Wide Residual Networks on CIFAR-10

Jia-Hua Cheng
Department of Computer Science
Texas A&M University
College Station, TX 77843
jhcheng@tamu.edu

Abstract

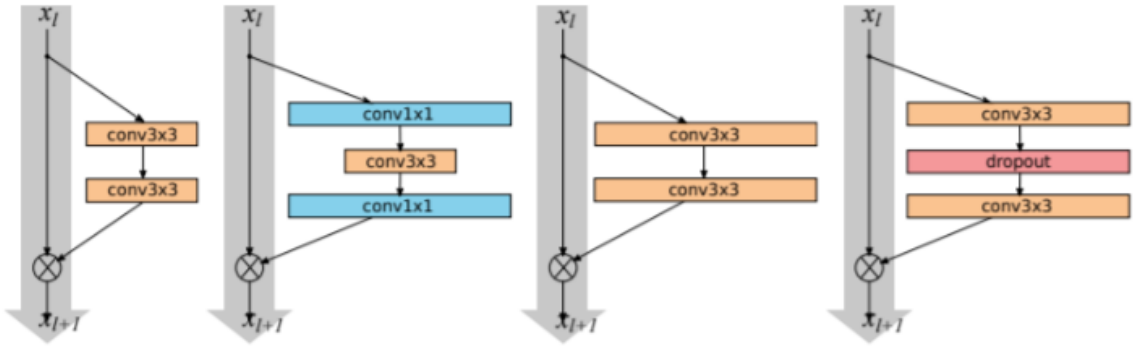
Deep residue networks have demonstrated the tremendous success on image recognition task. We have explored this architecture in homework2 on CIFAR-10 dataset. From the history of different network evolvement, it seems like we could always get benefit from making network deeper. However, it is not the case. The diminishing feature reuse causes ResNet(residual network) slow to train and it is costly to get further improvement in the deep network. In 2016, a novel network architecture, "Wide Residual Networks(WRNs)", developed by Sergey Zagoruyko, and Nikos Komodakis demonstrates the outperform accuracy that the ResNet on image classification. Besides the performance, the network itself is shallow so that the diminishing feature problem is improved. In this project, we choose WRNs to be our model. After proper training, we can achieve 96.7% accuracy on public test dataset.

1 Introduction

Convolutional neural networks(CNNs) have been proved the huge success on image recognition tasks. Training artificial neural network(ANNs) on image classification requires to convert image into 1-D array. The spatial features is lost and the trainable parameters increase exponentially with image size. Comparing to ANNs, CNNs use filters (kernels) to extract the relevant image features and learn on its own. Also, the spatial features is preserved since the arrangement of pixels and relationship between of them remain unchanged. These two huge advantages make CNNs dominate the image classification task.

Look back at history of evolvement of CNNs model. From AlexNet, VGG, Inception to Residual networks, the number of layers in the network have been increased in the past few years in terms of the return for accuracy. However, as networks going deeper, many issues happen making training more difficult. One of them is vanishing gradient which we talked in the class. Many research have been studied for training deep CNNs and there are some techniques we can apply to alleviate the problems above like initialization strategies, layer-wise training, knowledge transfer, better optimizers, and skip connections like we applied in ResNets.

Among above different architectures, Residual Networks (ResNets) had a huge success on image recognition task and won the 1st place on the ILSVRC and COCO 2015 competition. This architecture allows learned features to be utilized with better efficiency. The future work indicated



(a) basic (b) bottleneck (c) basic-wide (d) wide-dropout

that the convergence of deep networks can be speed up by using residual links. There are more relevant work aiming to make training deep networks more easier. “Deep Residual Learning for Image Recognition”, one of our reading assignment in HW2, is talking about a comprehensive study on ResNets. Up to this point, we have seen the the go-deeper mindset. But could we always get benefit from going deeper? With networks getting deeper, the “diminishing feature reuse” arises.

Identity mapping is a crucial component in ResNets architecture which allows training very deep networks. However, the problem with identity mapping is that some blocks won’t learn useful knowledges because the gradient flow may not pass these blocks. In the end, only a few blocks learn the usual features and most of the blocks contribute very little to the end result. One effective solution to this diminishing feature reuse is “dropout”. Dropout means disabling some blocks during training. We can understand this concept as enforcing some blocks being training. The paper we choose achieves great accuracy result using dropout.

