts the importance (or weights) of image features using a weight predictor model. The weight predictor model employed the attention mechanism where the attention mask is determined dynamically based on the input feature and template level information. Different from the attention networks mentioned above, our attention model has a *dynamic*-sized attention window (or attention span) instead of a fixed one. The dynamic-sized window is achieved by the weight-sharing mechanism as illustrated in Figure 1. Compared with Yang *et al.* [10], we use one single network instead of a cascaded style to perform weight prediction—thus we only need to look at the features once to determine the weights. Also, their attention-block method employed *local* feature information to learn the mapping, while we also use *template-level information* by introducing a normalization layer after the input layer, which computes template mean and variance to normalize the input data. Moreover, the weight predictor does not have ordering constraint as in [15]. In our WFPN, the significance score depends solely on the current input feature, and the network state is independent of the previous input. Meaningly, even if the ordering of input images has changed, the corresponding weights produced by the weight predictor module remain consistent. In the proposed WFPN model, feature pooling can be adapted to different datasets and employed to solve different problems with the aid of task-specific layers.

3 Template Representation using Weighted Feature Pooling Network (WFPN)

As mentioned in Section 1, template pooling can be utilized for in both image-based and template-based learning tasks. For example, in IJB-A [2] and IJB-B [3] face recognition datasets, a template is defined as a set of images and video frames that contain the face of the same person, whereas in image patch denoising problems [16], a template can be considered as a group of similar 2D image fragments. In single-image-based learning tasks where template does not exist, we can easily create a *pseudo template* from the base image using data augmentation or data adaptation techniques. For instance, in the MNIST dataset [17], we can arbitrarily rotate the images, such that the rotated copies form a pseudo template for base images.

The overall architecture of WFPN is shown in Figure 1(b). Let $X \in \mathbf{R}^{N \times l}$ be the template that contains an arbitrary number of images, where N is the number of images, and l is the length of the feature. The goal of WFPN is to compute a $X \rightarrow y_T$ mapping where y_T is the task-specific, learned representation. WFPN consists of a quintuple $\{ \mathcal{F}, \mathcal{P}, \mathcal{C}, \mathcal{M}\}$, where $\mathcal{F}$ is the underlying feature extraction module (blue block on the left side of Figure 1b), $\mathcal{P}$ is the weight predictor module (green network in the middle of Figure 1b), $\mathcal{M}$ is the fusion function that aggregates the image-level representations to produce template-level features using the weights estimated by $\mathcal{P}$, and $\mathcal{C}$ is the underlying, task-specific module (e.g., classifier or regressor). In this section, we describe the above four components of WFPN that perform the $X \rightarrow y_T$ mapping.

One of the main design goals of WFPN is to easily integrate our weight predictor (as a plug and play module) into existing network architectures, without changes to the underlying network. For every visual recognition task, we assume there is an underlying image-based learning network, i.e., base model, that has been developed specifically for the given task, as shown in Figure 1(a). Since deep learning-based recognition methods generally learn feature extraction along with the recognition task [18], a typical image-based recognition network can be conceptually divided into two parts: feature extraction $\mathcal{F}$ and task-specific layers $\mathcal{C}$ (e.g., classification layers). Therefore, we can define **split node** as the intermediate layer that separates the two parts in the original recognition network. The segment from the input layer to split node will be referred to as the feature extraction block $\mathcal{F}$, and the rest of the network will be thought of as task-specific layers $\mathcal{C}$. The output of the split node can be seen as the feature representation of the input image—as shown in Figure 1(a). The choice of the split node is not deterministic. Often, the split node is a fully connected layer or global average pooling layer that produces 1D representation. Figure 1(b) shows an example of constructing WFPN from the base model using the above splitting scheme. Built upon the base model, WFPN uses a weight predictor $\mathcal{P}$ to predict the importance of image-level features, and a fusion layer $\mathcal{M}$ takes predicted weights together with image features to compute the final template representation.

