Exploring images with deep learning for classification, retrieval and synthesis

Liu, Y.

Chapter 2

Convolutional Fusion Networks for Image Classification

In the previous chapter we have introduced the background and research questions for this thesis. Starting with this chapter, we begin to answer the research questions with our proposed approaches. In this chapter, we address how we can develop a simple and efficient deep fusion network upon a plain CNN (RQ 1).

Despite recent advances in deep fusion networks, they still have limitations due to expensive parameters and weak fusion modules. To address this issue, we propose a novel convolutional fusion network (CFN) to integrate multi-level deep features and fuse a richer visual representation. Specifically, CFN uses 1×1 convolutional layers and global average pooling to generate side branches with adding only a few parameters, and employs a locally-connected fusion module, which can learn adaptive weights for different side branches and form a better fused feature. Moreover, we propose fully convolutional fusion networks (FCFNs) that are an extension of CFNs for pixel-level classification, including semantic segmentation and edge detection. Our experiments demonstrate that our approach can achieve consistent performance improvements for diverse tasks.

Keywords
Image classification, Convolutional neural networks, Fully convolutional networks, Adaptive fusion

2.1 Introduction

A significant progress on convolutional neural networks is increasing their depth to learn more powerful visual representations. In particular, the depth has increased from several layers (*e.g.* LeNet [3] and Alexnet [4]) to several tens of layers (*e.g.* VGGnet [7] and GoogLeNet [8]). Nevertheless, training a very deep network is extremely difficult because of vanishing gradients and degradation. To overcome this challenge, recent work in both Highway networks [9] and ResNet [10] proposes to add shortcut connections between neighboring convolutional layers, which are able to alleviate the vanishing gradient issue and ease the training stage. Nevertheless, it is non-tractable to optimize very deep neural networks due to their large amount of parameters and the expensive cost of physical memory.

An alternative is to explore integration with the existing intermediate layers in a deep neural network, rather than deepening the network with new layers. Commonly, the topmost activations in deep networks (*i.e.* fully-connected layers) serve as discriminative visual representations to describe the image content. However, it is important to note that intermediate activations (*i.e.* convolutional layers) can also provide informative and complementary clues about images, including low-level textures, boundaries, and local parts. Therefore, researchers [15, 16, 25] have given greater attention to intermediate layers, and evaluated their contributions regarding image recognition performance. In addition, a large number of approaches [16, 23, 24, 121] have leveraged sophisticated encoding schemes (*e.g.* BoW, VLAD and Fisher Vector) to further encode intermediate feature activations. These approaches extract deep features from off-the-shelf CNNs without training new networks.

Moreover, extensive research efforts [17, 18, 26, 31] have turned to explicitly training deep fusion networks where multi-level intermediate layers are fused together by adding new side branches. As a result, the deep fused representation allows us to integrate the strengths of individual layers and generate superior prediction. Although these deep fusion networks have achieved promising performance, they may spend a large number of additional parameters required for generating the side branches [18]. In addition, their fusion modules (*e.g.* sum pooling) do not fully consider the importance of different side branches. Motivated by this problem, this chapter focuses on the research question **RQ 1: How can we develop a simple and efficient deep fusion network upon a plain CNN?**

To address this question, we propose a convolutional fusion networks (CFN), which is a new fusion architecture to integrate intermediate layers with adaptive weights. To be specific, CFN mainly consists of three key components: (1) *Efficient side outputs*: we use efficient 1×1 convolution and global average pooling [122] to generate side branches from intermediate layers and as a result it has a small number of additional parameters. (2) *Early fusion and late prediction*: it can not only provide a richer representation, but also reduce the number of parameters, compared to the

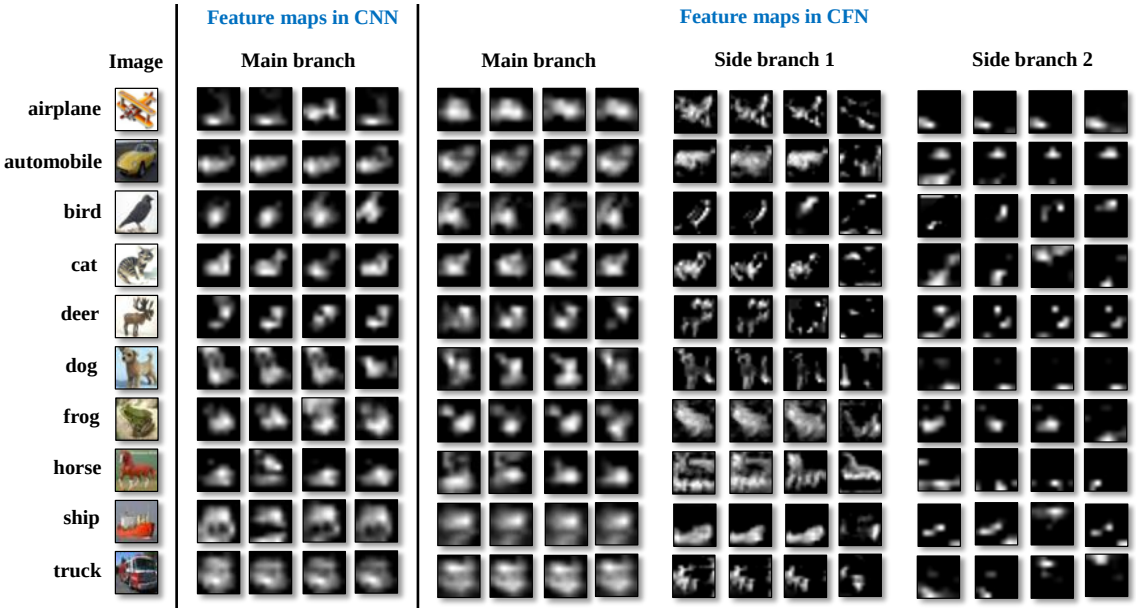


Figure 2.1: Illustration of features activations of the last convolutional layer in (a) CNN and (b) CFN. The CIFAR-10 images are used here. Compared with CNN, CFN can learn complementary clues in the side branches to the full depth main branch. For example, the side branch 1 mainly learns the boundaries or shapes around objects, and the side branch 2 focuses on some semantic "parts" that fire strong near the objects.

"early prediction and late fusion" strategy [18]. (3) *Locally-connected fusion*: we propose to adapt a locally-connected layer to act as a fusion module. It allows us to learn adaptive weights for different side outputs and generate a better fused representation. Figure 2.1 visually compares the feature activations learned in CNN and CFN, respectively. It can be seen that aggregating multi-level intermediate layers is essential to integrate their individual information.

The contributions of this chapter are as follows:

- We propose a new fusion architecture (CFN) which can provide promising insights towards how to efficiently exploit and fuse multi-level features in deep neural networks. In particular, to the best of our knowledge, this is the first attempt to use a locally-connected layer as a fusion module.

- We introduce CFN models to address the image-level classification task. The results on the CIFAR and ImageNet datasets demonstrate that CFN can achieve promising improvements over the plain CNN. In addition, we transfer the trained CFN model to three new tasks, including scene recognition, fine-grained recognition and image retrieval. By using the transferred model, we can achieve consistent performance improvements on these tasks.

- We further develop fully convolutional fusion networks (FCFN), which are able to perform pixel-level classification tasks including semantic segmentation and

edge detection. As a result, FCFN, as a fully convolutional extension, reveals the strong generalization capabilities of CFN for diverse tasks.

The rest of this chapter is organized as follows. Section 2.2 introduces the details of constructing the proposed CFN for image-level classification problem. In addition, we compare and highlight the differences of CFN from other deep models. The FCFN counterpart for pixel-level classification is described in Section 2.3. Section 2.4 presents experimental results that demonstrate the performance of CFN and FCFN on various visual recognition tasks. Finally, Section 2.5 concludes this work and point out two future directions.

2.2 Convolutional Fusion Networks

In this section, we introduce the details of building CFN on top of a plain CNN model, and formulate its training procedure. In addition, we compare its differences from other deep models.

2.2.1 Network architecture

First, we show a general architecture of a plain CNN model. As illustrated in Figure 2.2, it mainly comprises of successive convolutional layers and pooling layers. In addition, a 1×1 convolutional layer followed by global average pooling is used because of its high efficiency [8, 10, 122]. Based on this plain CNN, we can develop the proposed CFN by adding new side branches from intermediate layers and aggregating them in a locally-connected fusion module. Figure 2.3 illustrates the architecture of the proposed CFN. Our CFN is built on top of a plain CNN. To be specific, CFN mainly consists of the following three key components.

