What's new in Deep Learning

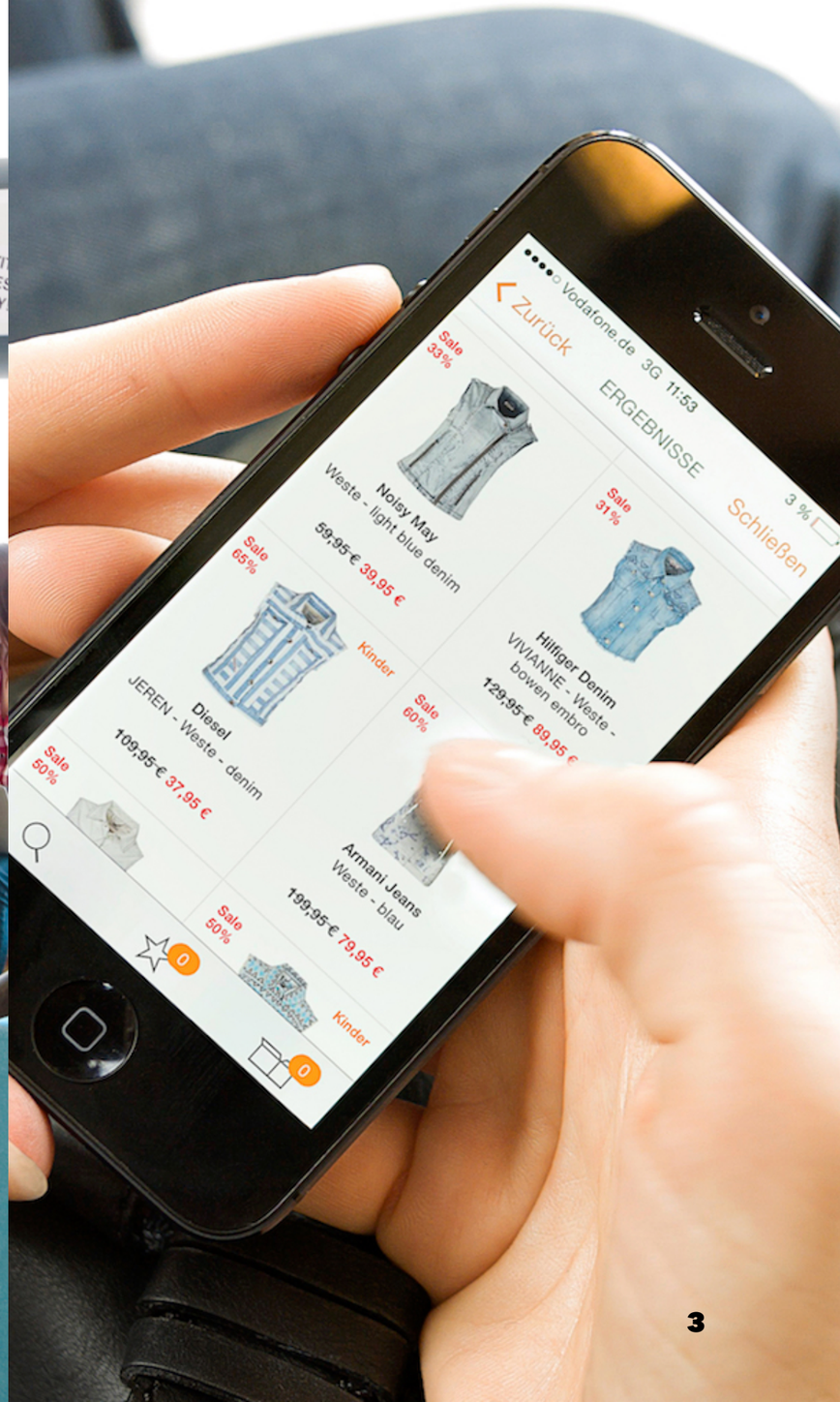
Dr. Kashif Rasul (Zalando SE)