Assume object X has a template size N (N can be different among different objects), given the group of images $X = \{x_1, x_2, ...x_N\}$ that belong to the same

template, feature extraction layers are applied to get image features, $F_X = \{f_{x_1}, f_{x_2}, ...f_{x_N}\}$, and we want to obtain a template feature f_T that represents all the features in F_X. Those image-level features $F_X = \{f_{x_1}, f_{x_2}, ...f_{x_N}\}$ are fed into the weight predictor module, which is essentially a regression network, to produce corresponding weights $W_X = \{w_{x_1}, w_{x_2}, ..., w_{x_N}\}$. Basically, input features are treated separately by the same network such that each input feature generates a weight value associated with it. In this way, we are able to handle arbitrary template size since all the operations are applied feature-wise instead of template-wise. Finally, the fusion layer computes $f_T = \sum_{i=1}^{N} w_{x_i} f_{x_N}$, which is the template-level feature representation. Note that $\sum_i^N w_{x_i} = 1$.

The weight predictor module is composed of multiple fully connected layers interpolated with drop-out layers (the number of hidden units and dropout rates that may vary depending on the specific task). The last fully connected layer uses *3D* softmax activation, since we want the predicted weights to ensure $\sum_i^N w_{x_i} = 1$. All layers are wrapped in a time-distributed layer such that each input feature produces a corresponding weight, as shown in Figure 1(b). Formally, without loss of generality, suppose the weight predictor has L fully-connected layers, and $x_i \in \mathbb{R}^M$. Then the weight score w_{x_i} can be computed from the following steps.

$$a_{x_i}^{(1)} = W^{(1)} f_{x_i} + b^{(1)} \tag{1}$$

$$g_{x_i}^{(1)} = max\{0, a_{x_i}^{(1)}\} \tag{2}$$

$$... \tag{3}$$

$$a_{x_i}^{(L)} = W^{(L)} g_{x_i}^{(L-1)} + b^{(L)} \tag{4}$$

$$w_{x_i} = \frac{exp(a_{x_i}^{(L)})}{\sum_j exp(a_{x_j}^{(L)})} \tag{5}$$

Notice that $W^{(L)}$ is of shape $(1 \times D)$ where D is the dimension of $g_{x_i}^{(L-1)}$. This constraint ensures that w_{x_i} is scalar. Equation 5 guarantees that the output is a probability distribution.

The above weight predictor network is wrapped in a time-distributed layer which reads in a template tensor $X \in \mathbb{R}^{N \times M}$ and performs the operations (1)-(4) on each slice of X, namely x_i, and their output will be softmax normalized. Each $x_i \in X$ is processed by $\mathcal{P}$, and the parameters of $\mathcal{P}$ are shared by all x_i, $\forall i \in \{1, 2, ...N\}$. The weight predictor looks at each image feature individually and predicts a value that expresses the significance of this feature. As shown above, the weight predictor parameters $\{W^{(1)}, ..., W^{(N)}\}$ are independent of template size. Thus we can use dynamic template size during training and testing, as long as the template size is fixed within each batch.

In some cases, we might want to harness template-level information to assist the weight prediction of each single features. A straightforward way to achieve this is to add a normalization layer before feeding the features into weight predictor, such that features are normalized with the template mean and the template variance. Template-level information is then encoded into image-level features.

4 Training Scheme and Evaluation Methodology

In order to accurately evaluate the contribution of weight predictor versus the underlying feature extraction and classification modules, we first pre-train the base model (see Figure 1(a)) and then initialize WFPN with pre-trained weights, such that feature extraction and classification blocks in these two models have the learned weights. We further freeze those weights and train the rest of layers in WFPN with randomly initialized weights. The loss is computed using the output of task-specific layers. For example, in classification tasks, we use categorical cross-entropy as our loss function, as shown in Equation 6.

$$L(p,q) = -\sum_{x} p(x) log(q(x)) \tag{6}$$

The error is back-propagated end-to-end, although only the weight predictor weights will be updated during the process because we have frozen the feature extraction and classification module.

