UNIVERSITAT POLITÈCNICA DE VALÈNCIA

MIARFID Official Master's Degree in Artificial Intelligence, Pattern Recognition and Digital Imaging

Chapter 1. Convolutional Layers - Part B

Neural Networks

2022/2023

Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

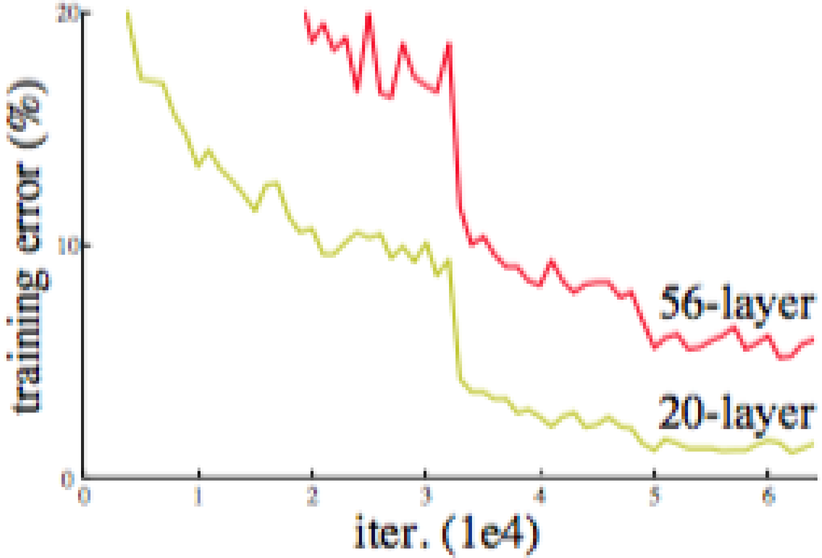
Departamento de Sistemas Informáticos y Computación

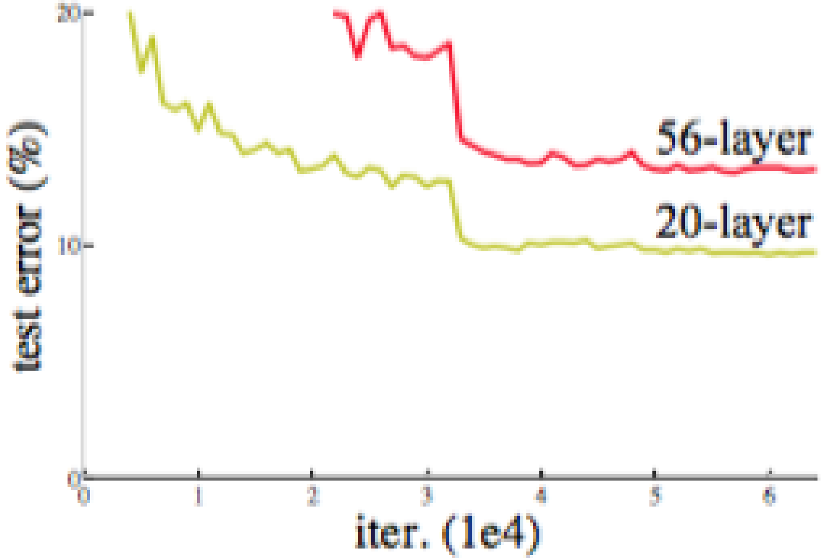
Index

Index

Introduction

- Train Deep Networks is hard:





Introduction

- Increasing network depth does not work by simply stacking layers together
- Still vanishing gradient problem
- Potential solutions: Optimizers, Initializers (Layer-sequential unit-variance, LSUV), Activation Functions (PReLU, ELU, SELU)
- Potential solutions: GoogleNet with auxiliary loss in a middle layer as extra supervision
- Better Solutions: identity shortcut connections, Residual Nets

Index

Highway Nets

- Inspired by the LSTM recurrent neural networks
- Special gate that can allow computation paths without attenuation
- This paths are known as *information highways*
- In general an output of a layer can be expressed as:

$$\mathbf{y} = H(\mathbf{x}, W_H)$$

- Additionally two non-linear transforms are define, $T(\mathbf{x}, W_T)$ and $C(\mathbf{x}, W_C)$

$$\mathbf{y} = T(\mathbf{x}, W_T) \cdot H(\mathbf{x}, W_H) + \mathbf{x} \cdot C(\mathbf{x}, W_C)$$

Highway Nets

- In this expression:

$$\mathbf{y} = T(\mathbf{x}, W_T) \cdot H(\mathbf{x}, W_H) + \mathbf{x} \cdot C(\mathbf{x}, W_C)$$

- T is the *transform* gate and C the *carry* gate

- For simplicity $C = 1 - T$

$$\mathbf{y} = T(\mathbf{x}, W_T) \cdot H(\mathbf{x}, W_H) + \mathbf{x} \cdot (1 - T(\mathbf{x}, W_T))$$

- we could define T as:

$$T(\mathbf{x}) = \sigma(W_T^t \mathbf{x} + \mathbf{b}_t)$$

Highway Nets

Network	No. of Layers	No. of Parameters	Accuracy (in %)
Fitnet Results (reported by Romero et. al.[25])			
Teacher	5	∼9M	90.18
Fitnet A	11	∼250K	89.01
Fitnet B	19	∼2.5M	91.61
Highway networks			
Highway A (Fitnet A)	11	∼236K	89.18
Highway B (Fitnet B)	19	∼2.3M	**92.46 (92.28±0.16)**
Highway C	32	∼1.25M	91.20