PyData Berlin 21.5.2016

Twitter: @krasul & Github: kashif


zalando


Open Questions
ASOS
Amazon
Schuhe
2015
no fair
unfair
problem
DEFER
JUDGEMENT!
BE VISUAL
BUILD ON THE IDEAS OF OTHERS!
ONE PERSON AT A TIME


Zurück
ERGEBNISSE
Schließen
Sale 33%
Noisy May
Weste - light blue denim
59,95 € 39,95 €
Sale 31%
Hilfiger Denim
VIVIANNE - Weste -
bowen embro
129,95 € 89,95 €
Sale 65%
Kinder
Diesel
JEREN - Weste - denim
109,95 € 37,95 €
Sale 60%
Sale 50%
Armani Jeans
Weste - blau
199,95 € 79,95 €
Sale 50%
Kinder

Andrej Karpathy
@karpathy

Following

BatchNorm, STN, DCGAN, DRAW, soft/hard attention, char-rnn, DeepDream, NeuralStyle, TensorFlow, ResNet, AlphaGo.. a lot happened over 1 year

RETWEETS	LIKES
144	315

8:47 PM - 14 Mar 2016

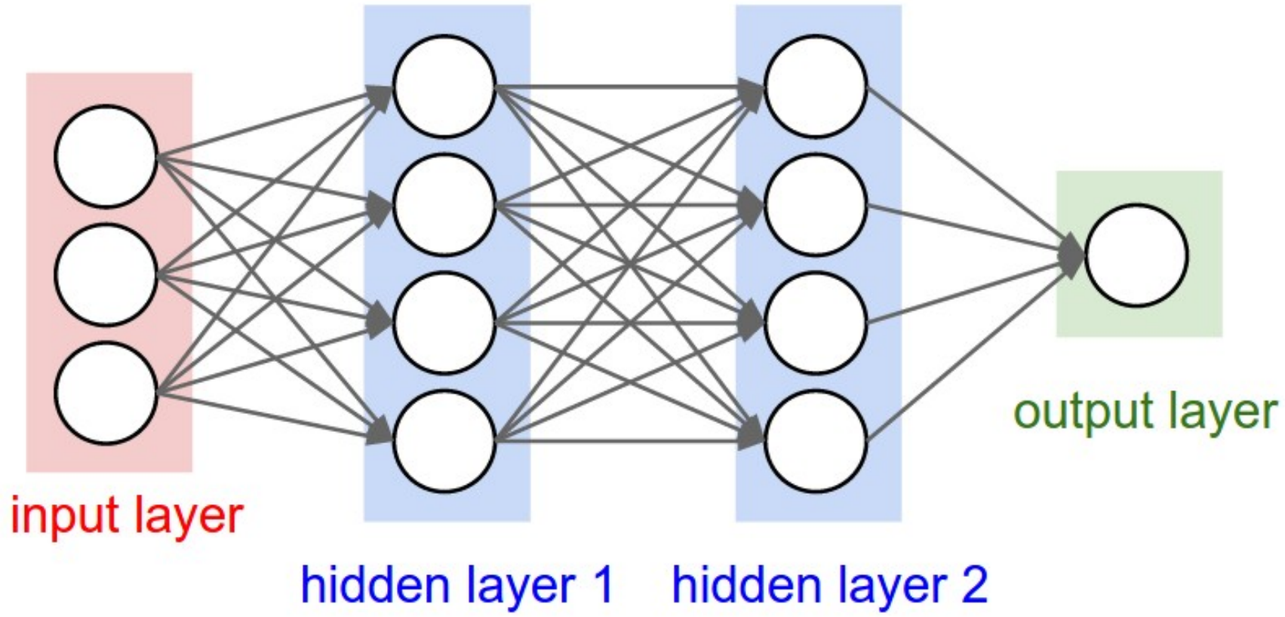
Agenda

- Introduction
- Initialization
- Batch Normalization
- Residual Networks
- Keras implementation

Data Driven Approach

- **Score**: $f(\vec{x}_i; W) = W\vec{x}_i$
- **Loss**: $L = \frac{1}{n}\sum_{i=1}^{n} L_i + \lambda R(W)$
- Use training data to find a W that minimizes L
- **Optimization**: change W in the direction of $-\partial L/\partial W$ to find the optimal W