(1) Efficient side outputs

Prior work often added new fully-connected (FC) layers in the side branch [18], but this strategy may severely increase the number of parameters. Instead, CFN is able to efficiently create the side branches from the intermediate layers by adding only a few parameters. First, the side branches are built from the pooling layers (Figure 2.3). Each side branch has a 1×1 convolutional and global average pooling as well. All 1×1 convolutional layers must have the same number of channels so that they can be integrated together. Then, global average pooling (GAP) is performed over the 1×1 convolutional maps so as to obtain a one-dimensional feature vector, called the GAP feature. As a result, the side branches have the similar top layers (1×1 Conv and GAP) to the full-depth main branch. One difference is that the 1×1 Conv in the main branch follows a convolutional layer but not a pooling layer. For

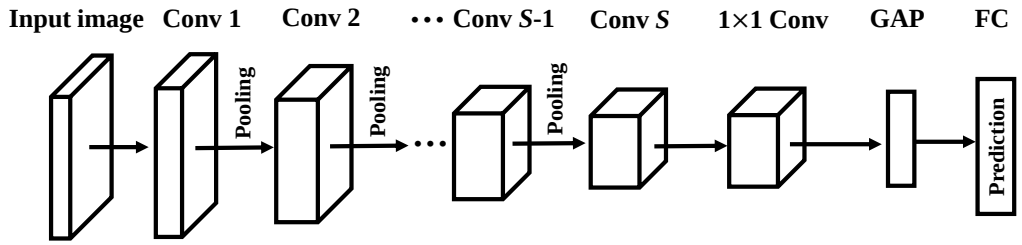


Figure 2.2: The general pipeline of a plain CNN model. Note that one 1×1 convolutional layer and global average pooling are used on the top layers.

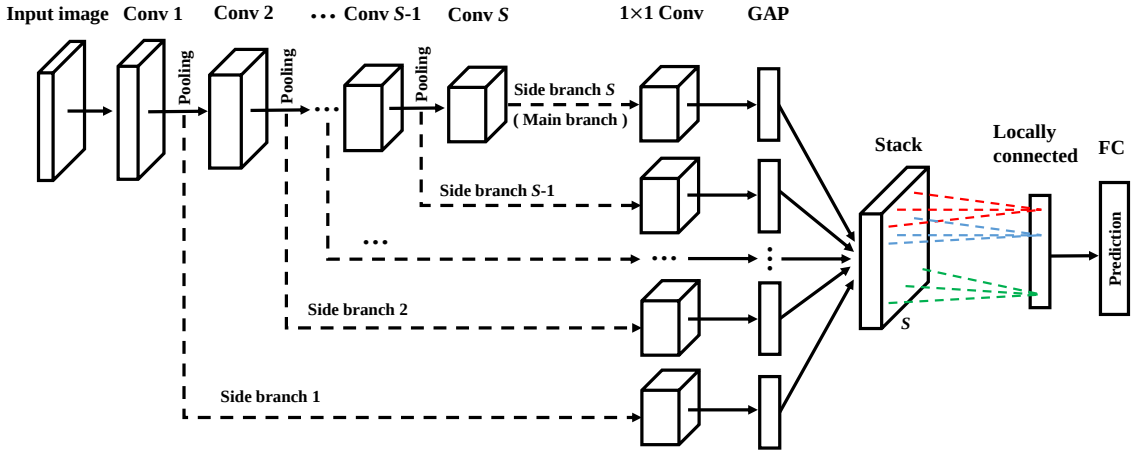


Figure 2.3: The general pipeline of the proposed CFN. First, the side branches start from the pooling layers and consist of a 1×1 convolution layer and global average pooling. Then, all side outputs are stacked together. A locally-connected layer is used to learn adaptive weights for the side outputs (drawn in different color). Finally, the fused feature is fed to the FC layer to make a better prediction.

concise formulation, we consider the full-depth main branch as another side branch.

Assume that there are S side branches in total and the last side branch (*i.e.* S-th) indicates the main branch. We notate $h_{i,j}^{(s)}$ as the input of the 1×1 convolution in the s-th side branch, where $s = 1, 2, \ldots, S$ and (i, j) is the spatial location in the feature maps. As the 1×1 convolution has K channels, its output associated with the k-th kernel is denoted as $f_{i,j,k}^{(s)}$, where $k = 1, \ldots, K$. Let $H^{(s)}$ and $W^{(s)}$ be the height and width of features maps derived from the s-th 1×1 convolution. Then, the global average pooling performed over the feature map $f_k^{(s)}$ is calculated by

$$g_k^{(s)} = \frac{1}{H^{(s)} W^{(s)}} \sum_{i=1}^{H^{(s)}} \sum_{j=1}^{W^{(s)}} f_{i,j,k}^{(s)}, \tag{2.1}$$

Where $g_k^{(s)}$ is the k-th element in the s-th GAP feature vector. We notate $g^{(s)} = [g_1^{(s)}, \ldots, g_K^{(s)}]$, a $1 \times K$ dimensional vector, as the GAP feature from the s-th side branch. $g^{(S)}$ represents the GAP feature from the full-depth main branch.

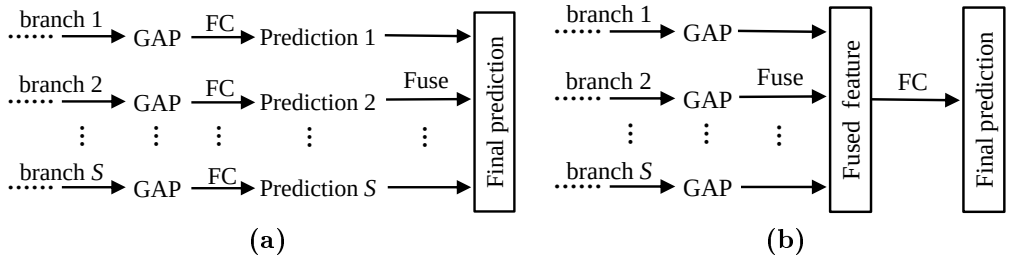


Figure 2.4: Comparison between EPLF and EFLP. (a) The schematic pipeline of EPLF strategy; (b) The schematic pipeline of EFLP strategy.

(2) Early fusion and late prediction

Considering how to incorporate the side branches, some work [18, 26, 31] used an "early prediction and late fusion" (EPLF) strategy. In Figure 2.4a, EPLF computes a prediction from the GAP feature using a fully-connected layer and then fuses side predictions together to make the final prediction. In contrast to EPLF [18], in which a couple of FC layers are added, we present another strategy called "early fusion and late prediction" (EFLP). EFLP first fuses the GAP features from the side branches and obtains a fused feature. Then, a fully-connected layer following the fused feature is used to estimate the final prediction. As seen in Figure 2.4b, EFLP has fewer parameters due to using only one fully-connected layer. Assume that each fully-connected layer has C units that correspond to the number of object categories. The fusion module has W_{fuse} parameters. Quantitatively, we can compare the number of parameters between EFLP and EPLF by

$$W_{EFLP} = K(C+1) + W_{fuse} < W_{EPLF} = SK(C+1) + W_{fuse}. \tag{2.2}$$

Hence, we make use of EFLP to fuse intermediate features earlier due to its efficiency. We observe that EFLP can achieve the same accuracy as EPLF, though EPLF contains more parameters. More importantly, the fused feature in EFLP is able to act as a richer image representation, however, EPLF cannot generate such a rich fused representation. In the experiments, we transfer the fused feature in EFLP to diverse vision tasks and show its promising generalization ability.

