

# Deep learning

## 6.5. Residual networks

François Fleuret

<https://fleuret.org/dlc/>



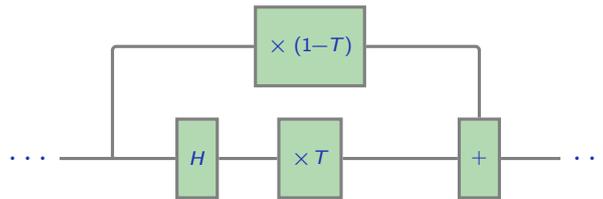
The “Highway networks” by Srivastava et al. (2015) use the idea of gating developed for recurrent units. It replaces a standard non-linear layer

$$y = H(x; W_H)$$

with a layer that includes a “gated” pass-through

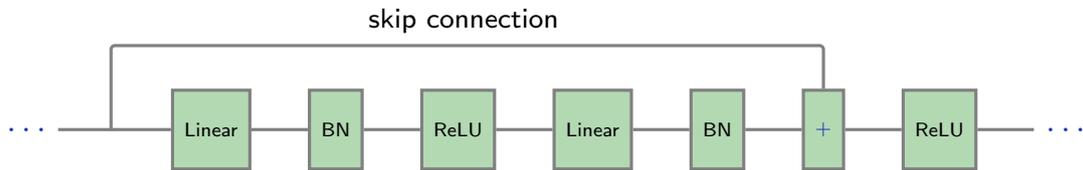
$$y = T(x; W_T)H(x; W_H) + (1 - T(x; W_T))x$$

where  $T(x; W_T) \in [0, 1]$  modulates how much the signal should be transformed.



Initializing  $T$ 's parameters so that  $T \simeq 0$  at first, assures that gradients will pass through, and allows to train networks with up to 100 layers.

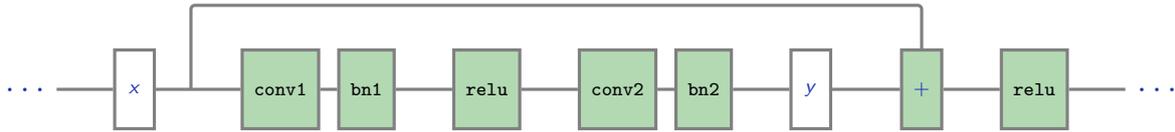
The residual networks proposed by He et al. (2015) simplify the idea and use a building block with a **skip connection**.



Thanks to this structure, the parameters are optimized to learn a **residual**, that is the difference between the value before the block and the one needed after.

We can implement such a network for MNIST, composed of:

- A first convolution layer `conv0` with kernels  $1 \times 1$  to convert the tensor from  $1 \times 28 \times 28$  to `nb_channels`  $\times 28 \times 28$ ,
- a series of `ResBlocks`, each composed of two convolution layers and two batch normalization layers, that maintains the tensor size unchanged,
- an average pooling layer `avg` that produces an output of size `nb_channels`  $\times 1 \times 1$ ,
- a fully connected layer `fc` to make the final prediction.



```
class ResBlock(nn.Module):
    def __init__(self, nb_channels, kernel_size):
        super().__init__()

        self.conv1 = nn.Conv2d(nb_channels, nb_channels, kernel_size,
                                padding = (kernel_size-1)//2)
        self.bn1 = nn.BatchNorm2d(nb_channels)

        self.conv2 = nn.Conv2d(nb_channels, nb_channels, kernel_size,
                                padding = (kernel_size-1)//2)
        self.bn2 = nn.BatchNorm2d(nb_channels)

    def forward(self, x):
        y = self.bn1(self.conv1(x))
        y = F.relu(y)
        y = self.bn2(self.conv2(y))
        y += x
        y = F.relu(y)
        return y
```

### Notes

We expect the kernel size  $k$  to be odd. To obtain activation maps of the same size as the input, we must add a padding of  $\frac{k-1}{2}$ .  
 More generally, we solve

$$n = \frac{n + 2p - k + 1}{s} \Leftrightarrow p = \frac{k - 1}{2}$$

where  $n$  is the input dimension,  $s$  the stride (equal to 1 here),  $p$  the padding, and  $k$  the kernel size (see lecture 4.4. "Convolutions").

```

class ResNet(nn.Module):
    def __init__(self, nb_channels, kernel_size, nb_blocks):
        super().__init__()

        self.conv0 = nn.Conv2d(1, nb_channels, kernel_size = 1)

        self.resblocks = nn.Sequential(
            # A bit of fancy Python
            *(ResBlock(nb_channels, kernel_size) for _ in range(nb_blocks))
        )

        self.avg = nn.AvgPool2d(kernel_size = 28)
        self.fc = nn.Linear(nb_channels, 10)

    def forward(self, x):
        x = F.relu(self.conv0(x))
        x = self.resblocks(x)
        x = F.relu(self.avg(x))
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x

```

---

## Notes

Note that the first convolution layer `conv0` is there so that the residual blocks get as input the proper number of channels.