Neural Networks (NN)



- Linear: $f(\vec{x}_i; W) = W\vec{x}_i$
- 2 Layer NN: $f = W_2 \max(0, W_1 \vec{x}_i)$
- 3 Layer NN:
  $f = W_3 \max(0, W_2 \max(0, W_1 \vec{x}_i))$
- Gradient of loss with respect to W_i via back-propagation (chain rule!)

```
def model(x, w_h, w_o):
  # ReLU layer
  h = T.maximum(0, T.dot(x, w_h))
  # Softmax layer
  p_y_given_x = T.nnet.softmax(T.dot(h, w_o))
  return p_y_given_x

w_h = init_weights((32 * 32, 100))
w_o = init_weights((100, 10))

p_y_given_x = model(x, w_h, w_o)

y_hat = T.argmax(p_y_given_x, axis=1)
```

```
cost = T.mean(
    T.nnet.categorical_crossentropy(p_y_given_x, y))

def sgd(cost, params, learning_rate):
  grads = T.grad(cost, params) # 😹 magic!
  updates = []
  for p, g in zip(params, grads):
    updates.append([p, p - g * learning_rate])
  return updates

updates = sgd(cost, [w_h, w_o], learning_rate=0.01)
```

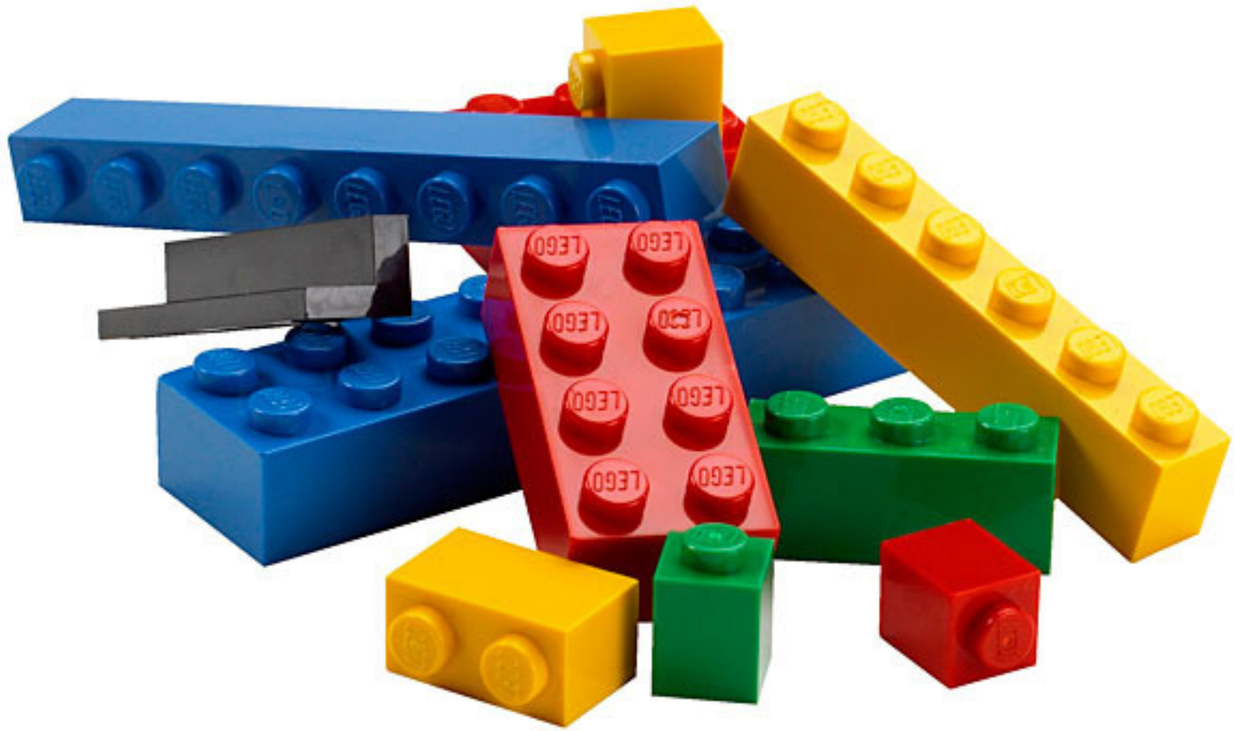
```
train = theano.function([x, y], cost, updates=updates)

predict = theano.function([x], y_hat)

batch_size = 50

for i in range(20):
  for start in range(0, len(X_train), batch_size):
    X_batch = X_train[start:start + batch_size]
    y_batch = y_train[start:start + batch_size]
    cost = train(X_batch, y_batch)

  predictions_test = predict(X_test)
  accuracy = np.mean(predictions_test == y_test)
  print "iteration %d: accuracy: %.5f" % ((i+1), accuracy)
```