(3) Locally-connected fusion

Another significant component in CFN is that it employs a locally-connected (LC) layer to fuse the side branches. Owing to its no-sharing filters over spatial dimensions, LC layer can learn different weights in each local field [123]. For example, DeepFace [124] used the LC filters to learn more discriminative face representations instead of spatially-sharing convolutional filters. Differently, our aim is to adapt a LC layer to learn adaptive weights for different side branches, and generate a better

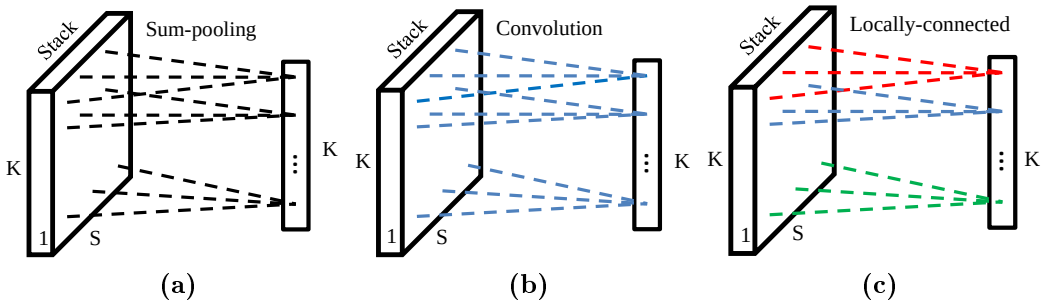


Figure 2.5: Comparison of three fusion modules. (a) No weights: Sum-pooling fusion has no weights; (b) Sharing weights: Convolution fusion learns sharing weights over spatial dimensions, as drawn in the same color; (c) No-sharing weights: Locally-connected fusion learns no-sharing weights over spatial dimensions, as drawn in different colors. To learn element-wise weights, we use the 1×1 kernel size.

fused feature. To the best of our knowledge, this is the first attempt to apply a locally-connected layer to a fusion module.

At first, we stack all GAP features together from $g^{(1)}$ to $g^{(S)}$, and form a layer G with size of $1\times K \times S$, see Figure 2.3. For example, the s-th feature map of G is $g^{(s)}$. Then, one LC layer which has K of no-sharing filters is convolved over G. Each filter has $1\times1\times S$ kernel size. Since LC is able to learn adaptive weights for different elements in the GAP features which measure the importance of the side branches, it is able to produce a better fused feature. Finally, the fused feature convolved by LC also has a $1\times K$ shape, denoted as $g^{(f)}$. The i-th element in $g^{(f)}$ is expressed:

$$g_i^{(f)} = \sigma\left(\sum_{j=1}^{S} W_{i,j}^{(f)} \cdot g_i^{(j)} + b_i^{(f)}\right), \tag{2.3}$$

where $i = 1, 2, \ldots, K$; σ indicates the ReLU activation function. $W_{i,j}^{(f)}$ and $b_i^{(f)}$ represent the weights and bias for fusing the i-th elements of GAP features from different side branches. The number of parameters in the LC fusion is $K \times (S+1)$. Using these additional parameters gives the benefit of adaptive fusion while it does not require any manual tuning.

To clearly demonstrate the advantage of the LC fusion module, Figure 2.5 compares LC fusion with other fusion methods. In Figure 2.5a, the sum-pooling fusion simply sums up the side outputs together without learning any weights, whereas this way treats each side branch equally and fails to consider their different importance. In Figure 2.5b, the convolutional fusion can learn only one sharing filter over all spatial dimensions (as drawn with the same blue color). In contrast, LC enables the fusion module to learn independent weights over each local field (*i.e.* size $1\times1\times S$) (drawn in different colors in Figure 2.5c). Although LC fusion consumes more parameters than the sum-pooling fusion (no weights) and the convolutional fusion ($S+1$),

these parameters are a negligible proportion of the total number of the network parameters.

2.2.2 Training procedure

CFN has a similar training procedure as a standard CNN, including forward pass and backward propagation. Assume a training dataset which contains N images: $\{x^{(i)}, y^{(i)}\}$, where $x^{(i)}$ is the i-th input image and $y^{(i)}$ is its ground-truth class label. W indicates the set of all parameters learned in the CFN (including the LC fusion weights). The full objective is to minimize the total loss cost

$$\text{argmin}_W \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(x^{(i)}; W), y^{(i)}), \tag{2.4}$$

where $f(x^{(i)}; W)$ indicates the predicted class of $x^{(i)}$. We use the softmax loss function to compute the cost $\mathcal{L}$. To minimize the loss cost, the partial derivatives of the loss cost with respect to any weight are recursively computed by the chain rule during the backward propagation [3]. Since the main parts in the CFN model are the side branches, we will induce the computations of their partial derivatives. We consider each image independently for notational simplicity.

First, we compute the gradient of the loss cost with respect to the outputs of the side branches. Taking the s-th side branch as an example, we compute the gradient of $\mathcal{L}$ with respect to the side output $g^{(s)}$

$$\frac{\partial \mathcal{L}}{\partial g^{(s)}} = \frac{\partial \mathcal{L}}{\partial g^{(f)}} \cdot \frac{\partial g^{(f)}}{\partial g^{(s)}}, s = 1, 2, \ldots, S. \tag{2.5}$$

Second, we formulate the gradient of $\mathcal{L}$ with respect to the inputs of the side branches. Let $a^{(s)}$ be the input of the s-th side branch. As depicted in Figure 2.3, $a^{(s)}$ corresponds to the pooling layer. However, the input of the main branch, denoted as $a^{(S)}$, refers to the last convolutional layer (*i.e.* conv S). It is important to note that the gradient of $a^{(s)}$ depends on several side branches. To be more specific, the gradient of $a^{(1)}$ is influenced by S branches; the gradient of $a^{(2)}$ needs to consider the gradient from the 2-nd to S-th branch; but the gradient of $a^{(S)}$ is updated by only the main branch. Then, the gradient of $\mathcal{L}$ with respect to the side input $a^{(s)}$ can be computed via

$$\frac{\partial \mathcal{L}}{\partial a^{(s)}} = \sum_{i=s}^{S} \frac{\partial \mathcal{L}}{\partial g^{(i)}} \cdot \frac{\partial g^{(i)}}{\partial a^{(i)}}, \tag{2.6}$$

where i indexes the related branch that contributes to the gradient of $a^{(s)}$. It needs to sum up the gradients from several side branches. As is common practice, we employ a standard stochastic gradient descent (SGD) algorithm with mini-batch [4] to train the entire CFN model.

2.2.3 Comparisons with other models

To get more insights into CFN, we compare it with other deep models.

Comparison with CNN. Typically, a plain CNN only estimates a final prediction based on the topmost layer. As a result, the effects of intermediate layers towards the prediction are implicit and indirect. In contrast, CFN connects the intermediate layers using additional side branches, and fuses them to jointly make the final predictions. In this way, CFN allows us to take advantage of intermediate layers explicitly and directly. This advantage explains why CFN is able to achieve more accurate prediction than a plain CNN.

Comparison with DSN. Deeply supervised nets (DSNs) [125] are the first model to add extra supervision to intermediate layers for earlier guidance. As a result, it can improve the directness and transparency of learning a deep neural network. Therefore, we can view DSN as a "**loss fusion**" model. Instead, CFN still uses one supervision towards the final prediction derived from the fused representation, however it is able to increase the effects of the loss cost on the intermediate layers without adding more supervision signals. In a word, we clarify that CFN is a "**feature fusion**" model. It is important to note that there is no technical conflict between CFN and DSN, so combining these two models together is a promising research direction.

Comparison with ResNet. ResNet [10] addresses the vanishing gradient problem by adding densely shortcut connections. CFN has three main differences with ResNet: (1) The side branches in CFN are not shortcut connections. They start from pooling layers and merge into a fusion module together. (2) In contrast to adding a "linear" connection in a residual block, we still use the non-linear ReLU in the side branches of CFN. (3) CFN employs a sophisticated fusion module to generate a richer feature, rather than using the simple summation employed in ResNet. As mentioned in the ResNet work, when the network is not overly deep, for example having 11 or 18 layers, ResNet may show few improvements over a plain CNN. However, CFN can obtain some considerable gains over CNN. Hence, CFN can serve as an alternative for improving the discriminative capabilities of not-very-deep models, instead of purely increasing the depth. ResNet tells us that "**depth that matters**", but CFN concludes to "**fusion that matters**".

2.3 Fully Convolutional Fusion Networks

Deep neural networks allow to bridge the gap between different vision tasks. For instance, CNN models for image-level classification can be well-adapted to other pixel-level classification tasks which aim to generate a per-pixel prediction in images. As a common practice, it is essential to cast traditional convolutional neural

networks to their corresponding fully convolutional networks (FCNs) by replacing the fully-connected layers with more convolutional layers. FCNs are able to infer any size of images without requiring specific input dimensionality. In this section, we introduce fully convolutional fusion networks (FCFN), which are used for two representative pixel-level classification tasks: semantic segmentation and edge detection. Similar to CFN, FCFN models are able to learn better pixel predictions based on the locally-connected fusion module.