This network has therefore:

- one convolution layer,
- `nb_blocks` residual blocks,
- one average pooling,
- one linear layer for the final classification.

Before the average pooling, the size of the signal is  $\text{nb\_channels} \times 28 \times 28$ , since MNIST images are of size  $28 \times 28$  and that the residual blocks preserve the size.

With 25 residual blocks, 16 channels, and convolution kernels of size  $3 \times 3$ , we get the following structure, with 117,802 parameters.

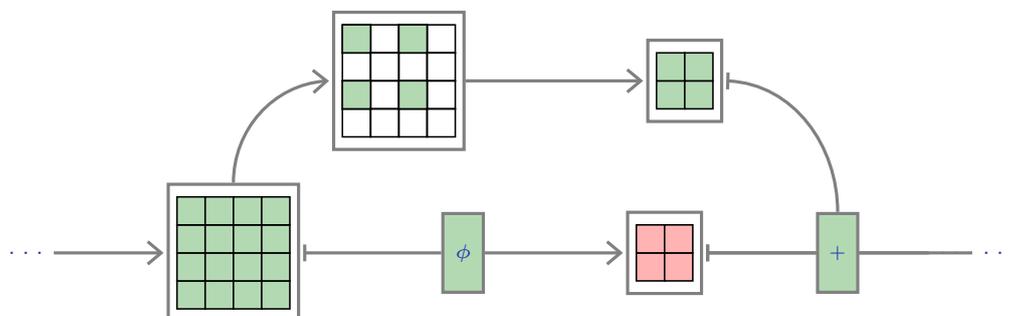
```
ResNet(  
  (conv0): Conv2d(1, 16, kernel_size=(1, 1), stride=(1, 1))  
  (resblocks): Sequential(  
    (0): ResBlock(  
      (conv1): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
      (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      (conv2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
      (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    )  
    /.../  
    (24): ResBlock(  
      (conv1): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
      (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
      (conv2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
      (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    )  
  )  
  (avg): AvgPool2d(kernel_size=28, stride=28, padding=0)  
  (fc): Linear(in_features=16, out_features=10, bias=True)  
)
```

A technical point for a more general use of a residual architecture is to deal with convolution layers that change the activation map sizes or numbers of channels.

He et al. (2015) only consider:

- reducing the activation map size by a factor 2,
- increasing the number of channels.

To reduce the activation map size by a factor 2, the identity pass-through extracts 1/4 of the activations over a regular grid (i.e. with a stride of 2),

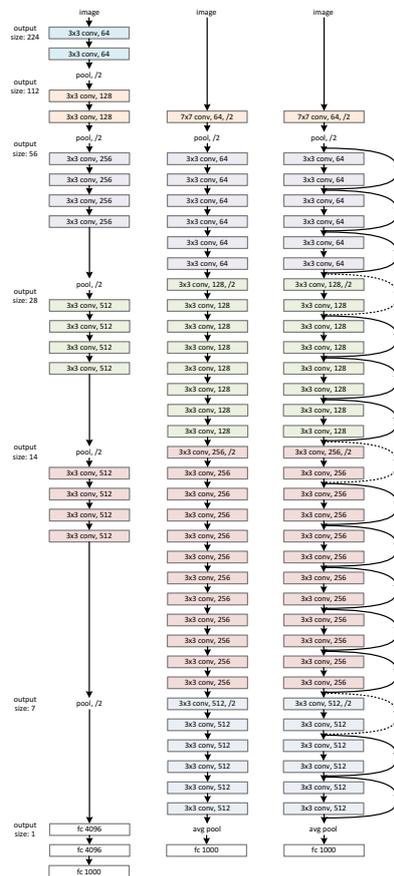


To increase the number of channels from  $C$  to  $C'$ , they propose to either:

- pad the original value with  $C' - C$  zeros, which amounts to adding as many zeroed channels, or
- use  $C'$  convolutions with a  $1 \times 1 \times C$  filter, which corresponds to applying the same fully-connected linear model  $\mathbb{R}^C \rightarrow \mathbb{R}^{C'}$  at every location.