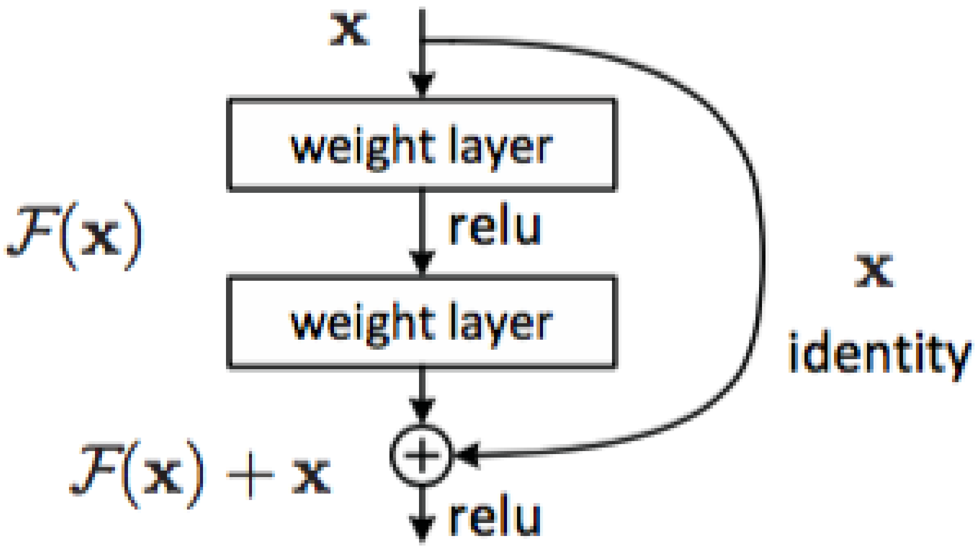
Highway Nets

Network	CIFAR-10 Accuracy (in %)	CIFAR-100 Accuracy (in %)
Maxout [20]	90.62	61.42
dasNet [36]	90.78	66.22
NiN [35]	91.19	64.32
DSN [24]	92.03	65.43
All-CNN [37]	**92.75**	66.29
Highway Network	92.40 (92.31±0.12)	**67.76 (67.61±0.15)**

Index

Residual Nets

- Key idea: fit residual mappings instead of mappings



like a highway but parameter-free, no gate.

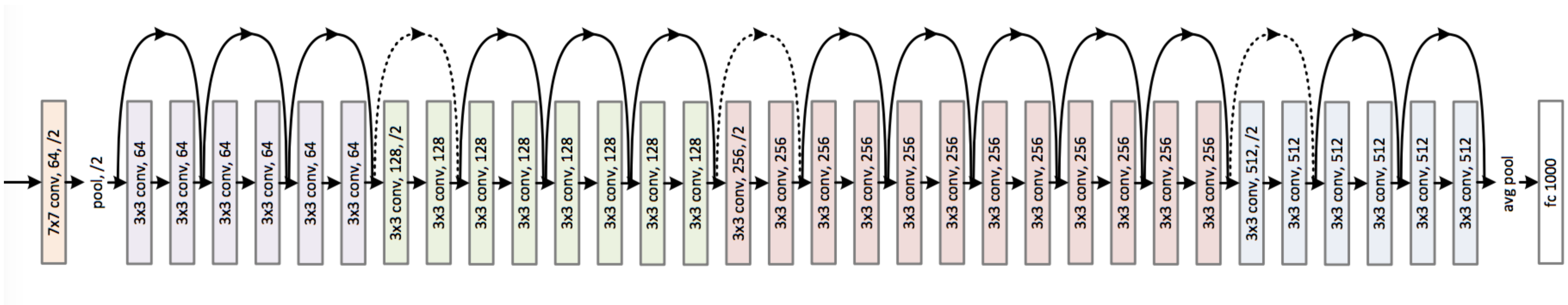
Residual Nets

- ResNet with different depth

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Residual Nets

- ResNet 34-layers



- Dotted lines usually: 1x1 convolution, stride 2

- Convolution + BN + Activation

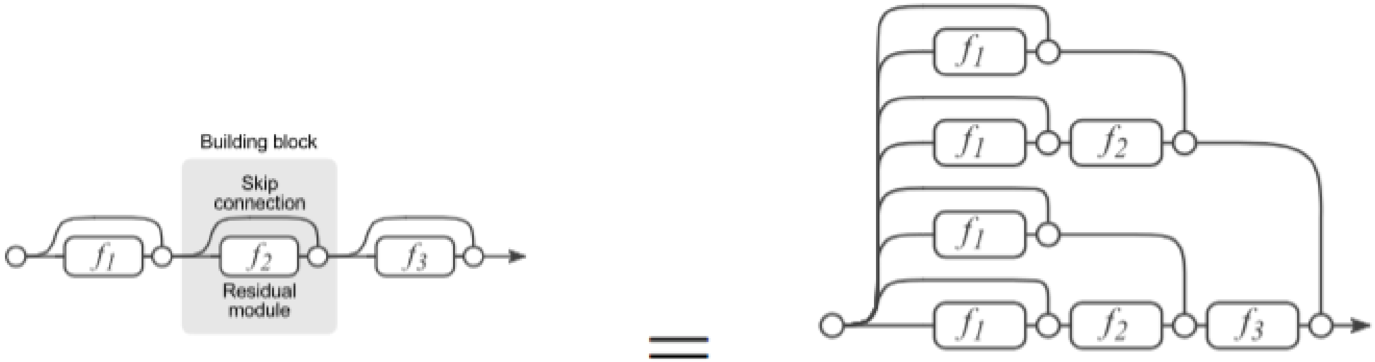
Residual Nets

- Results on ImageNet

model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
GoogLeNet [44]	-	9.15
PReLU-net [13]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	**21.43**	**5.71**

Residual Nets - Ensemble of shallow nets

- ResNets as an esemble:

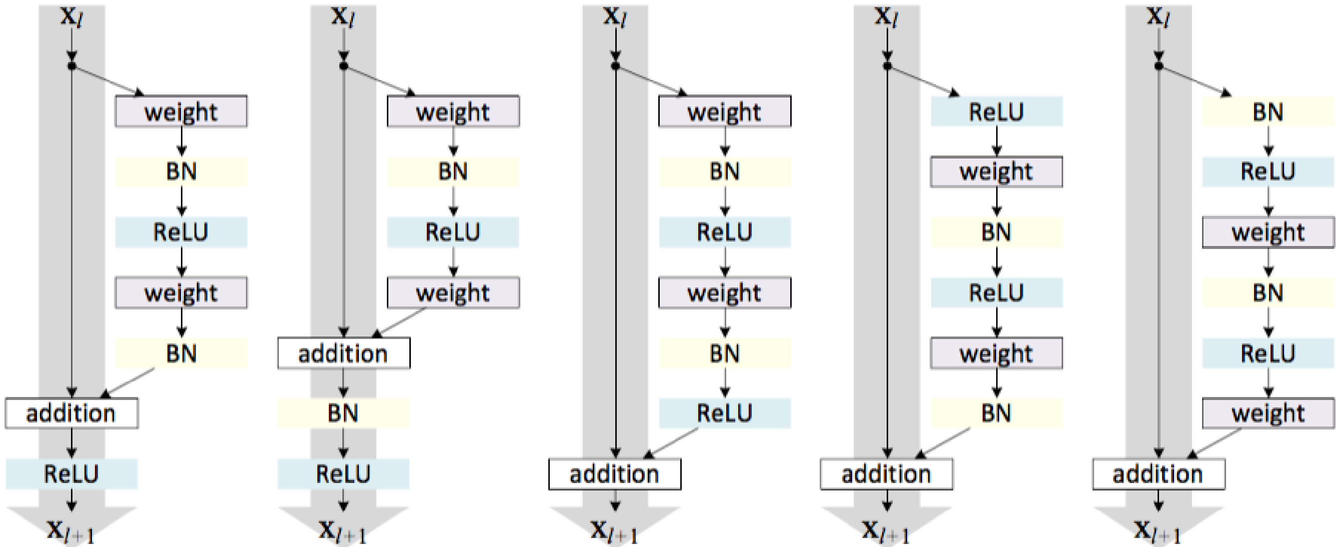


- All paths with one block, two blocks and three blocks are considered

Residual Nets - Pre-activation

- Identity Mapping, Pre-activation resnets:

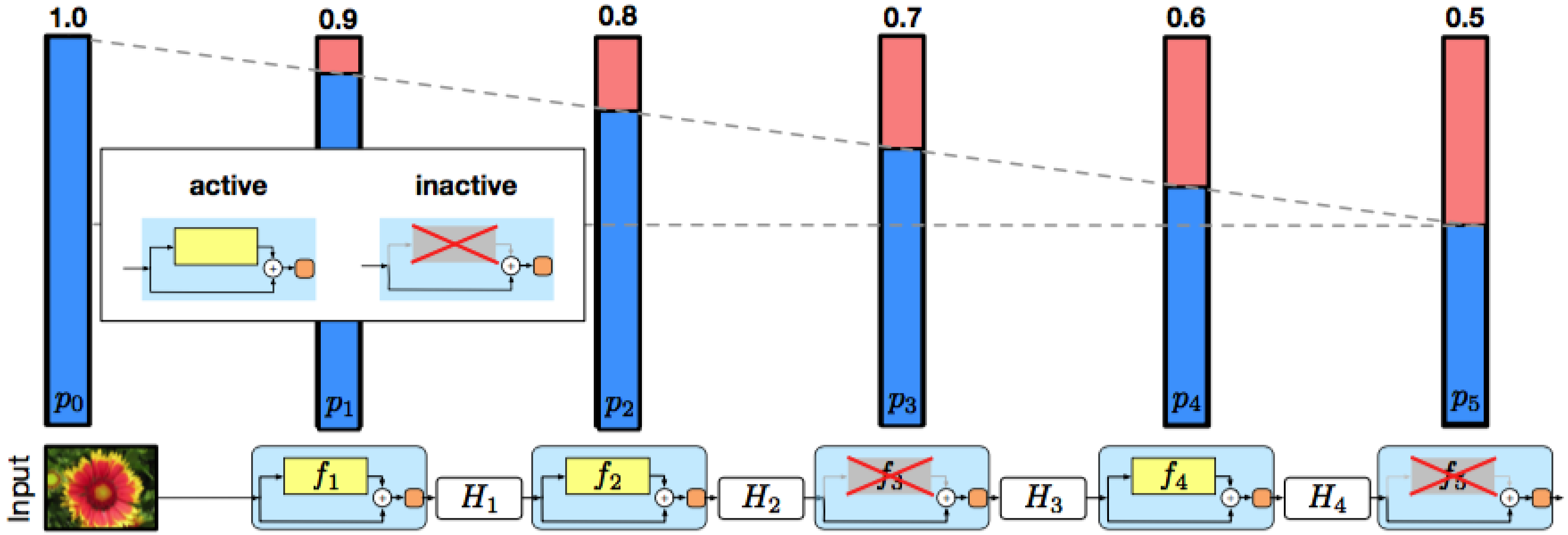
case	Fig.	ResNet-110	ResNet-164
original Residual Unit [1]	Fig. 4(a)	6.61	5.93
BN after addition	Fig. 4(b)	8.17	6.50
ReLU before addition	Fig. 4(c)	7.84	6.14
ReLU-only pre-activation	Fig. 4(d)	6.71	5.91
full pre-activation	Fig. 4(e)	**6.37**	**5.46**



(a) original (b) BN after addition (c) ReLU before addition (d) ReLU-only pre-activation (e) **full pre-activation**

Residual Nets - Stochastic Depth

- A drop-out over the resnet blocks:



Residual Nets - Stochastic Depth

- Results:

	CIFAR10+	CIFAR100+	SVHN	ImageNet
Maxout [21]	9.38	-	2.47	-
DropConnect [20]	9.32	-	1.94	-
Net in Net [24]	8.81	-	2.35	-
Deeply Supervised [13]	7.97	-	1.92	33.70
Frac. Pool [25]	-	27.62	-	-
All-CNN [6]	7.25	-	-	41.20
Learning Activation [26]	7.51	30.83	-	-
R-CNN [27]	7.09	-	1.77	-
Scalable BO [28]	6.37	27.40	1.77	-
Highway Network [29]	7.60	32.24	-	-
Gen. Pool [30]	6.05	-	1.69	28.02
ResNet with constant depth	6.41	27.76	1.80	21.78
ResNet with stochastic depth	5.25	24.98	1.75	21.98