General Implementation

```
class SomeLayer(object):
  def __init__(self, ...):
    ...

  # Forward API
  def forward(self,...):
    ...

  # Backpropagation API
  def backward(self, ...):
    ...
```

Initialization

- If the weights of the network start off too small 👉 signal shrinks 🐌
- If they are too large: signal blows up 💣
- Xavier (or Glorot) initialization (2010): weights "just right" then signal in a reasonable range

Intuition

- For $f = W\vec{x} = \sum_{i=1}^{n_{\text{in}}} W_i x_i$ we have:

  $$\mathbf{Var}(W_i x_i) = [\mathbb{E}(x_i)]^2 \mathbf{Var}(W_i) + [\mathbb{E}(W_i)]^2 \mathbf{Var}(x_i) + \mathbf{Var}(x_i)\mathbf{Var}(W_i)$$

- If x_i and W_i have mean 0 and are independent random then $\mathbf{Var}(f) = n_{\text{in}} \mathbf{Var}(x_i)\mathbf{Var}(W_i)$

- For the variance of input and output to be the same set $\mathbf{Var}(W_i) = 1/n_{\text{in}}$

- Gradient to have the same variance: $\mathbf{Var}(W_i) = 1/n_{\text{out}}$
- Compromise: $\mathbf{Var}(W_i) = 2/(n_{\text{in}} + n_{\text{out}})$

```
def glorot_normal(shape, name=None, dim_ordering='th'):
    ''' Reference: Glorot & Bengio, AISTATS 2010
    '''
    fan_in, fan_out = get_fans(shape, dim_ordering=dim_ordering)
    s = np.sqrt(2. / (fan_in + fan_out))
    return normal(shape, s, name=name)
```

Delving Deep into Rectifiers (2015)

- Instead of identity consider ReLU non-linearites
- Then: $\mathbf{Var}(f) = n\mathbb{E}(x_i^2)\mathbf{Var}(W_i)$ since mean of $\mathbf{max}(0, x_i)$ is not 0
- So for variance of input and output to be the same we need: $\mathbf{Var}(W_i) = 2/n_{\text{in}}$

```
def he_normal(shape, name=None, dim_ordering='th'):
    ''' Reference:  He et al., http://arxiv.org/abs/1502.01852
    '''
    fan_in, _ = get_fans(shape, dim_ordering=dim_ordering)
    s = np.sqrt(2. / fan_in)
    return normal(shape, s, name=name)
```

Optimal performance?*

- May be using the wrong criteria
- The weight properties do not persist as learning starts
- Learning speed increases but inadvertently so does the generalization error
- For large layers (big n_{in}), standard deviation like $\sqrt{2/n_{\text{in}}}$ means extremely small weights

---

* Deep Learning Book (2016)

Batch Normalization (BN) (2015)

- Very deep networks 👉 composition of many layers 👉 interactions
- Simultaneous updates can thus cause unexpected results
- Batch Normalization re-parametrization reduces this issue