Finally, He et al.'s residual networks are fully convolutional, which means they have no fully connected layers. We will come back to this.

Their one-before last layer is a per-channel global average pooling that outputs a  $1d$  tensor, fed into a single fully-connected layer.



(He et al., 2015)

## Notes

- The network on the left is a VGG19,
- The network in the middle is a 34-layer network without skip connections,
- The network on the right is a 34-layer residual network.

## Performance on ImageNet.

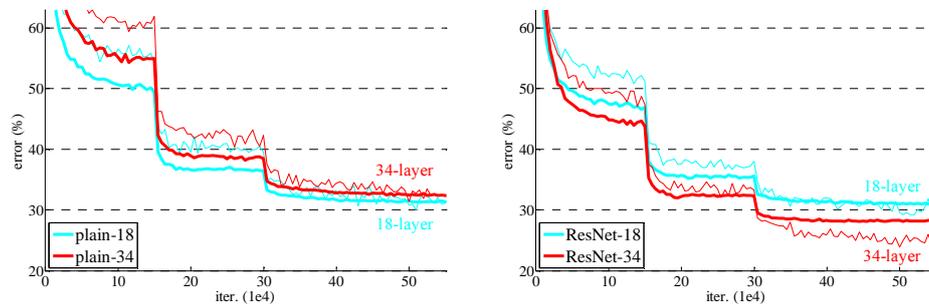


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

(He et al., 2015)

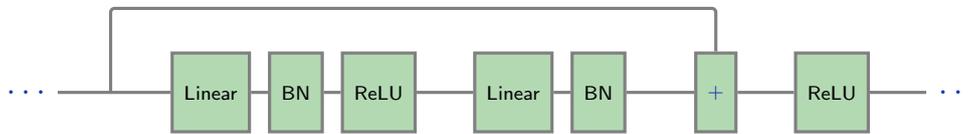
---

### Notes

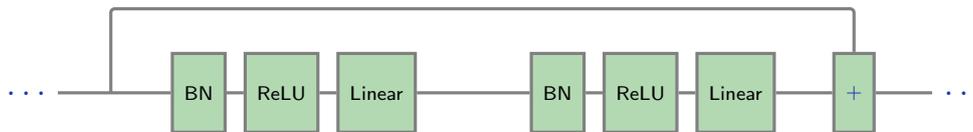
Residual networks were a breakthrough that reduced the (top-5) error rate on ImageNet down to 3.57 at the time when state-of-the-art was at 4.82.

The plots show the training and test errors on ImageNet. The breaks in slope correspond to moments where the learning is reduced.

He et al. (2016) proposed to sequence operations in a residual block so that the main “pathway” has no non-linearity. This results in substantial improvements.



Original (He et al., 2015)



Identity residual (He et al., 2016)

Veit et al. (2016) interpret a residual network as an ensemble, which explains in part its stability.

E.g., with three blocks we have

$$x_1 = x_0 + f_1(x_0)$$

$$x_2 = x_1 + f_2(x_1)$$

$$x_3 = x_2 + f_3(x_2)$$

hence there are four “paths”:

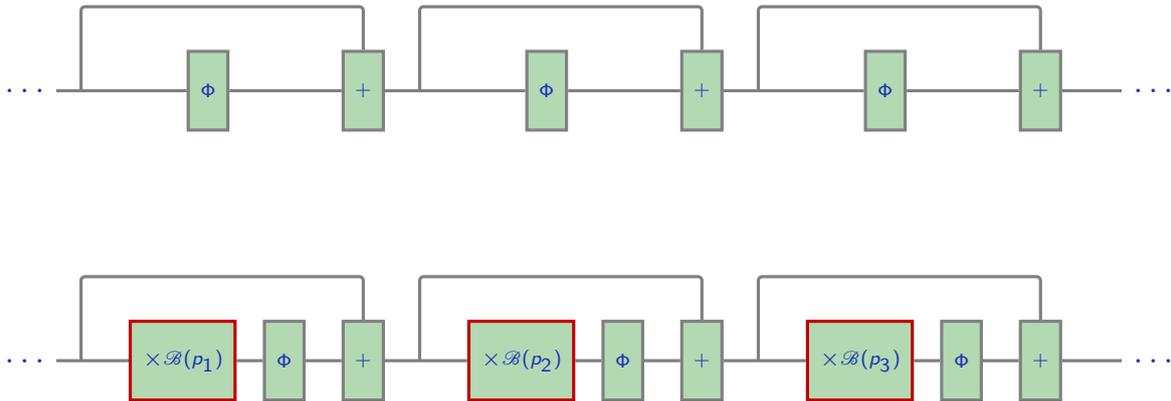
$$\begin{aligned} x_3 &= x_2 + f_3(x_2) \\ &= x_1 + f_2(x_1) + f_3(x_1 + f_2(x_1)) \\ &= \underbrace{x_0}_{\text{path 1}} + \underbrace{f_1(x_0)}_{\text{path 2}} + \underbrace{f_2(x_0 + f_1(x_0))}_{\text{path 3}} + \underbrace{f_3(x_0 + f_1(x_0) + f_2(x_0 + f_1(x_0)))}_{\text{path 4}}. \end{aligned}$$