Residual Nets - Stochastic Depth

- Computing cost improvement:

	CIFAR10+	CIFAR100+	SVHN
Constant Depth	20h 42m	20h 51m	33h 43m
Stochastic Depth	15h 7m	15h 20m	25h 33m

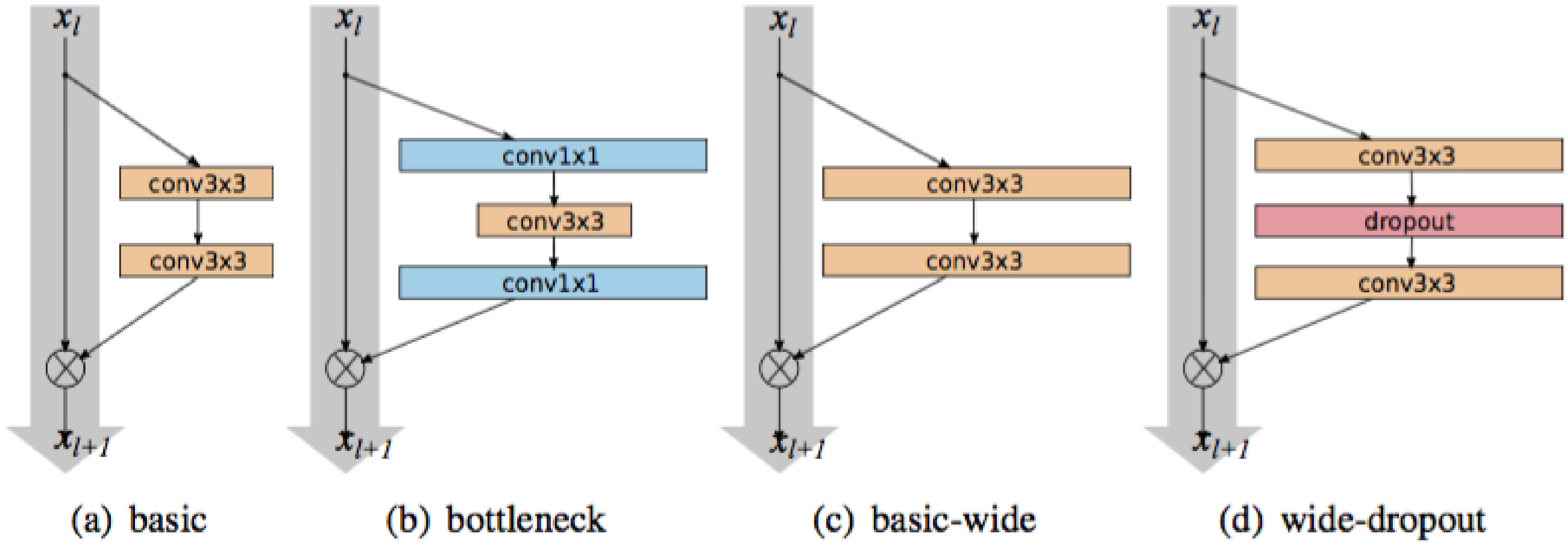
Index

Wide ResNets

- Some ResNet layers does not provide any beneficial representation learning, they are bypassed
- Provide more expresivity to ResNets and computation efficiency
- Wide the convolutions: more filters, more channels (depth)
- Use a scaling factor k to control the wide of the convolutions, where $k = 1$ is the original ResNet params
- Also introduce a dropout layer

Wide ResNets

- Different configurations:



Wide ResNets

```
def conv_block_wideresnet(input, filters=16, k=1, dropout=0.0):

    x = Conv2D(filters * k, (3, 3), padding='same')(input)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    if dropout > 0.0:
        x = Dropout(dropout)(x)

    x = Conv2D(filters * k, (3, 3), padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    m = add([input, x])

    return m
```

Wide ResNets

```
N = (depth - 4) // 6
k=10
dropout=0.0

x =conv_block(img_input); //initial conv+BN+ReLu

for i in range(N):
    x =conv_block_wideresnet(x, 16 , k , dropout)
x = MaxPooling2D((2, 2))(x)

for i in range(N):
    x =conv_block_wideresnet(x, 32 , k , dropout)
x = MaxPooling2D((2, 2))(x)

for i in range(N):
    x =conv_block_wideresnet(x, 64 , k , dropout)
x = MaxPooling2D((2, 2))(x)

x = GlobalAveragePooling2D()(x)
x = Dense(classes, activation='softmax')(x)
```

Wide ResNets

- Results, note the depth factor:

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

Wide ResNets

- Results:

	depth-k	# params	CIFAR-10	CIFAR-100
NIN [20]			8.81	35.67
DSN [19]			8.22	34.57
FitNet [24]			8.39	35.04
Highway [28]			7.72	32.39
ELU [5]			6.55	24.28
original-ResNet[11]	110	1.7M	6.43	25.16
	1202	10.2M	7.93	27.82
stoc-depth[14]	110	1.7M	5.23	24.58
	1202	10.2M	4.91	-
pre-act-ResNet[13]	110	1.7M	6.37	-
	164	1.7M	5.46	24.33
	1001	10.2M	4.92(4.64)	22.71
WRN (ours)	40-4	8.9M	4.53	21.18
	16-8	11.0M	4.27	20.43
	28-10	36.5M	**4.00**	**19.25**

Wide ResNets

- Results with dropout:

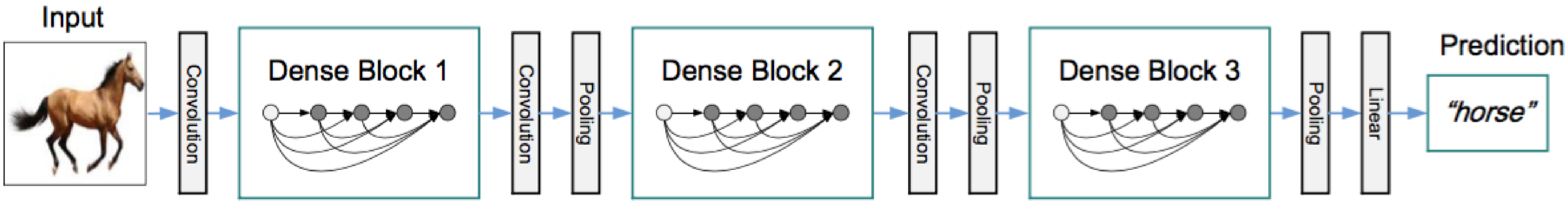
depth	k	dropout	CIFAR-10	CIFAR-100	SVHN
16	4		5.02	24.03	1.85
16	4	✓	5.24	23.91	1.64
28	10		4.00	19.25	-
28	10	✓	**3.89**	**18.85**	-
52	1		6.43	29.89	2.08
52	1	✓	6.28	29.78	1.70