2.3.1 Semantic segmentation

Semantic segmentation intends to predict a category label for spatial pixels in an image. FCN-8s [26] is a milestone model in the development of employing CNNs for semantic segmentation, and yields significant improvements in comparison with non-deep-learning approaches. First, FCN-8s is fine-tuned from the VGG-16 model [7] pre-trained on the ImageNet dataset [5]. Then, it adds two side branches to the full-depth main branch, which allow to integrate both coarse-level and fine-level pixel predictions to improve the semantic segmentation performance. Particularly, FCN-8s uses a simple sum-pooling to fuse the multi-level predictions. In contrast to FCN-8s, we extend the proposed CFN model and build the FCFN counterpart for generating fused pixel features. Moreover, we use two locally-connected layers in a two-stage fusion manner.

Recall that the locally-connected (LC) fusion module is able to learn independent weights for each spatial pixel in an image. We need to extend its formulations to be suitable for the LC fusion module in FCFN. In the first fusion module, two branches involving K channels of feature maps are taken as input. Note that the top layers are upsampled 2 times to retain the same spatial dimensions as the bottom layers. We consider the adaptive weights of each channel separately and reshape each two-dimensional feature map to a one-dimensional feature vector. For example, $g_{k,i}^{(1)}$ indicates the feature activation of the i-th pixel of the k-th channel in the first branch, and $g_{k,i}^{(2)}$ is the corresponding activation in the second branch, where $i = 1, \ldots, H \times W$ and $k = 1, \ldots, K$. Therefore, the fused pixel feature is given by

$$g_{k,i}^{(f)} = \sigma\left(\sum_{j=1}^{2} W_{k,i,j}^{(f)} \cdot g_{k,i}^{(j)} + b_{k,i}^{(f)}\right), \tag{2.7}$$

The number of parameters in this fusion module is $H \times W \times C \times 2$, where C is the number of object categories. Moreover, the second fusion module integrates coarser feature maps with the output of the first fusion module. Let $g_{k,i}^{'(1)}$ be the activation in the coarser layer. For notational simplicity, the activation $g_{k,i}^{(f)}$ from the output of the first fusion module, is renamed to $g_{k,i}^{'(2)}$. The computation in the second LC

fusion is

$$g_{k,i}^{'(f)} = \sigma\left(\sum_{j=1}^{2} W_{k,i,j}^{'(f)} \cdot g_{k,i}^{'(j)} + b_{k,i}^{'(f)}\right), \tag{2.8}$$

where $g_{k,i}^{'(f)}$ represents the final fused feature by using the two-stage fusion. Considering the computation of the loss cost with respect to the ground-truth, we still employ the softmax loss function and accumulate the loss of all pixels together.

$$\mathcal{L} = -\sum_{i=1}^{H\times W}\sum_{k=1}^{K} \mathbf{h}(y_i = k)\log p_{k,i}, \tag{2.9}$$

where y_i is the ground-truth pixel label. $\mathbf{h}(y_i = k)$ is equal to 1 when $y_i = k$, and 0 otherwise. The predicted pixel probability is normalized with the softmax function, where $p_{k,i} = \frac{\exp(g_{k,i}^{'(f)})}{\sum_{k=1}^{K}\exp(g_{k,i}^{'(f)})}$. As above, we give the loss computation for one image, but it is straightforward to extend it to a mini-batch size of images. Likewise, we use the SGD with mini-batch to train the entire FCFN model.

2.3.2 Edge detection

The problem of edge detection is to extract semantically meaningful edges in images, Typically, edge detection acts as a low-level task, but has significant contributions to other high-level visual tasks, such as object detection and image segmentation. Driven by the increasing developments of deep learning, edge features have moved from carefully-engineered descriptors such as Canny [109], gPb [126] and Structured Edges (SE) [127]), to discriminative deep features [29, 30, 31]. In particular, HED [31] is the first work to use FCNs for end-to-end edge detection, and leads to state-of-the-art performance on well-known benchmarks. HED integrates the strengths of efficient end-to-end FCNs [26] and additional deep supervision [125].

In contrast to HED that uses a convolutional fusion module, our FCFN fuses five intermediate side branches with a locally-connected layer. To be specific, one side branch generates a feature map where the activations measure the probabilities of pixels being edges. Five feature maps from side branches stack together, and are reshaped from $(H, W, 5)$ to $(H \times W, 5)$. We compute the fused prediction $g_i^{(f)}$

$$g_i^{(f)} = \sigma\left(\sum_{j=1}^{5} W_{i,j}^{(f)} \cdot g_i^{(j)} + b_i^{(f)}\right), \tag{2.10}$$

where $i = 1, \ldots, H \times W$. The sigmoid cross-entropy loss function is

$$\mathcal{L}_{fuse} = -\sum_{i=1}^{H\times W}\left[\beta_i \log g_i^{(f)} + (1-\beta_i)\log(1-g_i^{(f)})\right], \tag{2.11}$$

where the parameter β_i regulates the importance of edge and non-edge pixels, as mentioned in [31]. It is important to note that we also impose the intermediate supervision on the side branches similar to [31, 125], to discard the negative edges in the earlier intermediate layers. The loss cost in the k-th side branch (*i.e.* $k = 1, \ldots, 5$) is represented as follows

$$\mathcal{L}_{side}^{(k)} = -\sum_{i=1}^{H\times W} \left[\beta_i \log g_i^{(k)} + (1-\beta_i)\log(1-g_i^{(k)})\right], \tag{2.12}$$

where $g_i^{(k)}$ accounts for the predicted probability of the i-th pixel being an edge point. Finally, the overall loss cost in FCFN integrates a fused loss term and five intermediate loss terms together:

$$\mathcal{L} = \mathcal{L}_{fuse} + \sum_{k=1}^{5} \mathcal{L}_{side}^{(k)}. \tag{2.13}$$

This edge detection network is also fine-tuned end-to-end from the VGG-16 model and updated with the SGD algorithm with mini-batch.

2.4 Experiments

This experimental section evaluates the performance of the proposed CFN for image-level classification and FCFN for pixel-level classification. First, we train the CFN models on the datasets: CIFAR-10/100 [128] and ImageNet 2012 [5]. Then, we transfer the trained CFN model to three new tasks, including scene recognition, fine-grained recognition and image retrieval. Moreover, we train the specific FCFN models for semantic segmentation on the PASCAL dataset [129], and edge detection on the BSDS dataset [126], respectively. All experiments were conducted using the Caffe library [130] on a NVIDIA TITAN X card with 12 GB memory.

2.4.1 Image classification on CIFAR

Both CIFAR-10 [128] and CIFAR-100 [128] consist of 50,000 training images and 10,000 testing images. They define 10 and 100 object categories, respectively. We preprocessed their RGB images by global contrast normalization [131], and randomly shuffled the training set. We measure the classification performance by computing the error rates.

Table 2.1: Two plain CNN models built for the classification experiments on the CIFAR-10/100 dataset.

CNN-A	CNN-B
Input 32 × 32 RGB image	
5 × 5 × 64 conv, ReLU	3 × 3 × 96 conv, ReLU 3 × 3 × 96 conv, ReLU
3 × 3 max-pooling, stride 2. Dropout ratio 0.5	
5 × 5 × 64 conv, ReLU	3 × 3 × 192 conv, ReLU 3 × 3 × 192 conv, ReLU
3 × 3 average-pooling, stride 2. Dropout ratio 0.5	
5 × 5 × 64 conv, ReLU	3 × 3 × 192 conv, ReLU 3 × 3 × 192 conv, ReLU
1 × 1 × 192 conv, ReLU	
8 × 8 global average pooling. Dropout ratio 0.5	
10 or 100-way fully-connected layer	
Softmax classifier	

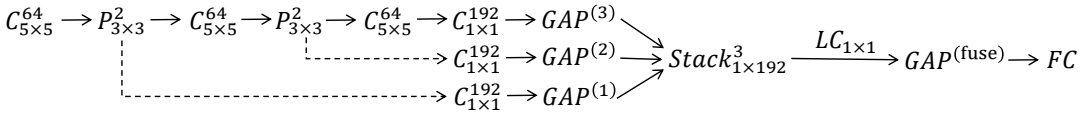
Network architecture and training details

We employ two plain CNNs models to build their CFN counterparts. Table 2.1 describes the two CNNs used for CIFAR-10/100, called CNN-A and CNN-B. (1) CNN-A is a shallow network similar to the Caffe-Quick model [130]. It has three 5×5 convolutions and a 1×1 convolution. The global average pooling is performed over the 1×1 convolutional maps. Finally, a fully-connected layer with 10 or 100 units is used to predict object categories; (2) CNN-B replaces each 5×5 convolutional layer in CNN-A with two 3×3 layers, as suggested in VGGnet [7]. In addition, CNN-B utilizes more feature channels than CNN-A. Note that, when training the CNN-B model on the CIFAR-100 dataset, the first and second convolutional layer use 192 channels instead of 96 channels. Correspondingly, the CFN-A and CFN-B models are built upon CNN-A and CNN-B respectively, by constructing two additional side branches after the pooling layers, as depicted in Figure 2.6(a) and (b).