In terms of evaluation, we compare WFPN performance with three baseline methods: single image classification without pooling, average pooling, and majority voting. For the single-image evaluation, we use the base model as shown in Figure 1(a) for classification, without template synthesis. For average pooling, we simply average over all features to obtain the template feature $f_T = \frac{1}{T}\sum_{i=1}^{T} f_{x_i}$ and do classification. In majority voting (or hard voting), we do not compute any template feature anymore. Instead, for feature vector $F_X = \{f_{x_1}, f_{x_2}, ...f_{x_T}\}$, we predict a label vector $Y_X = \{y_{x_1}, y_{x_2}, ...y_{x_T}\}$, which each y_{x_i} is the predicted label of f_{x_i} by the classification network. We then carry out voting using Y_X and mark the template label $\tilde{y}_x$ to be the label that received maximum votes. $\tilde{y}_x = argmax_j \sum_i \lambda_{ij} y_{x_i}$, where $j \in C$ is the class label and $\lambda_{ij} = [y_{x_i} = j]$.

5 Experimental Results

5.1 Influence of Template Synthesis Scheme on WFPN

In this section, we study the influence of the template synthesis scheme on WFPN performance and show how to apply the weighted feature pooling network on the MNIST [17] dataset, which is naturally a single-image dataset.

MNIST is a database of handwritten digits. It has 60,000 training images and 10,000 testing images. We randomly split the training set into training and validation sets with ratio 5:1. Since all the digits in MNIST were orientation-normalized and centered in a fixed-size image, in order to evaluate the performance of WFPN on different dataset distributions, we need first to create some variations of MNIST that have distinctive distributions. We perturb MNIST by randomly rotating the images by an angle $\Theta = \{\theta_1, \theta_2, ..., \theta_M\}$, where $\theta_i \in U(-\phi, \phi)$. We refer to this perturbed dataset as MNIST-ROTϕ. All the results in this section are reported on MNIST-ROTϕ instead of the original MNIST dataset. By setting $M = 10$, we will have MNIST-ROTϕ containing 500,000 training, 100,000 validation, and 100,000 testing samples.

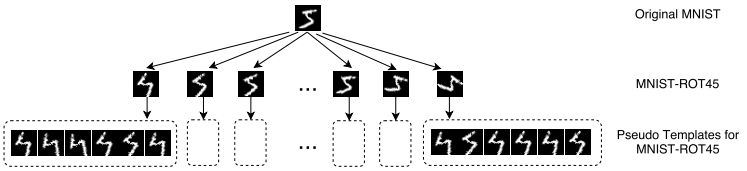
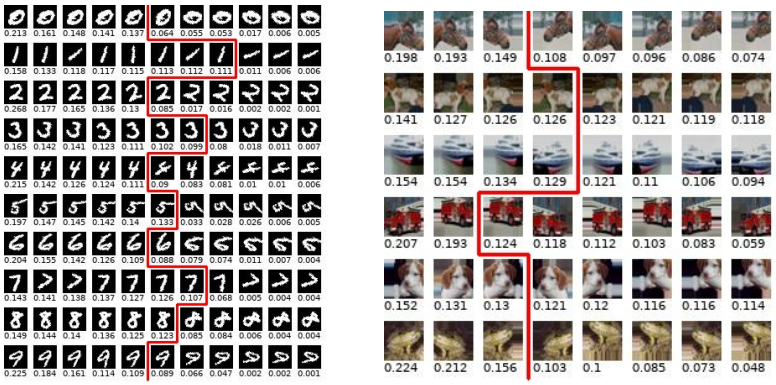


Fig. 2: MNIST pseudo template generation. First augment original MNIST dataset through rotation to generate MNIST-ROTϕ, then generate pseudo templates for MNIST-ROTϕ for different ϕ angles.



(a) MNIST (b) CIFAR10

Fig. 3: WFPN-predicted weights on MNIST and CIFAR-10 (base model: Resnet18). Note that weight prediction is performed on features. The red line splits template images into two parts: images on left side have weights above average, images on right side have weights below average.

Since MNIST-ROTϕ does not have the concept of *template*, we need to synthesize the pseudo templates in a meaningful way. Figure 2 illustrates how templates are generated from MNIST-ROTϕ images. An image x in MNIST-ROTϕ is augmented with a sequence of rotations with angle $\Omega = \{\omega_1, \omega_2, ...\omega_T\}$, where $\omega_i \in U(-\phi, \phi)$ and T is the size of the synthesized template. We can use either dynamic or fixed template size for training, and we use template size {8,11,20} for testing to better evaluate the performance of WFPN with controlled variables. Note that the rotation angle range in augmentation should be at least the same as in perturbation because we want all the tilted digits to have a chance to rotate back to their original positions. In this paper, we use the same angle range for perturbation and augmentation.