Index

Dense Nets

- Connections from all preceding layers of the same block
- All inputs are **concatenated** instead of added
- The basic operator to each input is: BN-ReLU-Conv3x3
- Transition layers between blocks to fit sizes: Conv1x1-MaxPool2x2
- Bottleneck version: BN-ReLU-Conv1x1 - BN-ReLU-Conv3x3 (DenseNet-B)
- Compression: Transition Layers reduce the number of feature-maps (DenseNet-C)

Dense Nets



- Implementation: Densenet in Keras

Dense Nets

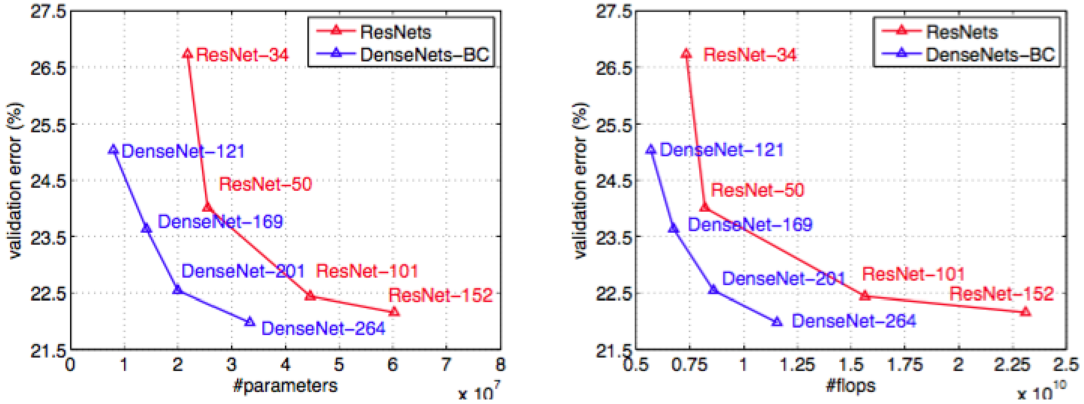
- Results

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**	-

Dense Nets

- Results

Model	top-1	top-5
DenseNet-121	25.02 / 23.61	7.71 / 6.66
DenseNet-169	23.80 / 22.08	6.85 / 5.92
DenseNet-201	22.58 / 21.46	6.34 / 5.54
DenseNet-264	22.15 / 20.80	6.12 / 5.29