The research of ResNets has focused mainly on making network deeper. In contract, the search conducted by Sergey Zagoruyko, and Nikos Komodakis proves that widening the networks could improve accuracy more effective than increasing depth. Also, their result shows that with 50 times less layers, the ResNets can achieve compatible or even better accuracy than a 1000-layer deep network and faster training time if properly trained. This architecture is named as Wide Residual Networks (WRNs).

In this project, we use WRNs training with CIFAR-10. For data augmentations, we use (1). image random flip, (2). random crop, (3). Cutout to enhance learning features. The detail model architecture will be introduced in later section. With 6 hrs of training, we achieve 95% accuracy on public test dataset.
Figure1: (a), (b) are ResNets blocks in our HW2. (c), (d) are WRNs blocks. In our project, we use (d) wide-dropout and batch normalization and ReLU precede each convolution [1].

2 Model Architecture & Augmentation Method

In this section, we give an introduction to the network architecture, image augmentation method we applied in our training model.

2.1 Architecture [1]

According to our reference paper, there are three intuitive ways to increase the representative power of model. We will give a brief comment about each idea.

- Increase filter sizes in convolutional layers
- Add more layers per block (making network more deeper)

- Widen the convolutional layers by adding more feature places

First, many successful architecture has proven that 3 x 3 filter is an effective choice for image recognition problem. We will just use 3 x 3 filter to be our filter size.

For second and third, we discuss about the benefit of making network deeper and wider. In reference paper, the authors experience different deepening factor l and widening factor k. The summary is shown in table1. We can see that the best performance for CIFAR-10 will be depth l = 28 and k = 10. Later, we will follow this setting to implement our network.

depth	*k*	# params	CIFAR-10	CIFAR-100
40	1	0.6M	6.85	30.89
40	2	2.2M	5.33	26.04
40	4	8.9M	4.97	22.89
40	8	35.7M	4.66	-
28	10	36.5M	**4.17**	20.50
28	12	52.5M	4.33	**20.43**
22	8	17.2M	4.38	21.22
22	10	26.8M	4.44	20.75
16	8	11.0M	4.81	22.07
16	10	17.1M	4.56	21.59

Table1: Test error of different depth, width networks on CIFAR-10 and CIFAR-100 [1]

Next, we will discuss about architecture of our network. We have 4 groups in our network. The structure is pretty similar to our HW2 ResNets. We can refer to a summary table (Table2) in reference paper for further explaining.

group name	output size	block type = $B(3,3)$
conv1	32×32	[3×3, 16]
conv2	32×32	$\begin{bmatrix} 3\times3, 16\times k \\ 3\times3, 16\times k \end{bmatrix} \times N$
conv3	16×16	$\begin{bmatrix} 3\times3, 32\times k \\ 3\times3, 32\times k \end{bmatrix} \times N$
conv4	8×8	$\begin{bmatrix} 3\times3, 64\times k \\ 3\times3, 64\times k \end{bmatrix} \times N$
avg-pool	1×1	$[8 \times 8]$

Table2: Summary table for our wide residual networks. Here N is a number of block in group. Block type B(3, 3) means this residual block has two 3 x 3 convolutional layers. The picture of this B(3, 3) can refer to figure1(c). [1]

Group1(conv1), is our starting layer. The detailed parameters of this convolutional layer is the following.

nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, stride=1, padding=1, bias=False)

Here, in_channels is RGB so it's 3, out_channels size is follow summary table below so it's 16. kernel_size is fixed to be 3. Since our input image size is 32 x 32, if we want to keep output size to be 32 x 32, the padding and stride are both 1.

Group2-4(conv2-4) are stacked block. Each block has 4 layers in our designed model. We will explain one block structure for simplicity. Others are the same concept.

Taking conv2 block as an example, the filter size is still 3 x 3. The input feature is from previous starting layer which is 16. The output_feature is refer to the table2 which is 32x k. In our project we choose k to be 10. Therefore, the output_feature is 320 in this block. Also from figure1(d), we have a dropout layer between 2 3 x 3 convolutional layer.

conv2 = stack_block(number_of_layer=4, input_feature=16, output_feature=320, stride=1)

This stride will be different in conv3 and conv4 in order to fulfill the output size shrinking.

stack_block is compose by the following stacked_layers.

*WRN_unit(filters_in= 16, filters_out=16*k, stride=1)*
*WRN_unit(filters_in=16*k, filters_out=16*k, stride=1)*
*WRN_unit(filters_in=16*k, filters_out=16*k, stride=1)*
*WRN_unit(filters_in=16*k, filters_out=16*k, stride=1)*

Last is average pooling layer and fully connection classification layer.