We first train an MNIST-CNN network on MNIST-ROTϕ. We use a simple 4-layer convolutional network as the base model.[1] We pre-train MNIST-CNN on MNIST-ROTϕ for 100 epochs with batch size = 128 and get the testing result as our single image classification baseline. For template-based recognition with

[1] `https://github.com/fchollet/keras/blob/master/examples/mnist_cnn.py`

Table 1: Comparison of training with fixed/dynamic template size. For testing data of a certain tsize in (8,11,20), if training tsize is *fixed*, then the template size of training data is (8,11,20) respectively. If training tsize is *dynamic*, then the template size of training data change dynamically for each batch and is range from [8,20]. (Numbers in the table are errors, the lower the better.)

Err. (%)	tsize	8		11		20	
Rot. range	img.	fixed	dynam.	fixed	dynam.	fixed	dynam.
5	0.73	0.72	0.71	0.71	0.72	0.71	0.72
15	1.09	0.86	0.88	1.24	1.19	0.83	0.85
45	2.26	1.20	1.14	1.86	1.36	1.17	1.09
75	2.93	2.10	1.88	1.55	1.44	1.43	1.46
90	4.59	2.53	1.88	1.99	1.86	2.19	2.21

Table 2: Comparison of different template synthesizing method. With *rand. rot. angle*, we randomly sample rotation angles for testing. With *fixed rot. sequence*, we use a set of angles with a fixed interval $\Omega = \{\omega_1, \omega_2, ...\omega_T\}$ where $|\omega_{i+1} - \omega_i| = \frac{2\phi}{T}$. (Numbers in the table are errors, the lower the better.)

Err. (%)	Single image: 4.59 ($\phi = 90$)							
	rand. rot. angles				fixed rot. sequence			
tmplt.	avg.	vote.	NAN[10]	ours	avg.	vote.	NAN[10]	ours
8	4.42	8.80	3.62	**2.53**	4.89	3.91	2.42	**1.98**
11	4.12	4.31	2.83	**1.99**	4.90	3.49	2.18	**1.71**
20	9.67	10.24	3.45	**2.19**	5.12	3.33	2.28	**1.77**

WFPN, we let the split node be the second-last fully connected layer and use a weight predictor network that has three fully connected layers and three dropout layers. The number of filters is {128, 8, 1} respectively. The dropout layers have dropout rates of 0.25. As mentioned in Section 3, all of the above layers are wrapped in time distributed layers. For the choice of ϕ, we experimented with a group of angle ranges: [5, 15, 45, 75, 90]. Thus we have five different perturbed datasets. After training the MNIST-CNN network for each dataset, we obtained the testing error rates shown in Table 1 in column labeled *img*.

During training of WFPN we can either use a dynamic template size or fixed template size. When using dynamic template size, the network will learn to handle test sets with arbitrary template size. We compare the results of WFPN against our single image classification baseline. Results are shown in Table 1 (the test sets are generated using randomly sampled rotation angles). We observe that the network trained with dynamic template size has generally better performance than the ones trained with fixed size templates. Also, there are multiple ways of generating templates for testing. We can either randomly sample rotation angles, or we use a set of angles with a fixed interval. The first setting can be used to evaluate the performance of the weight predictor when template generation is random. The second setting guarantees that the existence of an image in our

synthesized template is as close to an optimal tilted position as possible. Results are shown in Table 2.

5.2 Influence of Base Model Structure on WFPN

In this section, we study the influence of base model structure on the WFPN performance using CIFAR-10 dataset. CIFAR-10 has 60,000 color images of size 32×32, with 50,000 for training and 10,000 for testing. Those images contain objects from 10 classes. We keep 5,000 for validation and use the rest to train the WFPN network.