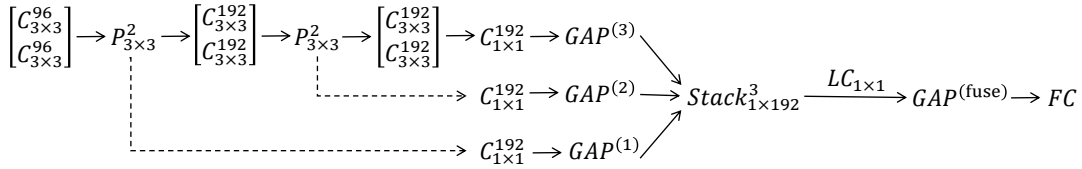
We use the same hyper-parameters to train CNN and CFN, for example, a weight decay of 0.0001, a momentum of 0.9, and a mini-batch size of 100. The learning rate is initialized with 0.1 and is divided by 10 after 10×10^4 iterations. The whole training will be terminated after 12×10^4 iterations. As for CFN, the initialized weights in the LC fusion module are set to 0.333, as there are three side branches in total (including the full-depth main branch).

Results and discussion

Table 2.2 shows the results on CIFAR-10/100 test sets. We can analyze the results considering the following three aspects:



(a) CFN-A



(b) CFN-B

Figure 2.6: Illustration of the proposed CFN models built for the CIFAR dataset. For the convolutional layers (denoted as C), the right lower number indicates the kernel size; the right upper numbers indicate the number of channels. For the pooling layers (denoted as P), the right lower numbers indicate the window size; the right upper numbers equal the size of strides.

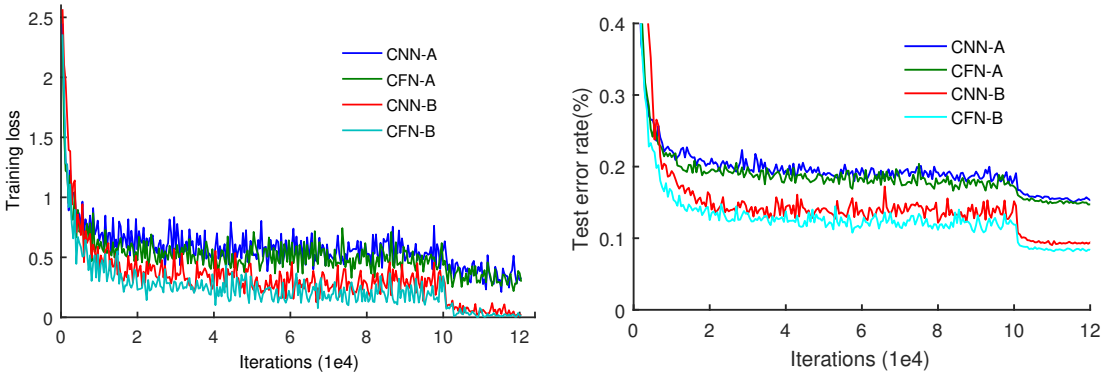
Table 2.2: Error rates (%) of image classification on the CIFAR-10/100 test set (without data augmentation). Better results are in bold face. CFNs can outperform the baseline CNNs by adding only a few parameters.

Model	#Parameters	CIFAR-10	CIFAR-100
CNN-A	0.224M (basic)	15.57	40.62
CNN-Sum-A	0.224M (basic) + 0.025M (side) + 0 (fusion)	15.33	40.32
CNN-Conv-A	0.224M (basic) + 0.025M (side) + 4 (fusion)	15.19	40.15
CFN-A	0.224M (basic) + 0.025M (side) + 768 (fusion)	**14.73**	**39.54**
CNN-B	1.287M (basic)	9.28	31.89
CNN-Sum-B	1.287M + 0.074M (side) + 0 (fusion)	8.84	31.42
CNN-Conv-B	1.287M + 0.074M (side) + 4 (fusion)	8.68	31.16
CFN-B	1.287M + 0.074M (side) + 768 (fusion)	**8.27**	**30.68**

(1) CFN achieves ∼1% improvements on the classification performance compared to the plain CNNs (both CNN-A and CNN-B). For example, on the CIFAR-10 dataset, CFN-A and CFN-B obtain 14.73 and 8.27 error rates that are ∼1% lower than the results of CNN-A and CNN-B, that are 15.57 and 9.28, respectively. The comparison between CFN and CNN demonstrates the effectiveness of fusing multi-level intermediate layers. Additionally, CFN is able to improve the expressive capabilities of deep neural networks for learning superior visual representations.

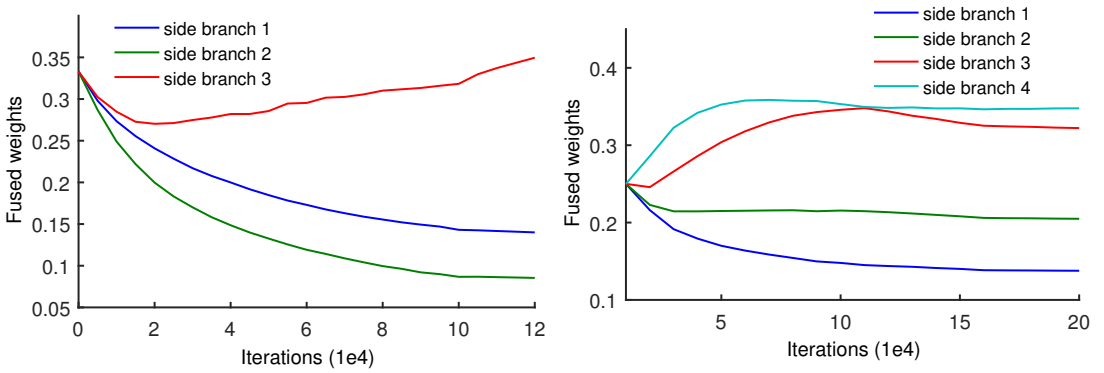
(2) In order to analyze the advantage of using the LC fusion, we also implement the existing sum-pooling fusion and convolutional fusion methods, denoted as CNN-Sum and CNN-Conv. By comparing CFN with CNN-Sum and CNN-Conv, we can observe that the LC fusion outperforms the other two fusion methods by a considerable margin. Hence, learning adaptive weights is essential to generate a better fused feature.

(3) Moreover, we compute the number of parameters in the models to estimate their efficiency. In the second column of Table 2.2, the additional number of parameters



(a) **(b)**

Figure 2.7: Comparison between CFN and CNN on the CIFAR-10 dataset. (a) The training loss when training CFN and CNN. (b) The test error rates along with the increasing iterations.



(a) CIFAR-10 **(b)** ImageNet

Figure 2.8: Illustration of adaptive weights of the side branches learned in the LC fusion. All side branches are initialized with the same weights before training. During the training stage, we can observe that the top branches have larger weights than the bottom branches.

for extra side branches and LC fusion are significantly smaller than the number of basic parameters in the models. Although the LC fusion consumes more parameters than the sum-pooling fusion and convolutional fusion, these parameters result in a minimal increase of the network complexity. In addition, we compare the training time between CNN and CFN. For example on the CIFAR-10 dataset, CNN-B and CFN-B train for approximately 1.67 and 2.08 hours, respectively.