Index

Bi-Linear Convolutional Networks

- Convolutional Networks show some problems on fine-grained classification

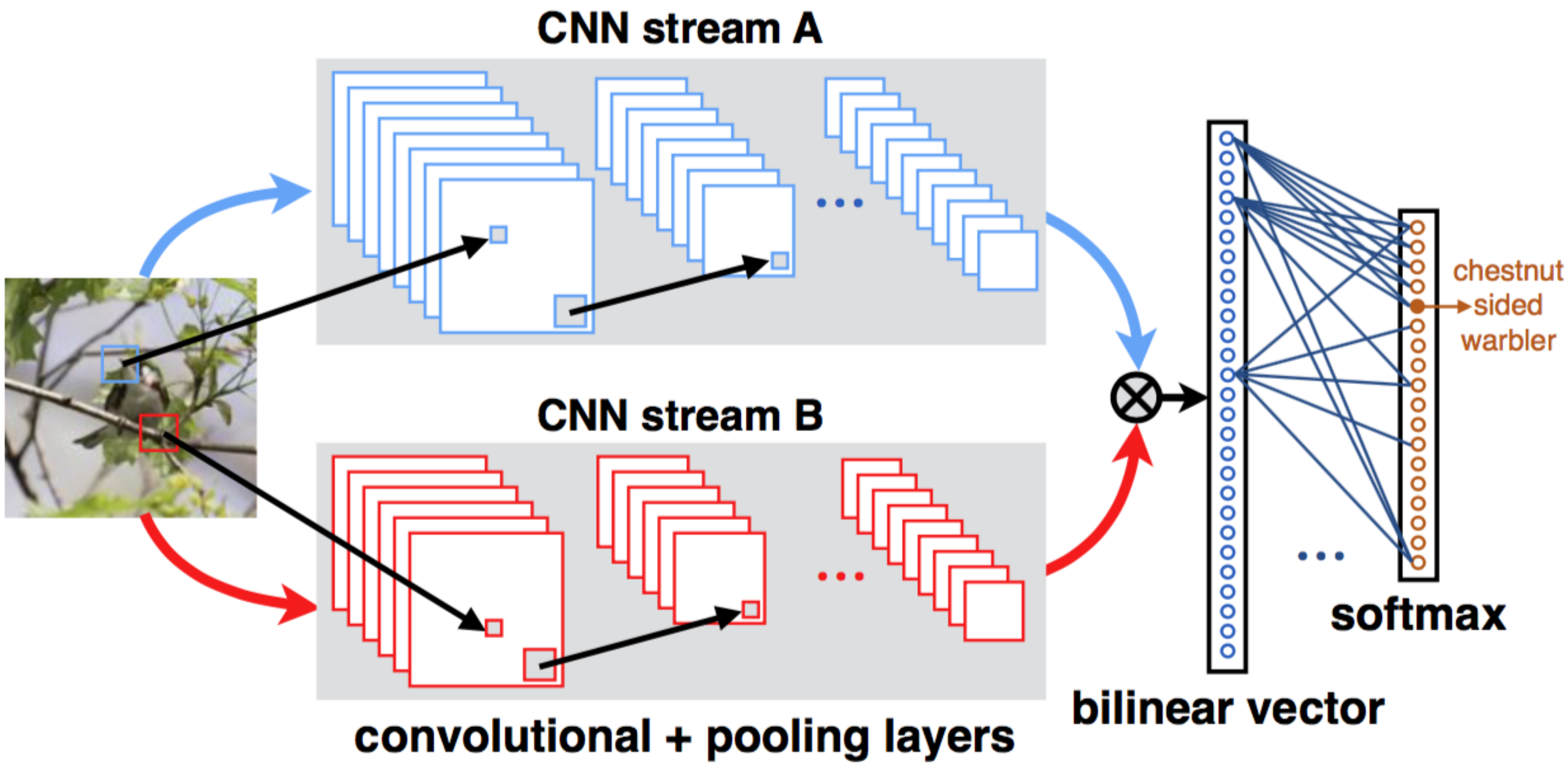


Bi-Linear Convolutional Networks

- Visual differences between the categories are small
- Larger differences caused by factors such as:
  - pose
  - viewpoint
  - location of the object in the image
- Some solutions:
  - Hand-localized parts of images
  - Combinationof CNN features + VLAD/Fisher Vectos
  - Outter products of CNN-features extractor $\rightarrow$ Bi-Linear CNN

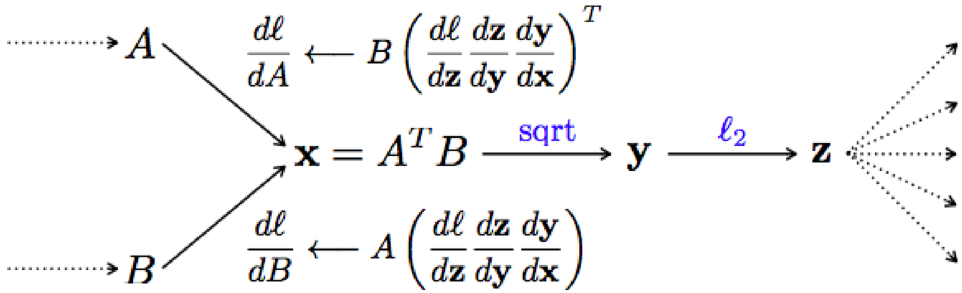
Bi-Linear Convolutional Networks

- Bi-Linear model



Bi-Linear Convolutional Networks

- End-to-end training through backpropagation

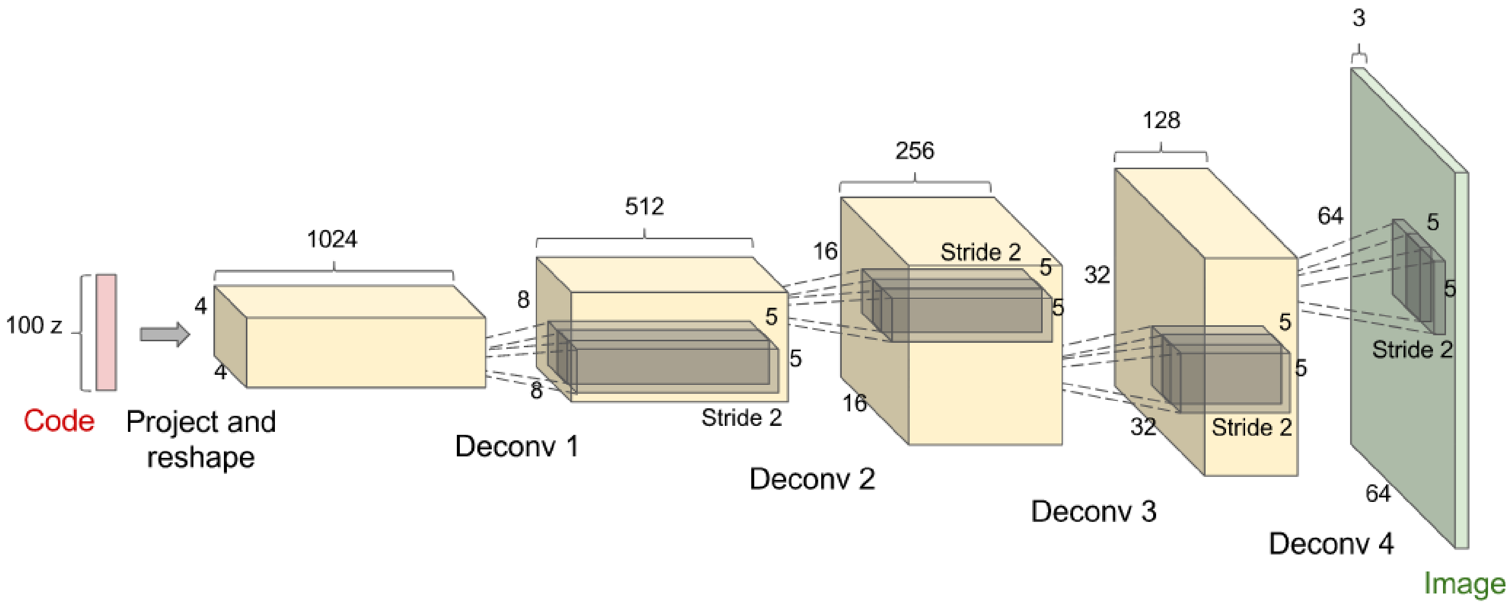



```
def outer_product(x):
  # Einstein Notation  [batch,rows,cols,depth] x [batch,rows,cols,depth] -> [batch,depth,depth]
  phi_I = tf.einsum('ijkm,ijkn->imn',x[0],x[1])

  # Reshape from [batch_size,depth,depth] to [batch_size, depth*depth]
  phi_I = tf.reshape(phi_I,[-1,128*128])

  # Divide by feature map size [sizexsize]
  phi_I = tf.divide(phi_I,31*31)

  # Take signed square root of phi_I
  y_ssqrt = tf.multiply(tf.sign(phi_I),tf.sqrt(tf.abs(phi_I)+1e-12))

  # Apply l2 normalization
  z_l2 = tf.nn.l2_normalize(y_ssqrt, dim=1)
  return z_l2
```