Veit et al. show that (1) performance reduction correlates with the number of paths removed from the ensemble, not with the number of blocks removed, (2) only gradients through shallow paths matter during train.

An extension of the residual network, is the **stochastic depth** network.

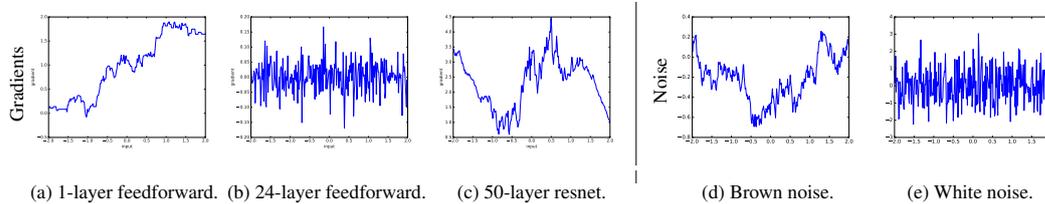
“Stochastic depth aims to shrink the depth of a network during training, while keeping it unchanged during testing. We can achieve this goal by randomly dropping entire ResBlocks during training and bypassing their transformations through skip connections.”

(Huang et al., 2016)



## Shattered Gradient

Balduzzi et al. (2017) points out that depth “shatters” the relation between the input and the gradient w.r.t. the input, and that Resnets mitigate this effect.



(Balduzzi et al., 2017)

Since linear networks avoid this problem, they suggest to combine CReLU (see lecture 6.2. “Rectifiers”) with a **Looks Linear initialization** that makes the network linear initially.

---

## Notes

The three left graphs show the gradient of the output with respect to the input, as a function of the input, on a 1d synthetic example.

- with a shallow network, the relation between the gradient and the input is quite structured;
- with a deep network, the relation is shattered. There is no dependency between the change in the input and the change in the gradient;
- with a resnet, there is actually a large scale structure between gradient and input.

The two right graphs show the structure of a brown noise and a white noise that exhibit similar differences of structure.

Let  $\sigma(x) = \max(0, x)$ , and

$$\Phi : \mathbb{R}^D \rightarrow \mathbb{R}^{2D}$$

the CReLU non-linearity, i.e.

$$\forall x \in \mathbb{R}^D, q = 1, \dots, D, \begin{cases} \Phi(x)_{2q-1} & = \sigma(x_q), \\ \Phi(x)_{2q} & = \sigma(-x_q) \end{cases}$$

and a weight matrix  $\tilde{W} \in \mathbb{R}^{D' \times 2D}$  such that

$$\forall j = 1, \dots, D', q = 1, \dots, D, \tilde{W}_{j,2q-1} = -\tilde{W}_{j,2q} = W_{j,q}.$$

So two neighboring columns of  $\Phi(x)$  are the  $\sigma(\cdot)$  and  $\sigma(-\cdot)$  of a column of  $x$ , and two neighboring columns of  $\tilde{W}$  are a column of  $W$  and its opposite.