Figure 2.7 shows the training loss and the test accuracy while training CFN and CNN. It can be seen that, both CFN-A and CFN-B models have less training loss and lower test error rates than the corresponding CNN models. In addition, Figure 2.8a presents the adaptive weights learned in the LC fusion of CFN-B. Recall that LC learns 192 filters (each filter is of size 1×3) and each filter has 1×3 weights. We compute the average weight in each branch, and estimate its fluctuation. By

Table 2.3: Test error rates on CIFAR-10/100 to compare CFN-B with other deep models. A superscripted * indicates the use of the standard data augmentation [125].

Method	Layers	CIFAR-10	CIFAR-10*	CIFAR-100
Maxout Networks [131]	5	11.68%	9.38%	38.57%
NIN [122]	9	10.41%	8.81%	35.68%
DSN [125]	9	9.69%	7.97%	34.54%
ALL-CNN [132]	9	9.08%	7.25%	33.71%
RCNN-160 [133]	6	8.69%	7.09%	31.75%
NIN + SReLU [134]	9	8.41%	6.98%	31.10%
CNN (baseline)	8	9.28%	7.34%	31.89%
CFN (ours)	8	**8.27%**	**6.77%**	**30.68%**

comparison, the side branch 3 (*a.k.a.* the full-depth main branch) plays a core role, while the other two side branches are complementary to the main branch. After a large amount of training iterations, the adaptive weights tend to be stable. Moreover, in Figure 2.1, we visualize and compare the learned feature maps in CNN-B and CFN-B. We select ten images from the CIFAR-10 dataset. The feature maps in the 1×1 convolutional layer of three side branches are extracted. We rank the feature maps by averaging spatial activations and select the top-4 maps to visualize. We can observe that CFN can learn complementary clues in the side branches, while retaining the necessary information in the main branch.

Comparison with other approaches.

Table 2.3 reports recent results on CIFAR datasets. For fair comparisons, we compare CFN-B with other not-very-deep models. Notably, "not-very-deep" is a relative concept. We use it to emphasize the differences between the models in Table 2.3 and other ResNet-like models [10]. Our method (CFN) and the compared methods develop less than 10-layer models to evaluate their effectiveness. These models certainly belong to deep neural networks, however, they are not very deep, compared to the ResNets that have more than hundreds of layers built on datasets like CIFAR-10/100. In addition, we report the depth of these models for a clear comparison and analysis. In summary, CFN obtains comparative results and outperforms these compared methods. In this work, we aim to investigate the potential of integrating multiple intermediate layers, and these results verify the effectiveness of CFN. Building CFN on top of a much deeper model (*e.g.* ResNet) is beyond the focus of our work, but it is suggestive for future research.

2.4.2 Image classification on ImageNet

The ImageNet 2012 dataset [5] consists of about 1.2 million training images, 50,000 validation images and 100,000 test images.

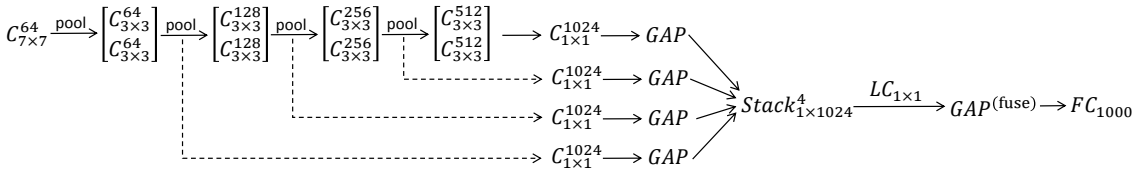


Figure 2.9: Overview of the CFN-11 architecture built on top of CNN-11. Three additional side branches are generated from the pooling layers, and fused together with the full-depth main branch (*i.e.* the last side branch).

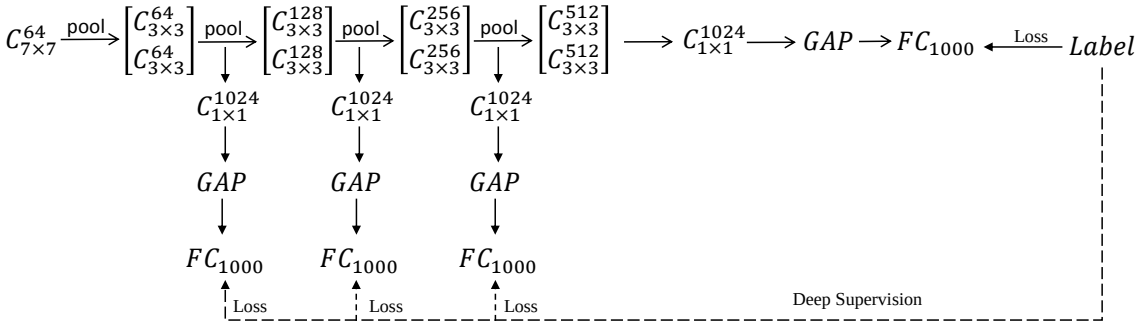


Figure 2.10: Overview of the DSN-11 architecture built on top of CNN-11. DSN-11 creates three side branches that can provide intermediate predictions for the input image. The ground-truth label is also used to guide these intermediate predictions, to enhance the discriminative abilities of hidden layers.

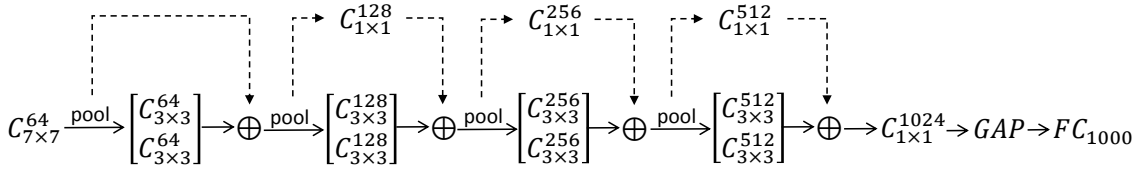


Figure 2.11: Overview of the ResNet-11 architecture built on top of CNN-11. There are four residual connections in total. Due to inconsistent numbers of channels, 1x1 convolution layers are needed in the residual connections, but they are not followed by ReLU to make sure linear transformation.

Network architecture and training details

We developed a basic 11-layer plain CNN (called CNN-11) where the channels of convolutional layers range from 64 to 1024. This baseline model is inspired by prior widely-used deep models [7, 8, 10]. Based on this CNN, we built its CFN counterpart (called CFN-11) as illustrated in Figure 2.9. Notably, we can create three extra side branches from the intermediate pooling layers (excluding the first pooling layer).

The training setup in our implementation follows the empirical practice in existing literature [4, 7, 8, 10]. The original image is resized to 256×256. In training phase, a 224×224 crop is randomly sampled from the resized image or its flipped one. The cropped input image is subtracted with per-pixel mean. We initialize the weights

and biases following GoogLeNet [8], for example a weight decay of 0.0001, and a momentum of 0.9. Batch normalization (BN) [135] is added after every convolutional layer. The learning rate starts from 0.01 and decreases to 0.001 and to 0.0001 at 10×10^4 iterations and 15×10^4 iterations respectively. The whole training will be terminated after 20×10^4 iterations. The LC weights in the fusion module are initialized with 0.25, as there are four side branches in total. We use SGD to optimize the models in a mini batch of size 64.

Results and discussion

Table 2.4 compares the results on the validation set. The following gives an analysis of the results from several aspects.

(1) CNN-11 is able to achieve competitive results when compared to AlexNet [4], however, it consumes much fewer parameters (∼6.3 millions) than Alexnet (∼60 millions). This is due to replacing several fully-connected layers with simple global average pooling.

(2) CFN-11 obtains an improvement of ∼1% over CNN-11 with adding only a few parameters (∼0.5 millions). It shows a consistent performance improvement by CFN for a large-scale dataset. Moreover, for fair comparison with other deep models, we implement the DSN-11 and ResNet-11 based on the plain CNN-11, which are shown in Figure 2.10 and Figure 2.11, respectively. It can be seen that, CFN-11 can still achieve better accuracy than DSN-11 and ResNet-11. Therefore, we can view CFN as an alternative to improve the feature representational abalities of such a not-overly deep CNN model, rather than increasing the depth as in ResNet. Notably, CFN-11 can improve CNN-11, but ResNet-11 cannot. But this does not show that CFN may be better than ResNet, as the networks are not very deep. Our primary purpose is to evaluate the superiority of CFN over CNN.