F.avg_pool(out, 8)
nn.Linear(out, num_classes)

From table2 we know the output size from conv4 is 8 x 8 so we use size 8 in our average pooling layer. The output from average pooling layer is 1 x 1. Then pass to fully connection layer with class number = 10.

2.2 Augmentation method [2], [3]

The purpose of image augmentation is to make our model more robust. Remember what we care about is the accuracy on test dataset not on our training set. Over training on the given training set will cause over-fitting, which means the model memorizes the noise and fits too close to the training set. Therefore, our model is unable to generalize to new data. To solve this problem, we could add some random noise during training so that our model won't be bias to the given data. In this project, we use three methods, and two of them are flip and crop which we already used in HW2. The third is Cutout. We will give a brief introduction in the following paragraphs.

2.2.1 Flipping

Flipping allow us to rearranges the pixels while preserving the image features. We can add this augmentation into our transform pipeline in a single line, and argument p denotes the probability of being flipped.

torchvision.transforms.RandomHorizontalFlip(p=0.5)

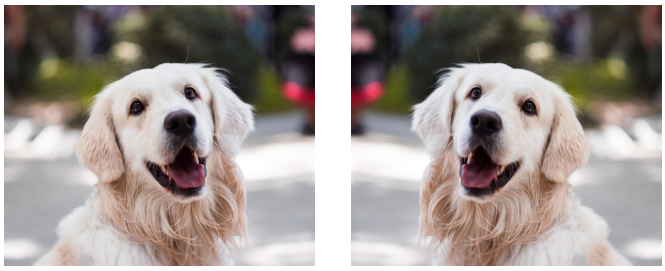


Figure2: Left: original image. Right: Flipped image.

2.2.2 Cropping

Cropping is to select a certain part of image and resize it to the original image size. The implement code is the following. Here, size denotes to the output image size after cropping, and padding indicates the white space we add to the image border.

torchvision.transforms.RandomCrop(size=32, padding=4)

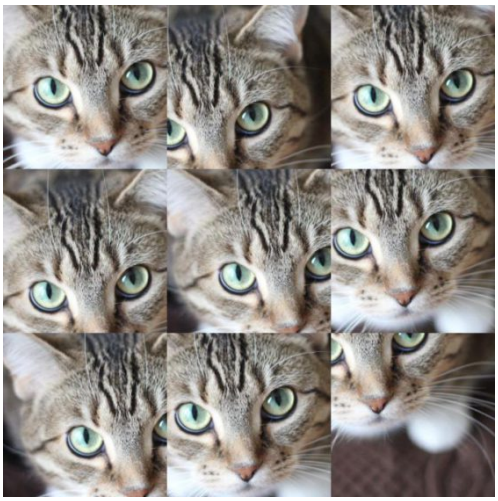


Figure3: A demonstration of cropping. LeftUpper is the original picture, and others are different cropping of the image.

2.2.3 Cutout [4]

Cutout is a simple regularization technique introduced by Terrance De Vries and Graham W. Talyor in their paper [4] in 2017. This technique is to mask out square area of input image randomly during training. Applying cutout would improve robustness and improve over-fitting problem (). Since our training and testing are all on CIFAR-10 dataset, we would like to apply cutout into our pipeline to minimize the possibility of over-fitting.

Method	C10	C10+	C100	C100+	SVHN
ResNet18 [5]	10.63 ± 0.26	4.72 ± 0.21	36.68 ± 0.57	22.46 ± 0.31	-
ResNet18 + cutout	9.31 ± 0.18	3.99 ± 0.13	34.98 ± 0.29	21.96 ± 0.24	-
WideResNet [22]	6.97 ± 0.22	3.87 ± 0.08	26.06 ± 0.22	18.8 ± 0.08	1.60 ± 0.05
WideResNet + cutout	$\mathbf{5.54 \pm 0.08}$	3.08 ± 0.16	$\mathbf{23.94 \pm 0.15}$	18.41 ± 0.27	$\mathbf{1.30 \pm 0.03}$
Shake-shake regularization [4]	-	2.86	-	15.85	-
Shake-shake regularization + cutout	-	$\mathbf{2.56 \pm 0.07}$	-	$\mathbf{15.20 \pm 0.21}$	-

Table3: The experience result of cutout in different networks (C10 means CIFAR-10). For both ResNet and WRNs we concern, cutout can improve about 1.5% on test accuracy.
The implement of cutout is provided by author's GitHub in their paper [5].