---

## Notes

$$W = \begin{bmatrix} W_{1,1} & \dots & W_{1,D} \\ \vdots & & \vdots \\ W_{D',1} & \dots & W_{D',D} \end{bmatrix} \in \mathbb{R}^{D' \times D}$$

$$\tilde{W} = \begin{bmatrix} W_{1,1} & -W_{1,1} & \dots & W_{1,D} & -W_{1,D} \\ \vdots & & & & \vdots \\ W_{D',1} & W_{D',1} & \dots & W_{D',D} & -W_{D',D} \end{bmatrix} \in \mathbb{R}^{D' \times 2D}$$

From this we get,  $\forall i = 1, \dots, B, j = 1, \dots, D'$ :

$$\begin{aligned} (\tilde{W}\Phi(x))_j &= \sum_{k=1}^{2D} \tilde{W}_{j,k} \Phi(x)_k \\ &= \sum_{q=1}^D \tilde{W}_{j,2q-1} \Phi(x)_{2q-1} + \tilde{W}_{j,2q} \Phi(x)_{2q} \\ &= \sum_{q=1}^D W_{j,q} \sigma(x_q) - W_{j,q} \sigma(-x_q) \\ &= \sum_{q=1}^D W_{j,q} x_q \\ &= (Wx)_j. \end{aligned}$$

Hence

$$\forall x, \tilde{W}\Phi(x) = Wx$$

and doing this in every layer results in a linear network.

---

## Notes

When the network is initialized in such a way (“looks linear” initialization), at start, right before training, it behaves as a linear network.

This induces a strong relation between gradient and the input, as it happens thanks to the pass-through in residual networks.

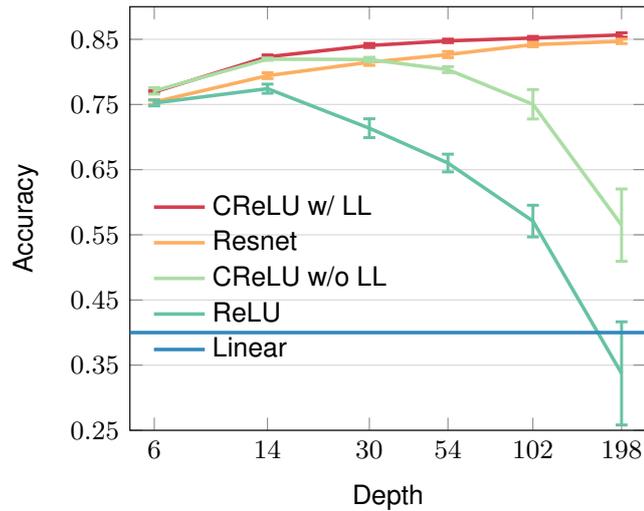


Figure 6: **CIFAR-10 test accuracy**. Comparison of test accuracy between networks of different depths with and without LL initialization.

(Balduzzi et al., 2017)

---

## Notes

- “Linear” stands for a simple linear network;
- “ReLU” stands for a Resnet with ReLU activations and no skip-connection;
- “CReLU w/o LL” stands for a deep network (no residual connection) with CReLU activations, initialized with the standard Resnet procedure;
- “Resnet” stands for the same architecture as “ReLU” but with skip connections;
- “CReLU w/ LL” stands for a deep network (no residual connections) with CReLU activations and “look linear” initialization.

The standard network (with CReLU and no residual connection) initialized with the “look linear” procedure is competitive with the resnet architecture.

We can summarize the techniques which have enabled the training of very deep architectures:

- rectifiers to prevent the gradient from vanishing during the backward pass,
- dropout to force a distributed representation,
- batch normalization to dynamically maintain the statistics of activations,
- identity pass-through to keep a structured gradient and distribute representation,
- smart initialization to put the gradient in a good regime.

---

## Notes

The structures were carefully engineered to make them trainable: The architectures may result in a “less optimal” mapping space, but the ease of optimization allows to eventually get a better trained model.

## References

- D. Balduzzi, M. Frean, L. Leary, J. Lewis, K. Wan-Duo Ma, and B. McWilliams. **The shattered gradients problem: If resnets are the answer, then what is the question?** CoRR, abs/1702.08591, 2017.
- K. He, X. Zhang, S. Ren, and J. Sun. **Deep residual learning for image recognition.** CoRR, abs/1512.03385, 2015.
- K. He, X. Zhang, S. Ren, and J. Sun. **Identity mappings in deep residual networks.** CoRR, abs/1603.05027, 2016.
- G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. **Deep networks with stochastic depth.** CoRR, abs/1603.09382, 2016.
- R. Srivastava, K. Greff, and J. Schmidhuber. **Highway networks.** CoRR, abs/1505.00387, 2015.
- A. Veit, M. Wilber, and S. Belongie. **Residual networks behave like ensembles of relatively shallow networks.** CoRR, abs/1605.06431, 2016.