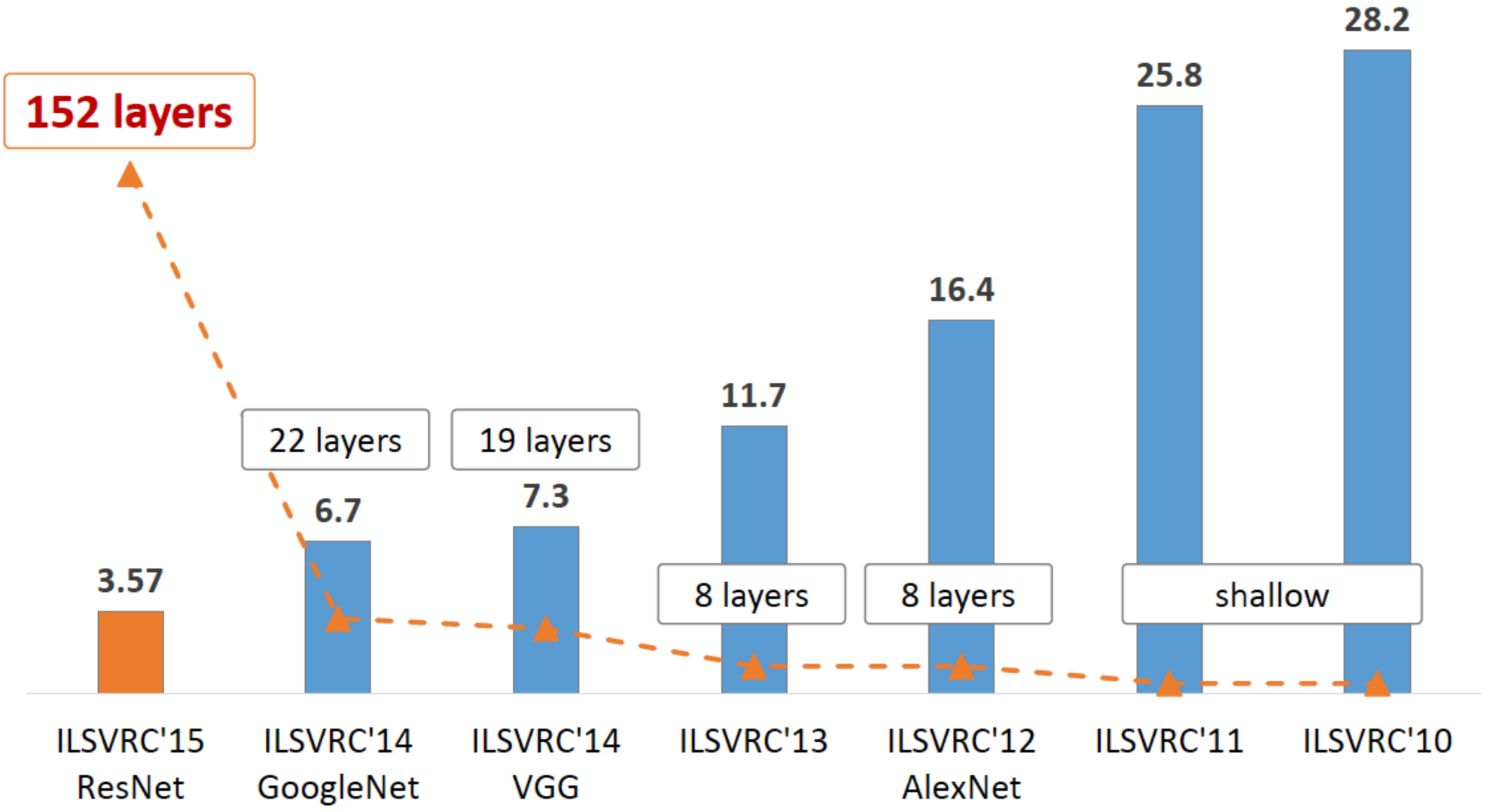
BN Forward pass

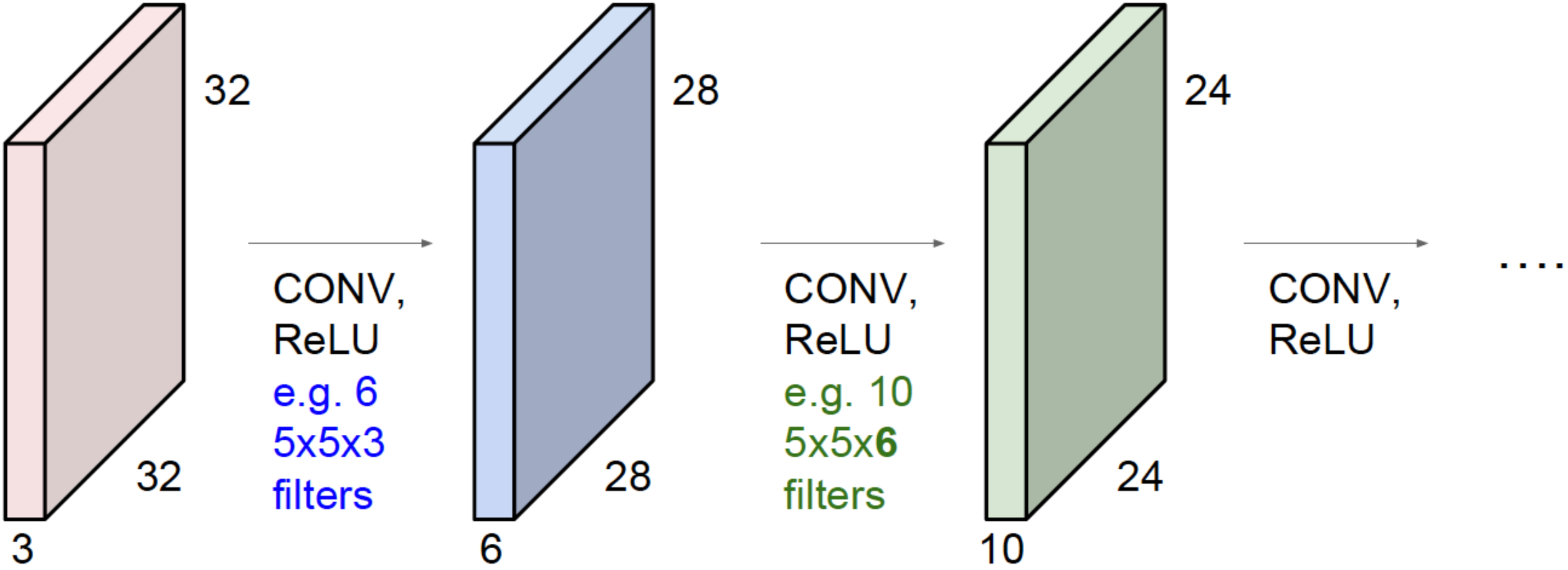
- Calculate sample mean and sample variance from mini-batch
- Normalize data to have mean 0 and variance 1
- Keep a running mean and variance of data to normalize at test time:

```
running_mean = momentum * running_mean + (1 - momentum) * sample_mean
running_var  = momentum * running_var  + (1 - momentum) * sample_var
```

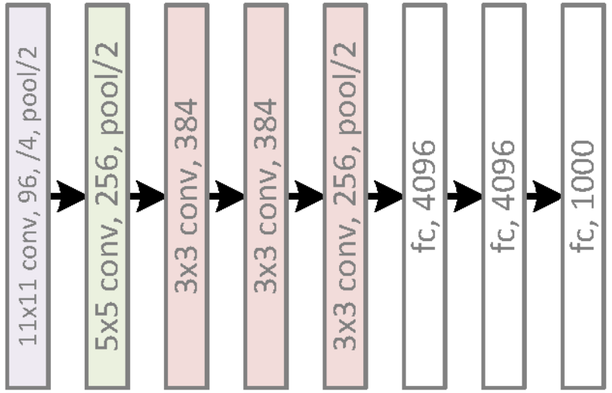
Accelerating Networks with BN

- Increase learning rate
- Remove Dropout (2014)
- Accelerate learning rate annealing
- Shuffle training examples more thoroughly
- Reduce image augmentation