We experimented with three base models, Resnet-18 [5], Resnet-20 [5] and adapted-VGG16 on CIFAR-10[19]. For the first two models, we let the split node be the flatten layer before the last fully connected layer. Then the classification block would be the one fully connected layer (last layer) with softmax activation. The feature extraction block consists of layers in-between the input layer and split node. The output feature dimension is 512 for Resnet-18 and 64 for Resnet-20. For the VGG16 model, the split node is again the flatten layer—thus the classification block has two fully connected layers, and the feature extraction block has 13 convolutional layers. The flatten layer serves as the output of the feature extraction block. With this structure, the feature dimension is 512. During training, we use the loss function defined in Equation 6. In the corresponding WFPN models, all weight predictors have the same structure: 3 fully connected layers with {128, 8, 1} filters; dropout layers have a dropout rate of 0.25.

WFPN is initialized from the pre-trained base model, such that feature extraction block and classification block weights are loaded. The loaded weights are frozen during the training of WFPN thus only the weight predictor module is trainable. Then the performance change will only be related to feature pooling methods instead of the fine-tuning of feature extractor and classifier.

In this experiment, we synthesize templates by randomly shifting and horizontally flipping the base image. The shifting range is [-6, +6] pixels in all directions, and shift distance is uniformly and randomly selected. The number of augmentations indicates template size. Note that the above augmentation methods can also be used to train our base models described earlier, but the purposes of augmentation are different. During training the base model, augmentation is a common trick used against over-fitting. While training the WFPN model, augmentation is used to generate templates from one seed image, and the model learns to disregard redundant or noisy information to produce compact and discriminative features.

We compared the performance of WFPN with two baselines and the NAN network in [10]. We observed that WFPN has higher accuracy than mostly all three baselines. Results are shown in Table 3.

5.3 WFPN For Face Recognition

In this section we evaluate WFPN on the face recognition task using the IJB-A[2], IJB-B [3] and Janus CS4 dataset. Our base model is ResFace101 in [20],

Table 3: Comparison of baselines and WFPN performance. WFPN consistently outperforms baselines and NAN for different network structures with different depth.

Acc. (%)	img.	tmplt.	avg. pool	voting	NAN[10]	ours
Resnet20	91.42	3	91.47	90.06	91.64	**92.49**
		5	92.09	91.59	92.09	**93.17**
		8	91.88	90.98	92.71	**93.17**
Resnet18	89.41	3	89.80	88.85	89.69	**90.02**
		5	90.66	89.97	90.36	**90.89**
		8	90.47	89.40	90.63	**90.84**
VGG16	93.59	3	93.69	93.32	93.54	**93.84**
		5	93.88	93.62	**94.11**	93.92
		8	93.66	93.39	93.88	**94.02**

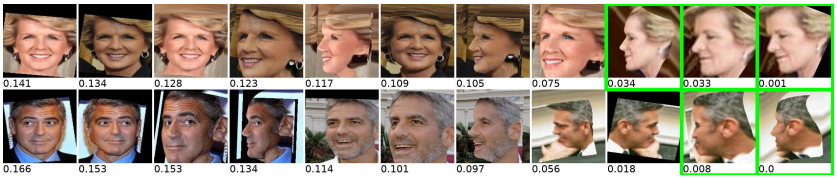
which is a deep neural network with 101 convolutional layers. Since these datasets have already pre-defined the templates, we do not need to synthesize templates anymore.

In facial recognition problems, we adapt our weight predictor module by adding one extra normalization layer after the input of features. The purpose of of this layer is to encode local information within the template, such that even with exactly the same image feature input, the absolute weight value could be different depending on its neighboring features inside the same template. The normalization layer is defined as $g(\mathbf{x_i}) = (\mathbf{x_i} - \bar{\mathbf{x}})/\sigma(X)$, where $X \in \mathbb{R}^{N \times M}$ is a template tensor with template size N and feature length M, $\sigma(X)$ denotes the variance of X, and $\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x_i}$.

For face recognition problems, we are only interested in the areas that contain faces. However, images in the uncontrolled environment have large illumination, background, and environment variations [18]. Thus necessary preprocessing steps are applied to crop out face regions, detect landmarks, and align the faces [18] [24]. The input to face recognition networks should be aligned face images cropped by tight bounding boxes.