(3) To test the generalization of CFN to deeper networks, we build a 19-layer model following a similar principle as for the 11-layer model. Likewise, CFN-19 outperforms CNN-19 with ∼1% gains for both the top-1 and top-5 performance. For simplicity, we did not use the same hyperparameters as in ResNet [10], such as scale augmentation, large mini-batch size, multi-scale test. Therefore, our results of CNN-19 and CFN-19 are not as high as CNN-18 and ResNet-18 in [10]. We believe that our results can raise awareness of the potential of building deep multi-layer fusion networks. It is promising to develop much deeper networks to test the effectiveness of CFN, such as 50 or 100 layers.

Similar to CIFAR-10, Figure 2.8b illustrates the adaptive weights learned in the LC fusion of CFN-11. It is important to note that, the top branches (*i.e.* side 3 and side 4) have larger weights than the bottom branches (*i.e.* side 1 and side 2). Additionally, we extract the feature activations of the 1×1 convolutional layer in

Table 2.4: Error rates (%) regarding image classification on the ImageNet 2012 validation set.

Method	AlexNet	CNN-11	DSN-11	ResNet-11	CFN-11	CNN-19	CFN-19
Top-1	42.90	43.11	42.24	43.02	41.96	36.99	35.47
Top-5	19.80	19.91	19.24	19.85	19.09	14.74	13.93

Table 2.5: Configurations of six datasets for scene recognition, fine-grained recognition and image retrieval.

	Scene 15	Indoor 67	Flower	Bird	Holidays	UKB
#categories	15	67	102	200	–	–
#train images	1,500	5,360	2,040	5,994	991	10,200
#test images	2,985	1,340	6,149	5,794	500	10,200

one side branch. Figure 2.12 shows and compares the feature maps learned from different side branches.

2.4.3 Transferring deep fused features

To evaluate the generalization of CFN, we transfer the trained ImageNet model (*e.g.* CFN-11) to three new tasks: scene recognition, fine-grained recognition and image retrieval. Each task is evaluated on two widelly-used datasets: Scene-15 [136] and Indoor-67 [137], Flower [138] and Bird [139], and Holidays [140] and UKB [141]. The configurations of these six datasets are summarized in Table 2.5. Also, image examples are shown in Figure 2.13.

Specifically, AlexNet [4] acts as a baseline that uses the fc7 layer (4096-Dim) to provide an image representation. For CNN-11, we use the output of the global average pooling (1024-Dim) as image feature. Notably, CFN-11 allows us to utilize a fused feature (1024-Dim) that integrates multiple intermediate layers. For scene and fine-grained recognition, a linear SVM [142] is trained to compute the classification accuracy. For image retrieval, we compute the mean average precision (mAP) on Holidays and the N-S score on UKB. In terms of mAP, we take a ranked list of retrieved candidates and calculate performance based on the rank of the positive instances in the list. Given a query image, N-S is used to measure how many of the matched images are in the top-4 rank.

Results and discussion

Table 2.6 reports the transfer learning results on the six datasets. Although it is challenging to generalize a deep model to diverse visual tasks, he following summarizes the capability of CFNs in this respect.

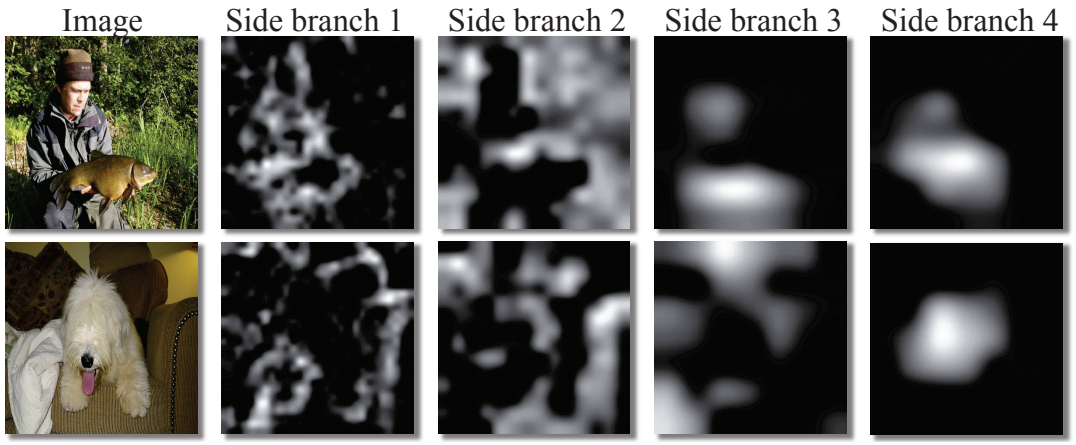


Figure 2.12: Illustration of feature maps in the four side branches. On one hand, the side branch 1 and 2 can capture some low-level clues about images, such as boundaries and textures. On the other hand, side branch 3 and 4 aim to obtain more abstract features that fire strong around objects. Therefore, CFN can incorporate multi-layer intermediate features explicitly and adaptively so as to improve visual representation.

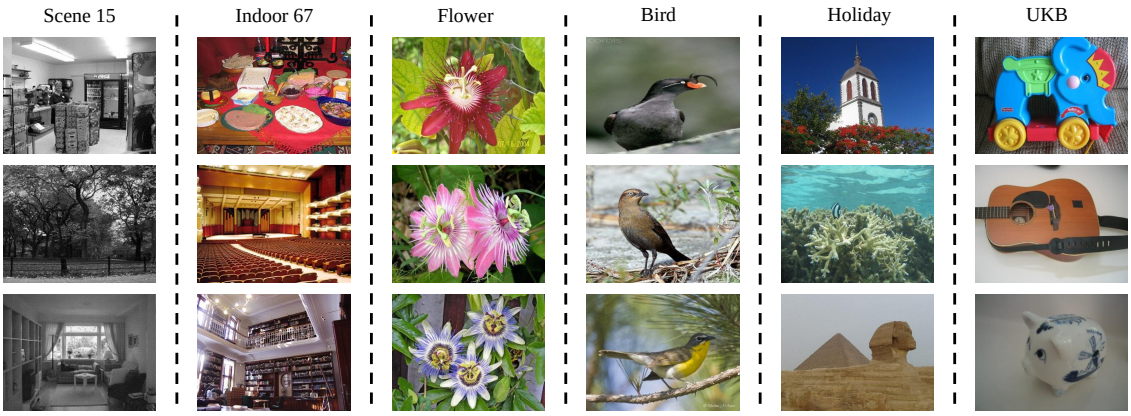


Figure 2.13: Image examples from six datasets about scene recognition, fine-grained recognition and image retrieval. We can see their significant differences with respect to the image content.

(1) Overall, CFN-11 obtains consistent improvements for the three tasks on all datasets, compared with the baseline CNN-11. In addition, CFN-11 outperforms Alexnet while using a much lower dimensional feature vector. These results reveal that learning fused deep representations is beneficial for not only image classification, but also a variety of visual tasks, even though the images in these tasks have large differences.

(2) Notably, the improvements on these three tasks are more significant than those on the ImageNet itself. In particular, CFN-11 yields a gain of $\sim$6% on the Flower dataset for fine-grained recognition. On other datasets, an accuracy improvement of $\sim$2% is obtained as well (Note that the UKB uses the N-S score which is different from precision accuracy). We believe that fine-tuning the models on the target datasets will further improve the results.

Table 2.6: Results on transferring the ImageNet model to three target tasks.

Method	#Dim	Scene recognition		Fine-grained recognition		Image retrieval	
		Scene 15	Indoor 67	Flower	Bird	Holidays	UKB
AlexNet [4]	4096	83.99	58.28	78.68	45.79	76.77	3.45
CNN-11	1024	84.32	60.45	76.79	45.98	78.33	3.47
CFN-11	1024	**86.83**	**62.24**	**82.57**	**48.12**	**80.32**	**3.54**

2.4.4 Semantic segmentation on PASCAL VOC

We conduct the semantic segmentation experiment on the PASCAL VOC 2012 segmentation dataset [129] that consists of 20 foreground object classes and a background class. The original dataset contains 1,464 training images, 1,449 validation images and 1,456 test images. When evaluating the validation set, we use a merged training dataset with the original training images and the augmented training images as in [143]. As there are validation images included in the merged training set, we need to pick the non-intersecting set of 904 images [26] as a new validation set.