The core component of cutout is the following. We randomly pick an (x, y) to be our mask center, then calculate the corner of(x1, x2, y1, y2) then set the pixels equal to zero (white) within mask region. Last, we apply mask to input image. The resulted image is like having a hole in its image as figure4 shown.

```
def __call__(self, img):
    """
    Args:
        img (Tensor): Tensor image of size (C, H, W).
    Returns:
        Tensor: Image with n_holes of dimension length x length cut out of it.
    """
    h = img.size(1)
    w = img.size(2)

    mask = np.ones((h, w), np.float32)

    for n in range(self.n_holes):
        y = np.random.randint(h)
        x = np.random.randint(w)

        y1 = np.clip(y - self.length // 2, 0, h)
        y2 = np.clip(y + self.length // 2, 0, h)
        x1 = np.clip(x - self.length // 2, 0, w)
        x2 = np.clip(x + self.length // 2, 0, w)

        mask[y1: y2, x1: x2] = 0.

    mask = torch.from_numpy(mask)
    mask = mask.expand_as(img)
    img = img * mask

    return img
```

Figure4: the python code for cutout.

This approach aim to encourage the network to predict image with less prominent features because our mask area might be the key activated feature for our model. For example, the model for identifying dog and cat may use their nose as the key feature for prediction. By masking the nose, we encourage model to consider other different possible underlying features to make judgement. In other words, the model is forced to generalize its knowledge for different image class. Therefore, our model is more robust for unseen test data.

Figure5: Cutout applied to images from the CIFAR-10 dataset [4].

3 Experience Condition

In this section, we list down all our setting for training. Also, we compare different epoch and the benefit of our augmentation techniques.

3.1 Network Setting

In this project, we follow best preset shown in WRNs paper [1] to train our model. Here are the reverent parameters for WRN-28-10 model. (WRN-“a”-“b" denotes Wide Residual networks with depth “a" and widening factor “b”). As we mentioned in previous architecture section, the order of batch normalization, activation and convolution is BN-ReLU-conv. According to the paper, this order was shown to be faster for training and achieved better test result.

depth	*k*	dropout	CIFAR-10	CIFAR-100	SVHN
16	4		5.02	24.03	1.85
16	4	✓	5.24	23.91	1.64
28	10		4.00	19.25	-
28	10	✓	**3.89**	**18.85**	-
52	1		6.43	29.89	2.08
52	1	✓	6.28	29.78	1.70

Table4: Effect of dropout and different depth/k affect to test error % in residual block. (mean/std preprocessing, CIFAR numbers are based on median of 5 runs) [1]

3.2 Training Condition

We follow paper to use SGD with Nesterov momentum to be our optimizer and cross-entropy loss as our error function. As for learning rate, we follow paper to set 0.1. Batch_size is 128, momentum is 0.9, weight_decay is 5e-4. And in our code, we use MultiStepLR which means the learning rate will decrease after certain milestones. In our model, we set milestones to be [75, 150, 175] and decreasing factor to be 0.1. This means after reaching our preset milestone, the learning will be previous learning rate * decreasing factor. For example, the model will start from learning rate 0.1 and after epoch 75, the learning rate will be become 0.01.

The training time on our local machine is about 1.5 minutes per epoch. The total training time for 200 epochs takes about 5-6 hrs. We also compare the improvement from image augmentation and accuracy on test dataset during different epochs. The different condition and corresponding accuracy is summered as below.

Epoch	Augmentation	Test accuracy	Epoch	Augmentation	Test accuracy
0	X	60.4%	0	V	40.4%
10	X	84.6%	10	V	78.4%
20	X	80.4%	20	V	85.6%
50	X	85.6%	50	V	85.6%
100	X	92.7%	100	V	94.7%
150	X	92.2%	150	V	96.0%
200	-	-	200	V	96.7%

Table 5: Summary of different experience condition in this project.

4 Result

The test accuracy of our trained model with image augmentation on public test data is 96.7%.
Comparing to the model without augmentation, we improve almost 4% from simply doing flip, cropping and cutout (from ~92% to ~96%). Also, the test accuracy from model with data augmentation is more robust for unseen data. Here, we will use this trained model to test private dataset for final submission.

Reference

[1] Wide Residual Networks (https://arxiv.org/abs/1605.07146)
[2] AI Multiple (https://research.aimultiple.com/data-augmentation-techniques/)
[3] Nanonets (https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/)
[4] Improved Regularization of Convolutional Neural Networks with Cutout (https://arxiv.org/pdf/1708.04552.pdf)
[5] Cutout python code (https://github.com/uoguelph-mlrg/Cutout/blob/master/util/cutout.py)