For training, we used a combination of three datasets: Microsoft Celeb [25], Oxford VGG face [22] and CASIA WebFace dataset [26]. The combined data set has a total of 6,641,205 images from 68,906 subjects, and we split them into three parts — train, val, and test. Training and validation have 6,597,046 images, and the remainder is used for testing. The advantage of using this combined dataset is the huge amount of data. However, this dataset does *not* have the concept of template thus we need to synthesize templates for training weight predictor. In this case, we cannot simply use augmentations as we did for MNIST and CIFAR datasets since synthesized templates from one single image do not provide much information about the subject. Instead, we should select different images for one subject and group them as a template. Since we want our model to be able to handle *dynamic* template size during testing time, we should also train our model with dynamic size templates. In order to decide the appropriate template size, we further studied the IJB-B gallery set and analyzed its template

(a) WPFN assigns features from low resolution images less weights



(b) WPFN assigns features from bad augmentations less weights

(c) WPFN is able to distinguish outliers

Fig. 4: Weights visualization of IJB-B.

size distribution. We observed that the template size could be fitted using an exponential distribution. Therefore, during training, we determine the template size dynamically according to the exponential probability of mass distribution. The loss function we use is defined in Equation 6.

The trained network was testing on the IJB-A, IJB-B and CS4 datasets without fine-tuning. There are 1845 subjects in IJB-B dataset with still images, video frames, and videos collected from the web. CS4 has 3548 subjects with a total of 23,221 templates. Evaluation is performed using the still images and keyframes protocol. Results are shown in Table 5. The last two columns are the absolute improvement and relative improvement respectively. We can better understand the effect of WFPN by visualizing the images and their associated weights. Figure 4 illustrates that WFPN has favorable properties.

5.4 WFPN for Action Recognition

In this experiment we apply WFPN on UCF-101 dataset [27], an action recognition dataset that has 101 classes of human actions with 13320 videos. We followed the same pre-processing steps as in [28] and used RGB stream and optical flow

Table 4: Comparison of WFPN and state-of-the-art networks on IJB-A (*: result reported in [21], †: NAN single attention model ,‡: NAN cascade model)

Method	1:1 Verification TAR@FAR			1:N Identification TPIR@FPIR			
	1e-3	1e-2	1e-1	1e-4	1e-3	1e-2	1e-1
VGG-Face* [22]	-	0.805	-	-	-	0.461	0.670
Triplet [23]	0.813	0.900	0.964	-	-	0.753	0.863
Multi-pose[18]	-	0.787	0.911	-	-	0.876	0.954
Templt-Adapt [21]	0.836	0.939	0.979	-	-	0.774	0.882
NAN †[10]	0.860	0.933	0.979	-	-	0.804	0.909
NAN‡ [10]	0.881	0.941	0.978	-	-	0.817	0.917
Pool-Face [7]	-	-	-	0.538	0.735	0.875	-
WFPN	**0.906**	**0.954**	**0.981**	**0.878**	**0.932**	**0.966**	**0.981**

Table 5: Comparison of average pooling and WFPN on IJB-B and CS4

	IJB-B 1-N Identification				CS4 1-N Identification			
	Avg.	WFPN	Absolute	Relative	Avg.	WFPN	Absolute	Relative
TAR@FAR=0.001%	0.484	0.563	0.079	15.31%	0.584	0.651	0.067	16.11%
TAR@FAR=0.01%	0.748	0.786	0.038	15.08%	0.786	0.814	0.028	13.08%
TAR@FAR=0.1%	0.891	0.901	0.010	9.17%	0.907	0.915	0.008	8.60%
Rank 1	0.890	0.904	0.014	12.73%	0.895	0.905	0.010	9.52%
Rank 5	0.945	0.950	0.005	9.09%	0.941	0.947	0.006	10.17%
Rank 10	0.960	0.964	0.004	10.00%	0.956	0.960	0.004	9.09%

streams for our experiments. For optical flow stream, videos are processed using the TV-L1 algorithm [29] with pixel values truncated to [-20,20] and also resized with short side equals to 256. During training, we randomly crop out a (224 $\times$ 224) region spatially and select 64 consecutive frames temporally. For videos that are not long enough, we replicate the video until it has more than 64 frames, and take the first 64. For testing, we crop out the center region of size (224 $\times$ 224) and select 250 time frames. Images are loaded in [0,255] range and then scaled to [-1,1] range by multiplying a constant number 255/2 and subtracting 1. Flows are also scaled using the same procedure.