We used the same hyper-parameters to train both the baseline FCN-8s [26] and the proposed FCFN, including a fixed learning rate of 10^{-4}, a weight decay of 0.0001, a momentum of 0.9, and a mini-batch size of 1. The training stage will be terminated after 100K iterations. It is worth mentioning that, we fine-tune FCN-8s directly from the VGG-16 model, without pre-training FCN-32s and FCN-16s. FCFN undergoes the same training procedure. The segmentation performance is measured with the pixel intersection-over-union (IoU):

$$IoU = \frac{TP}{TP + FP + FN}, \tag{2.14}$$

where TP, FP and FN denote the true positive, false positive and false negative counts, respectively.

Results and discussion

Table 2.7 reports the mean IoU accuracy and the detailed results of 20 object classes. The proposed FCFN achieves 1.6% gains on the mean IoU performance compared to the baseline FCN-8s. In addition, FCFN achieves superior results for more object classes, compared to FCN-8s. Figure 2.14 shows a visual example to highlight the segmentation details between the two models. We clarify that FCFN is a general architecture that can be integrated with other sophisticated techniques such as CRF [27] and Recurrent Neural Networks (RNN) [144], in order to further recover the segmentation details.

Table 2.7: Semantic segmentation results (IoU accuracy) on the PASCAL VOC 2012 validation set. For the 20 object classes, better results are in bold face.

	FCN-8s [26]	FCFN
aero	**75.5**	75.2
bike	**34.5**	33.8
bird	69.5	**72.0**
boat	**56.7**	53.3
bottle	59.7	**63.8**
bus	68.7	**71.2**
car	**70.3**	69.2
cat	73.4	**75.0**
chair	23.8	**24.0**
cow	53.0	**63.4**
table	39.7	**40.7**
dog	63.3	**65.6**
horse	46.3	**57.6**
mbike	**75.2**	74.5
person	73.9	**75.4**
plant	**42.2**	40.2
sheep	59.7	**62.3**
sofa	27.0	**30.3**
train	73.4	**74.0**
tv	**58.7**	55.7
mean	**57.1**	**60.3**

2.4.5 Edge detection on BSDS500

We evaluate the edge detection performance on the BSDS500 dataset [126] that consists of 200 training, 100 validation and 200 testing images. One image is manually annotated by five human annotators on average. The validation set is used to fine-tune the hyper-parameters, similar to HED [31]. For example, we use a momentum of 0.9 and a weight decay of 0.0002. In addition, the weights of the side-output convolutional filters are initialized with 0, and the initialization of the LC fusion filter is set to 0.2 due to fusing five side branches. The training images are resized to 400×400 and the batch size is 8. The learning rate is initialized with 10^{-6}, and the training is terminated after 25 epoches. The performance measurements for edge detection include the fixed contour threshold (ODS), the per-image best threshold (OIS) and the average precision (AP). Both ODS and OIS compute the F-score

$$F = \frac{2 \cdot precision \cdot recall}{precision + recall}. \tag{2.15}$$

Notably, ODS uses a fixed threshold to binarize all edge detection images in the test set, while OIS computes the best threshold for each image separately.

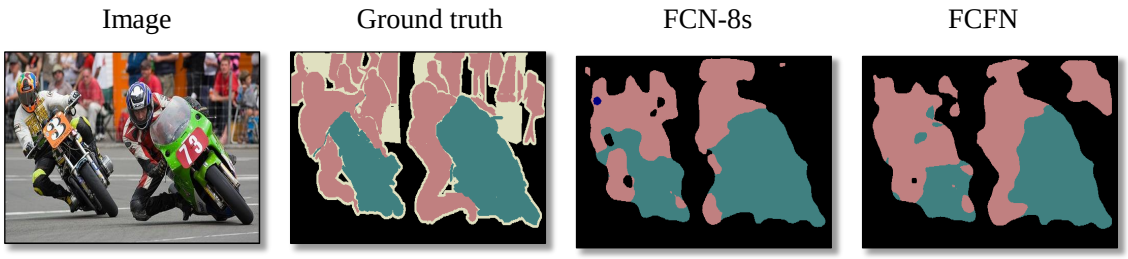


Figure 2.14: Comparison of a semantic segmentation example between the baseline FCN-8s and the proposed FCFN.

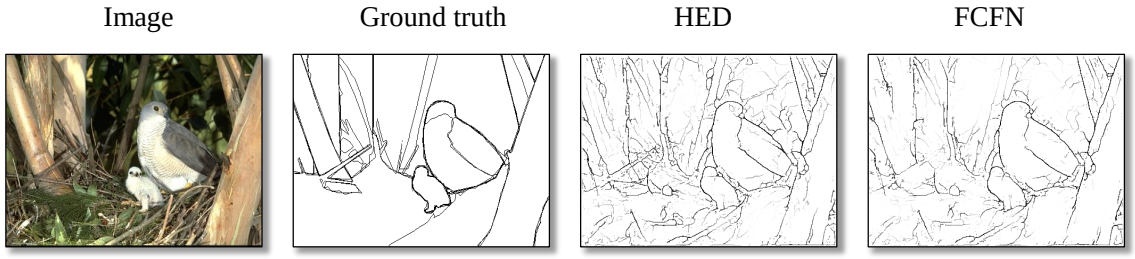


Figure 2.15: Comparison of an edge detection example between the baseline HED and the proposed FCFN. The FCFN results look more similar with the ground-truth annotations than the HED results.

Table 2.8: Edge detection results on the BSDS dataset. The upper group lists some representative approaches without using deep learning. The lower group gives the deep learning based approaches.

Method	ODS	OIS	AP
Canny [109]	.600	.630	.580
gPb-owt-ucm [126]	.726	.757	.696
SE-Var [127]	.746	.767	.803
DeepEdge [29]	.753	.772	.807
DeepContour [30]	.757	.776	.790
HED [31]	0.780	0.802	0.786
FCFN	**0.784**	**0.806**	**0.788**

Results and discussion

Table 2.8 provides a comparison of edge detection results on the BSDS dataset. First, we can see that deep learning approaches (in the lower group) largely promote the state-of-the-art performance compared to the hand-crafted edge detection approaches (in the upper group). In addition, the proposed FCFN outperforms the baseline HED with considerable improvements. This shows the advantage of learning adaptive weights in the locally-connected fusion module. Figure 2.15 shows an edge detection example and compares the visual details between FCFN and HED. FCFN can detect more satisfactory edges than HED.

2.5 Chapter Conclusions

In this chapter, we proposed a deep fusion architecture (CFN) built on top of plain CNNs. It allowed to aggregate intermediate layers with adaptive weights, and generated a discriminative feature representation. We conducted comprehensive experiments to evaluate its effectiveness for both image-level and pixel-level classification tasks. We can summarize several remarks and insights based on the experiments:

(1) On the CIFAR and ImageNet datasets, the CFN models have achieved considerable improvements while adding few parameters, even though these models are not very deep. CFN is a simple yet efficient architecture that has potential to be adapted to both deep (e.g. 10 layers) and much deeper (e.g. 100 layers) networks. In future work, we aim to build CFN on top of other deeper networks.

(2) CFN shows promising results when it is transferred to three different tasks, since CFN inherits the generalization capabilities of CNN. Additionally, CFN yields remarkable gains over CNN in the Flower dataset for fine-grained recognition. We find that it is quite important and necessary to make use of intermediate features to describe fine-grained attributes of objects.

(3) Although the FCFN models need to learn more adaptive weights in the fusion module, it can bring considerable performance improvements for semantic segmentation and edge detection. We find that many complementary details related to objects (*e.g.* boundary) are obtained from the intermediate layers.

Future work. Recall that the proposed CFN (and FCFN) model is a general extension of a plain CNN, and can be applied to a variety of visual recognition tasks. We can further improve CFN from the following two promising directions.
(1) While computing adaptive weights in the LC fusion module, we use a 1×1 kernel filter to independently consider each spatial location in the feature maps. A potential improvement would be to utilize larger kernel sizes such as 1×2 and 1×3, which can incorporate the contextual information in the feature maps.
(2) The adaptive weights are learned with the training images and are then directly applied to the test images for inference. It may be beneficial to learn input-specific weights to decrease the variance between images. Jaderberg, et al. [145] proposed a new learnable module, called the Spatial Transformer, that can perform explicit spatial transformations of features within CNNs. Similarly, Brabandere, et al. [146] proposed a Dynamic Filter Network (DFN), where filters are dynamically generated conditioned on an input image. Driven by these works, CFN can also learn dynamical filters conditioned on an input image.