Index

De-Convolution

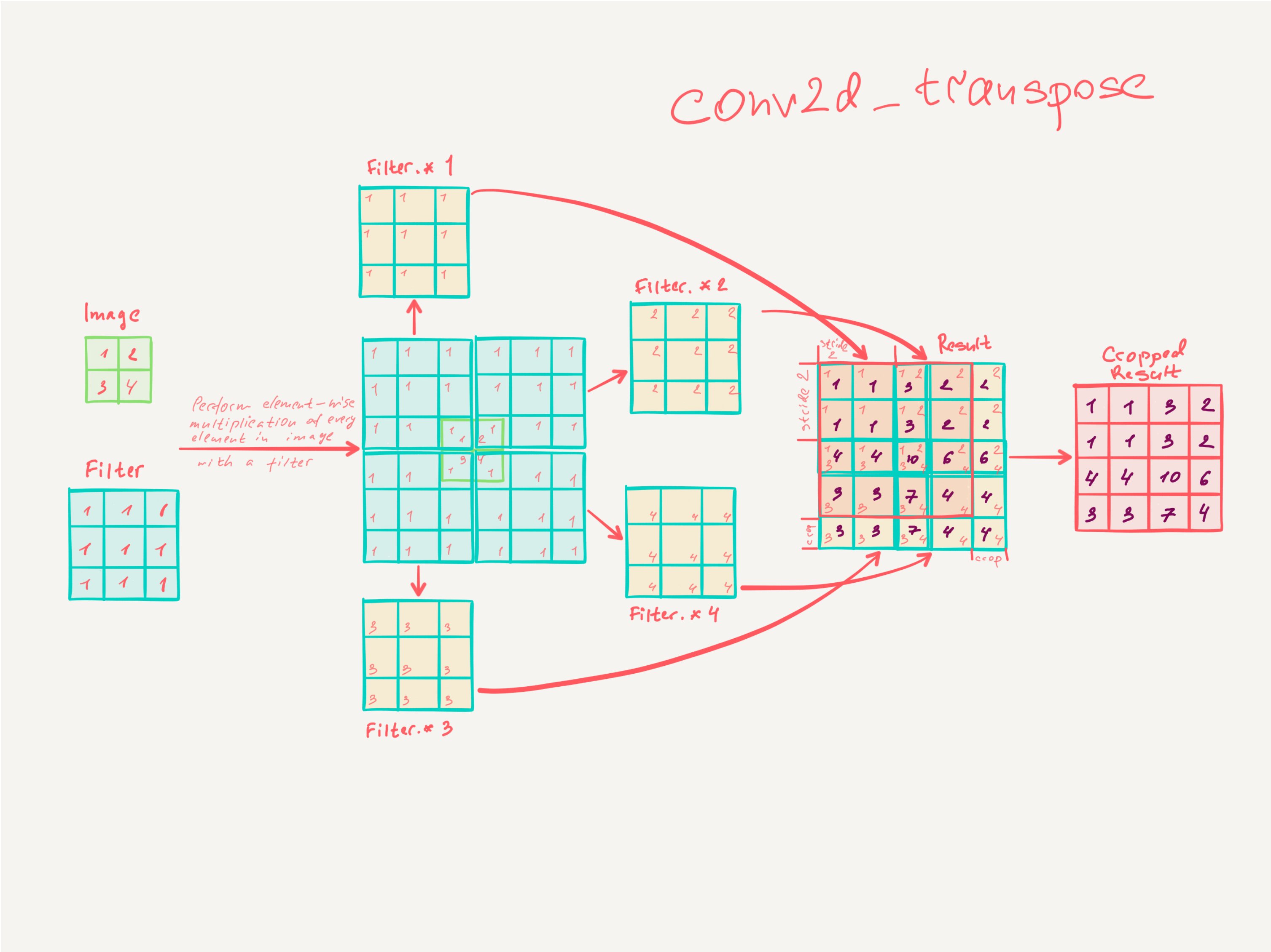
- Goal of a De-Convolution network:



De-Convolution

- Deconvolution layer is a very unfortunate name and should rather be called a transposed convolutional layer
- Deconvolution works as the transpose mechanism of the Convolution
- Convolution Forward is the Deconvolution Backward
- Convolution Backward is the Deconvolution Forward
- Some animations: `https://github.com/vdumoulin/conv_arithmetic`

De-Convolution



De-Convolution

- In Keras:
  - Conv2DTranspose
  - UpSampling2D

- Options for incresing the size of the map:
  - Deconv with fractional stride
  - UpSampling

- Tying options:
  - Conv-DeConv (tying the weights)
  - DePooling (tying the pool indexes)

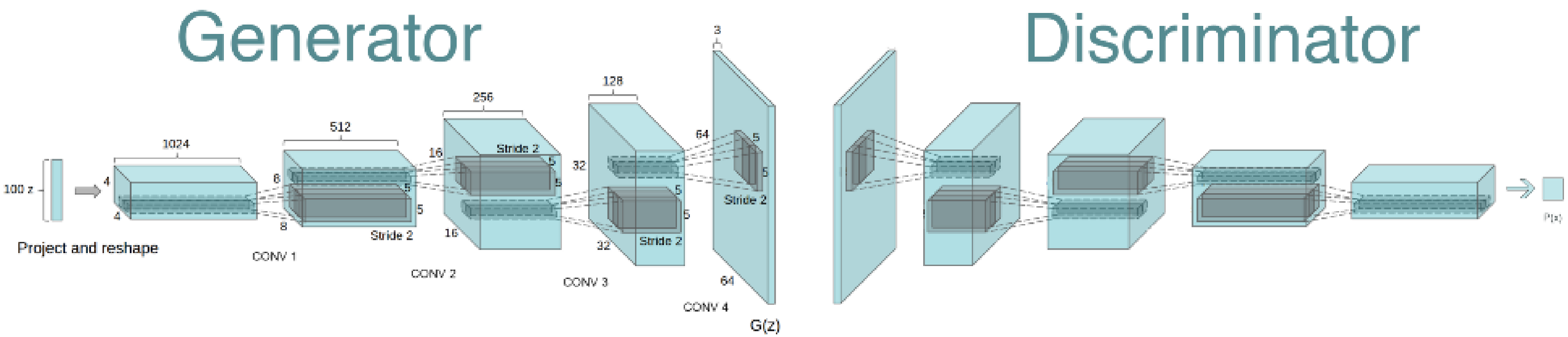
- I suggest to check this implementation:

  https://github.com/nanopony/keras-convautoencoder

Index

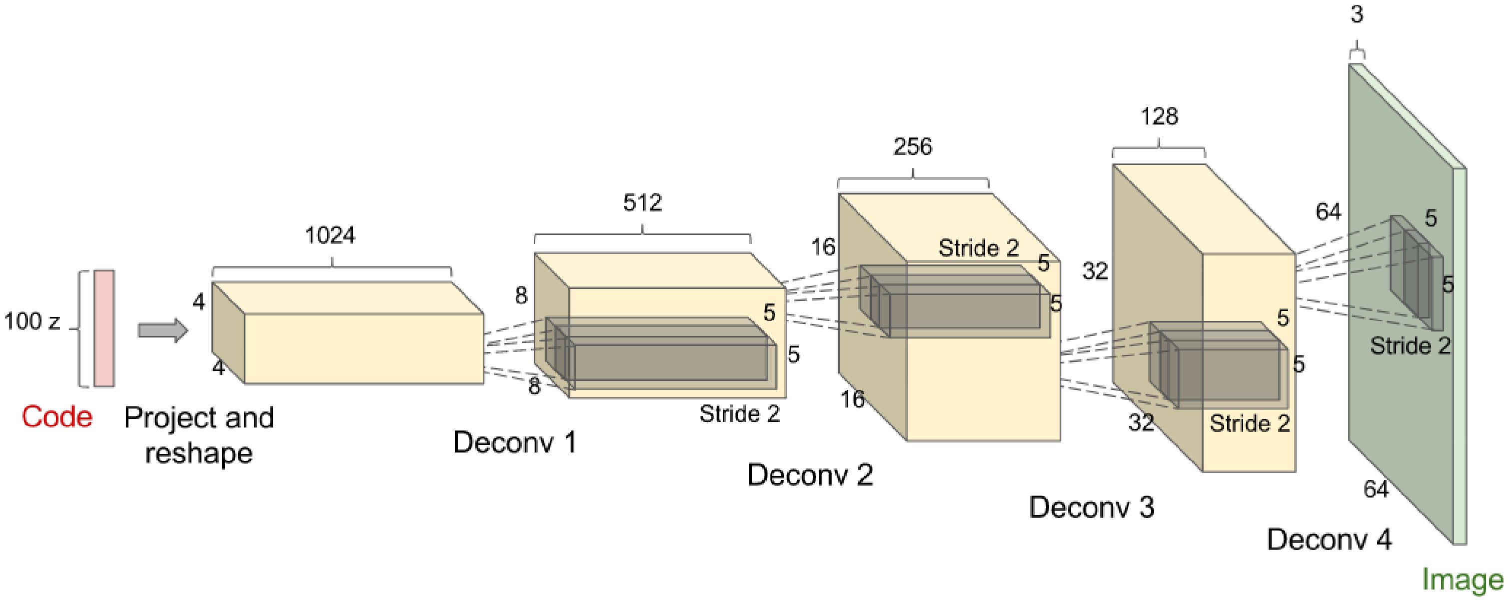
DCGAN

- DeConvolution for the Generator
- Convolution for the Discriminator



DCGAN

- Can you determine the operations involved in the Generator?



DCGAN

- In TensorFlow deconv2D:

```
tf.nn.conv2d_transpose(
    value,
    filter,
    output_shape,
    strides,
    padding='SAME',
    data_format='NHWC',
    name=None
)
```

DCGAN

- In TensorFlow (https://github.com/carpedm20/DCGAN-tensorflow)

```
self.z_, self.h0_w, self.h0_b = linear(
z, self.gf_dim*8*s_h16*s_w16, 'g_h0_lin', with_w=True)

self.h0 = tf.reshape(
self.z_, [-1, s_h16, s_w16, self.gf_dim * 8])
h0 = tf.nn.relu(self.g_bn0(self.h0))

self.h1, self.h1_w, self.h1_b = deconv2d(
h0, [self.batch_size, s_h8, s_w8, self.gf_dim*4], name='g_h1', with_w=True)
h1 = tf.nn.relu(self.g_bn1(self.h1))

h2, self.h2_w, self.h2_b = deconv2d(
h1, [self.batch_size, s_h4, s_w4, self.gf_dim*2], name='g_h2', with_w=True)
h2 = tf.nn.relu(self.g_bn2(h2))

h3, self.h3_w, self.h3_b = deconv2d(
h2, [self.batch_size, s_h2, s_w2, self.gf_dim*1], name='g_h3', with_w=True)
h3 = tf.nn.relu(self.g_bn3(h3))

h4, self.h4_w, self.h4_b = deconv2d(
h3, [self.batch_size, s_h, s_w, self.c_dim], name='g_h4', with_w=True)
```

DCGANS

- Some Keras implementation use Conv2D and UpSampling:

```
def generator_model():
    model = Sequential()
    model.add(Dense(input_dim=100, output_dim=1024))
    model.add(Activation('tanh'))
    model.add(Dense(128*7*7))
    model.add(BatchNormalization())
    model.add(Activation('tanh'))
    model.add(Reshape((7, 7, 128), input_shape=(128*7*7,)))
    model.add(UpSampling2D(size=(2, 2)))
    model.add(Conv2D(64, (5, 5), padding='same'))
    model.add(Activation('tanh'))
    model.add(UpSampling2D(size=(2, 2)))
    model.add(Conv2D(1, (5, 5), padding='same'))
    model.add(Activation('tanh'))
    return model
```