The base model we use to initialize WFPN is the Inflated 3D Convnet (I3D) in [28]. The weights are pretrained on ImageNet and Kinetics and fine-tuned on UCF101 with learning rate 0.1, mini-batch size 6, and weight decay 1e-7. I3D is a two-stream network that operates on both RGB and flow streams, and we treat each stream as the base model. In the base model, a stack of 3D convolutional filters are applied to the input, and then a spatial global average pooling is applied on top of them. The tensor has a temporal receptive field of 99 frames in input RGB stream. In the classifier, each remaining frame predicts a logit distribution of output categories, which are averaged to generate the video logit distribution. The softmax of video logit distribution gives class probability distribution. It is clear that each spatially convolved frame has equal weight

in determining the final probability distribution, which might not be optimal. Thus we can insert our weight predictor module into the classifier and perform the soft aggregation. Instead of averaging the logit distributions, the weight predictor takes in all the logit distributions produced by each spatially convolved frame and predicts the importance of each distribution for generating video logit distribution.

Results of WFPN compared with other state-of-the-art networks are shown in Table 6. We can see that WFPN outperforms its corresponding base models [28] and achieves the highest accuracy on this dataset.

Table 6: Comparison of WFPN and other methods on UCF101 dataset. (WFPN initialized from imagenet + kinetics pretrained weights, *: Keras implementation)

	Two-Strm [30]	Conv-TS [9]	TSN [31]	HiddenTSN [32]	I3D [28]	WFPN
RGB	73.0	82.61	84.5	85.7	92.2*	**94.1**
Flow	83.7	86.25	87.2	86.3	94.7*	**96.3**
Joint	88.0	90.62	92.0	92.5	96.0*	**97.8**

6 Conclusion

We present the weighted feature pooling network (WFPN) that is designed for template-based recognition problems. This network is conceptually composed of four parts: feature extractor, weight predictor, fusion function and task-specific layers. Feature extractor produces image-level features for all the images in the same template, and weight predictor predicts the significance of each image and fusion function aggregates image-level features to template-level features according to their significance. This network can extract complementary information, remove noise, and produce a compact and discriminative template feature. WFPN is lightweight and easy to generalize on many tasks such as object classification, face recognition, and video activity recognition.

Acknowledgments

This research is based upon work supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via IARPA 2014-14071600011. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purpose notwithstanding any copyright annotation thereon.

References

1. Wolf, L., Hassner, T., Maoz, I.: Face recognition in unconstrained videos with matched background similarity. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2011) 529–534
2. Klare, B.F., Klein, B., Taborsky, E., Blanton, A., Cheney, J., Allen, K., Grother, P., Mah, A., Jain, A.K.: Pushing the frontiers of unconstrained face detection and recognition: IARPA Janus Benchmark A. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2015) 1931–1939
3. Whitelam, C., Taborsky, E., Blanton, A., Maze, B., Adams, J., Miller, T., Kalka, N., Jain, A.K., Duncan, J.A., Allen, K., et al.: IARPA Janus Benchmark-B face dataset. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. (2017)
4. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
5. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2016) 770–778
6. Ma, L., Lu, J., Feng, J., Zhou, J.: Multiple feature fusion via weighted entropy for visual tracking. In: Proceedings of the IEEE International Conference on Computer Vision. (2015) 3128–3136
7. Hassner, T., Masi, I., Kim, J., Choi, J., Harel, S., Natarajan, P., Medioni, G.: Pooling faces: template based face recognition with pooled face images. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. (2016) 59–67
8. Kawulok, M., Celebi, E., Smolka, B.: Advances in Face Detection and Facial Image Analysis. Springer (2016)
9. Feichtenhofer, C., Pinz, A., Zisserman, A.: Convolutional two-stream network fusion for video action recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2016) 1933–1941
10. Yang, J., Ren, P., Chen, D., Wen, F., Li, H., Hua, G.: Neural aggregation network for video face recognition. arXiv preprint arXiv:1603.05474 (2016)
11. Liao, Q., Leibo, J.Z., Poggio, T.: Learning invariant representations and applications to face verification. In: Advances in Neural Information Processing Systems. (2013) 3057–3065
12. Pal, D.K., Juefei-Xu, F., Savvides, M.: Discriminative invariant kernel features: a bells-and-whistles-free approach to unsupervised face recognition and pose estimation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2016) 5590–5599
13. Luong, M.T., Pham, H., Manning, C.D.: Effective approaches to attention-based neural machine translation. arXiv preprint arXiv:1508.04025 (2015)
14. Wang, P., Cao, Y., Shen, C., Liu, L., Shen, H.T.: Temporal pyramid pooling based convolutional neural networks for action recognition. arXiv preprint arXiv:1503.01224 (2015)
15. Sharma, S., Kiros, R., Salakhutdinov, R.: Action recognition using visual attention. arXiv preprint arXiv:1511.04119 (2015)
16. Alkinani, M.H., El-Sakka, M.R.: Patch-based models and algorithms for image denoising: a comparative review between patch-based images denoising methods for additive noise reduction. EURASIP Journal on Image and Video Processing **2017** (2017) 58

17. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86** (1998) 2278–2324
18. AbdAlmageed, W., Wu, Y., Rawls, S., Harel, S., Hassner, T., Masi, I., Choi, J., Leksut, J., Kim, J., Natarajan, P., et al.: Face recognition using deep multi-pose representations. In: Proceedings of the IEEE Winter Conference on Applications of Computer Vision. (2016) 1–9
19. Liu, S., Deng, W.: Very deep convolutional neural network based image classification using small training sample size. In: Proceedings of the IEEE Asian Conference on Pattern Recognition. (2015) 730–734
20. Masi, I., Trn, A.T., Hassner, T., Leksut, J.T., Medioni, G.: Do we really need to collect millions of faces for effective face recognition? In: Proceedings of the IEEE European Conference on Computer Vision, Springer (2016) 579–596
21. Crosswhite, N., Byrne, J., Stauffer, C., Parkhi, O., Cao, Q., Zisserman, A.: Template adaptation for face verification and identification. In: Proceedings of the IEEE International Conference on Automatic Face & Gesture Recognition, IEEE (2017) 1–8
22. Parkhi, O.M., Vedaldi, A., Zisserman, A., et al.: Deep face recognition. In: Proceedings of the British Machine Vision Conference. Volume 1. (2015) 6
23. Sankaranarayanan, S., Alavi, A., Castillo, C.D., Chellappa, R.: Triplet probabilistic embedding for face verification and clustering. In: Proceedings of the IEEE International Conference on Biometrics Theory, Applications and Systems. (2016) 1–8
24. Masi, I., Hassner, T., Tran, A.T., Medioni, G.: Rapid synthesis of massive face sets for improved face recognition. In: Proceedings of the IEEE Automatic Face & Gesture Recognition. (2017) 604–611
25. Guo, Y., Zhang, L., Hu, Y., He, X., Gao, J.: MS-Celeb-1M: A dataset and benchmark for large-scale face recognition. In: Proceedings of the European Conference on Computer Vision, Springer (2016) 87–102
26. Yi, D., Lei, Z., Liao, S., Li, S.Z.: Learning face representation from scratch. arXiv preprint arXiv:1411.7923 (2014)
27. Soomro, K., Zamir, A.R., Shah, M.: Ucf101: A dataset of 101 human actions classes from videos in the wild. arXiv preprint arXiv:1212.0402 (2012)
28. Carreira, J., Zisserman, A.: Quo vadis, action recognition? a new model and the kinetics dataset. In: Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition. (2017) 4724–4733
29. Pérez, J.S., Meinhardt-Llopis, E., Facciolo, G.: Tv-l1 optical flow estimation. Image Processing On Line **2013** (2013) 137–150
30. Simonyan, K., Zisserman, A.: Two-stream convolutional networks for action recognition in videos. In: Advances in neural information processing systems. (2014) 568–576
31. Wang, L., Xiong, Y., Wang, Z., Qiao, Y., Lin, D., Tang, X., Van Gool, L.: Temporal segment networks: Towards good practices for deep action recognition. In: European Conference on Computer Vision, Springer (2016) 20–36
32. Zhu, Y., Lan, Z., Newsam, S., Hauptmann, A.G.: Hidden two-stream convolutional networks for action recognition. arXiv preprint arXiv:1704.00389 (2017)