周学习总结

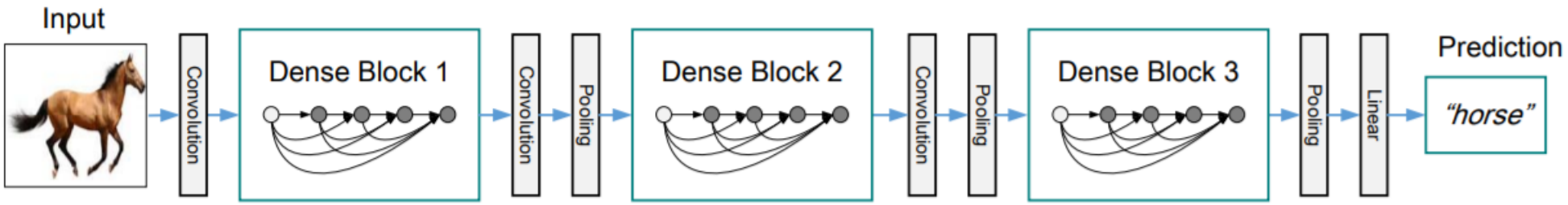
许典

视频学习

- 完成Life Long Learning 和 Deep Reinforcement Learning

论文

- Densely Connected Convolutional Networks (CVPR 2017) DenseNet



Dense Block 和 Dense Layer

```
(conv0): Conv2d(3, 24, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
(denseblock1): _DenseBlock(
  (denselayer1): _DenseLayer(
    (norm1): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu1): ReLU(inplace=True)
    (conv1): Conv2d(24, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (norm2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu2): ReLU(inplace=True)
    (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  )
  (denselayer2): _DenseLayer(
    (norm1): BatchNorm2d(36, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu1): ReLU(inplace=True)
    (conv1): Conv2d(36, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (norm2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu2): ReLU(inplace=True)
    (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  )
  (denselayer3): _DenseLayer(
    (norm1): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu1): ReLU(inplace=True)
    (conv1): Conv2d(48, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (norm2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu2): ReLU(inplace=True)
    (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  )
  (denselayer4): _DenseLayer(
    (norm1): BatchNorm2d(60, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu1): ReLU(inplace=True)
    (conv1): Conv2d(60, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (norm2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu2): ReLU(inplace=True)
    (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  )
  (denselayer5): _DenseLayer(
    (norm1): BatchNorm2d(72, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu1): ReLU(inplace=True)
    (conv1): Conv2d(72, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (norm2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu2): ReLU(inplace=True)
    (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  )
```

```python
class _DenseBlock(nn.Module):
    def __init__(self, num_layers, num_input_features, bn_size, growth_rate, drop_rate, efficient=False):
        super(_DenseBlock, self).__init__()
        for i in range(num_layers):
            layer = _DenseLayer(
                num_input_features + i * growth_rate,
                growth_rate=growth_rate,
                bn_size=bn_size,
                drop_rate=drop_rate,
                efficient=efficient,
            )
            self.add_module('denselayer%d' % (i + 1), layer)

    def forward(self, init_features):
        features = [init_features]
        for name, layer in self.named_children():
            new_features = layer(*features)
            features.append(new_features)
        return torch.cat(features, 1)
```

Method	Depth	Params	C10	C10+	C100	C100+	SVHN
Network in Network [22]	-	-	10.41	8.81	35.68	-	2.35
All-CNN [32]	-	-	9.08	7.25	-	33.71	-
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57	1.92
Highway Network [34]	-	-	-	7.72	-	32.39	-
FractalNet [17]	21	38.6M	10.18	5.22	35.34	23.30	2.01
with Dropout/Drop-path	21	38.6M	7.33	4.60	28.20	23.73	1.87
ResNet [11]	110	1.7M	-	6.61	-	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22	2.01
ResNet with Stochastic Depth [13]	110	1.7M	11.66	5.23	37.80	24.58	1.75
	1202	10.2M	-	4.91	-	-	-
Wide ResNet [42]	16	11.0M	-	4.81	-	22.07	-
	28	36.5M	-	4.17	-	20.50	-
with Dropout	16	2.7M	-	-	-	-	1.64
ResNet (pre-activation) [12]	164	1.7M	11.26*	5.46	35.58*	24.33	-
	1001	10.2M	10.56*	4.62	33.47*	22.71	-
DenseNet ($k = 12$)	40	1.0M	**7.00**	5.24	**27.55**	24.42	1.79
DenseNet ($k = 12$)	100	7.0M	**5.77**	**4.10**	**23.79**	**20.20**	1.67
DenseNet ($k = 24$)	100	27.2M	**5.83**	**3.74**	**23.42**	**19.25**	**1.59**
DenseNet-BC ($k = 12$)	100	0.8M	**5.92**	4.51	**24.15**	22.27	1.76
DenseNet-BC ($k = 24$)	250	15.3M	**5.19**	**3.62**	**19.64**	**17.60**	1.74
DenseNet-BC ($k = 40$)	190	25.6M	-	**3.46**	-	**17.18**	-

Table 2: Error rates (%) on CIFAR and SVHN datasets. k denotes network's growth rate. Results that surpass all competing methods are **bold** and the overall best results are **blue**. "+" indicates standard data augmentation (translation and/or mirroring). $*$ indicates results run by ourselves. All the results of DenseNets without data augmentation (C10, C100, SVHN) are obtained using Dropout. DenseNets achieve lower error rates while using fewer parameters than ResNet. Without data augmentation, DenseNet performs better by a large margin.

```
Test    Iter: [149/157] Time 0.024 (0.019)      Loss 0.1856 (0.2925)    Error 0.0625 (0.0848)
Test    Iter: [150/157] Time 0.021 (0.019)      Loss 0.2729 (0.2923)    Error 0.0469 (0.0846)
Test    Iter: [151/157] Time 0.018 (0.019)      Loss 0.3795 (0.2929)    Error 0.0781 (0.0845)
Test    Iter: [152/157] Time 0.018 (0.019)      Loss 0.1332 (0.2919)    Error 0.0312 (0.0842)
Test    Iter: [153/157] Time 0.019 (0.019)      Loss 0.3244 (0.2921)    Error 0.1250 (0.0845)
Test    Iter: [154/157] Time 0.018 (0.019)      Loss 0.6795 (0.2946)    Error 0.1719 (0.0850)
Test    Iter: [155/157] Time 0.018 (0.019)      Loss 0.0713 (0.2932)    Error 0.0312 (0.0847)
Test    Iter: [156/157] Time 0.018 (0.019)      Loss 0.3249 (0.2934)    Error 0.0938 (0.0847)
Test    Iter: [157/157] Time 0.008 (0.019)      Loss 0.3348 (0.2934)    Error 0.1250 (0.0848)
Final test error: 0.0848
Done!
```