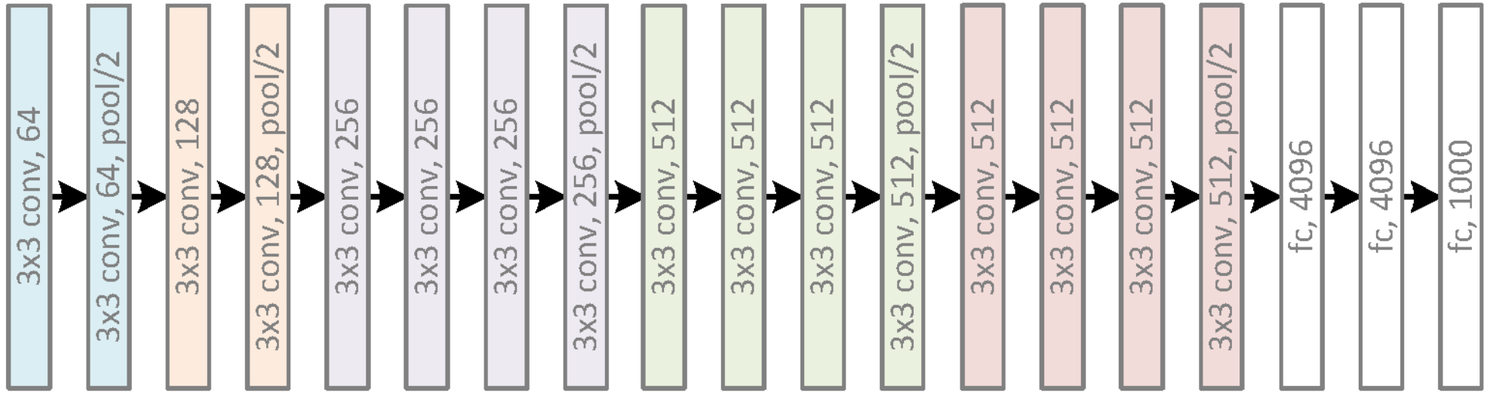

152 layers
22 layers
19 layers
8 layers
8 layers
shallow
3.57
6.7
7.3
11.7
16.4
25.8
28.2
ILSVRC'15 ResNet
ILSVRC'14 GoogleNet
ILSVRC'14 VGG
ILSVRC'13
ILSVRC'12 AlexNet
ILSVRC'11
ILSVRC'10
ImageNet Classification top-5 error (%)

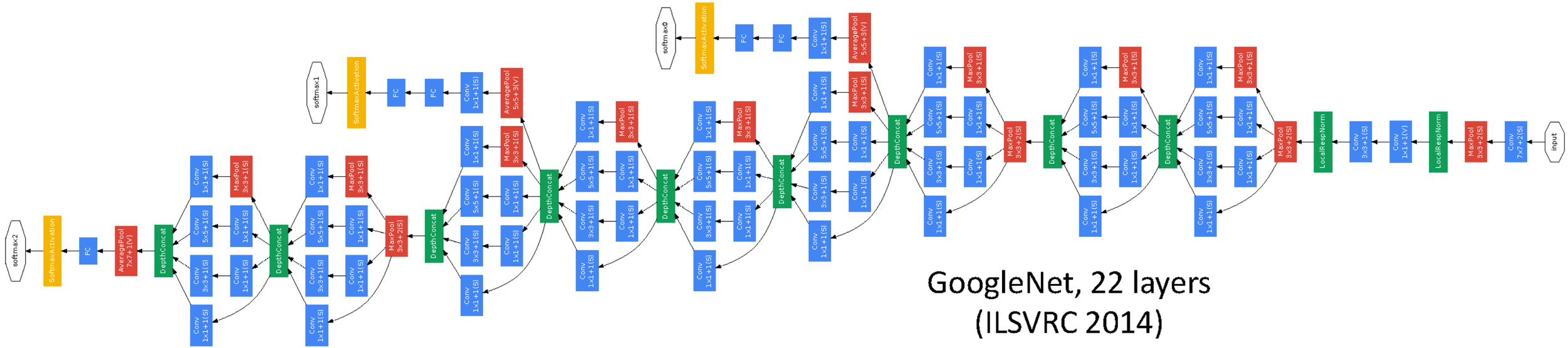

32
32
3
CONV,
ReLU
e.g. 6
5x5x3
filters
28
28
6
CONV,
ReLU
e.g. 10
5x5x6
filters
24
24
10
CONV,
ReLU
....

AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)



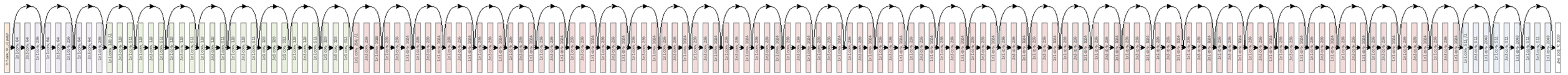


GoogleNet, 22 layers
(ILSVRC 2014)

AlexNet, 8 layers
(ILSVRC 2012)

VGG, 19 layers
(ILSVRC 2014)

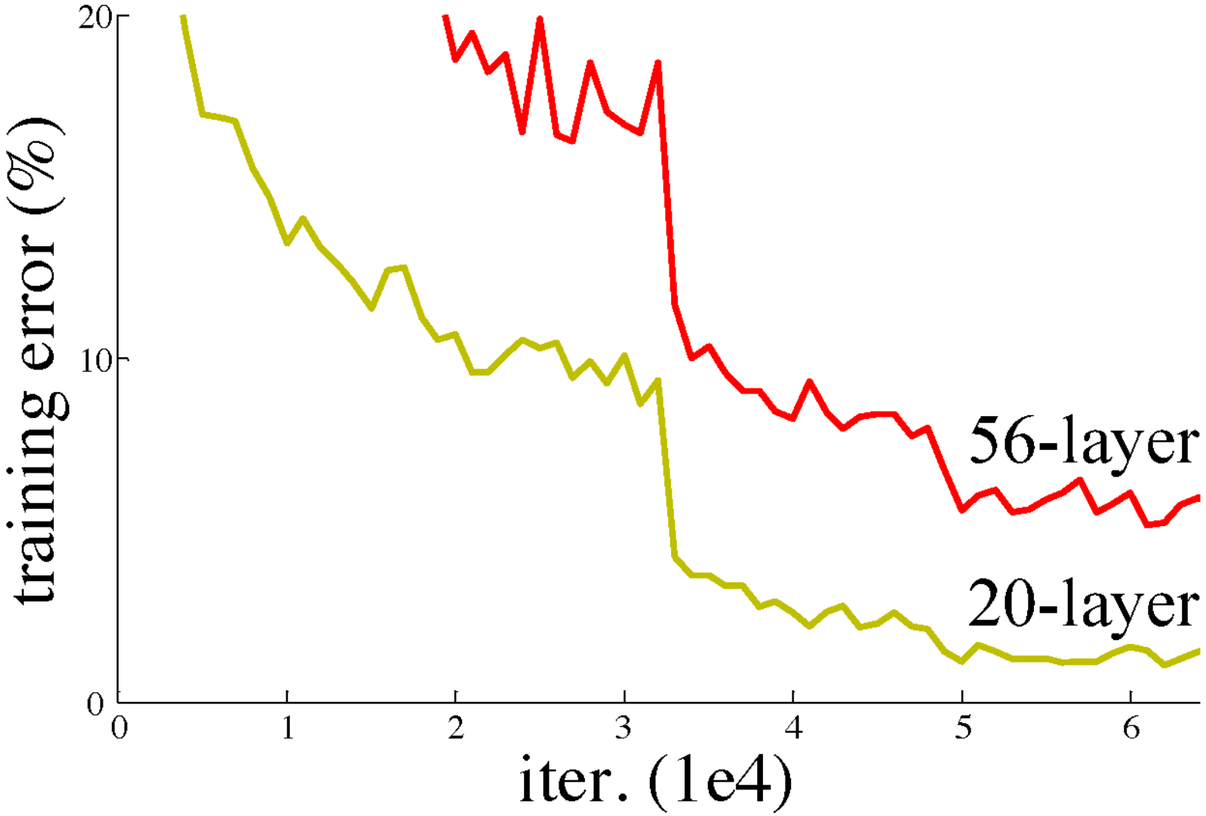
GoogleNet, 22 layers
(ILSVRC 2014)

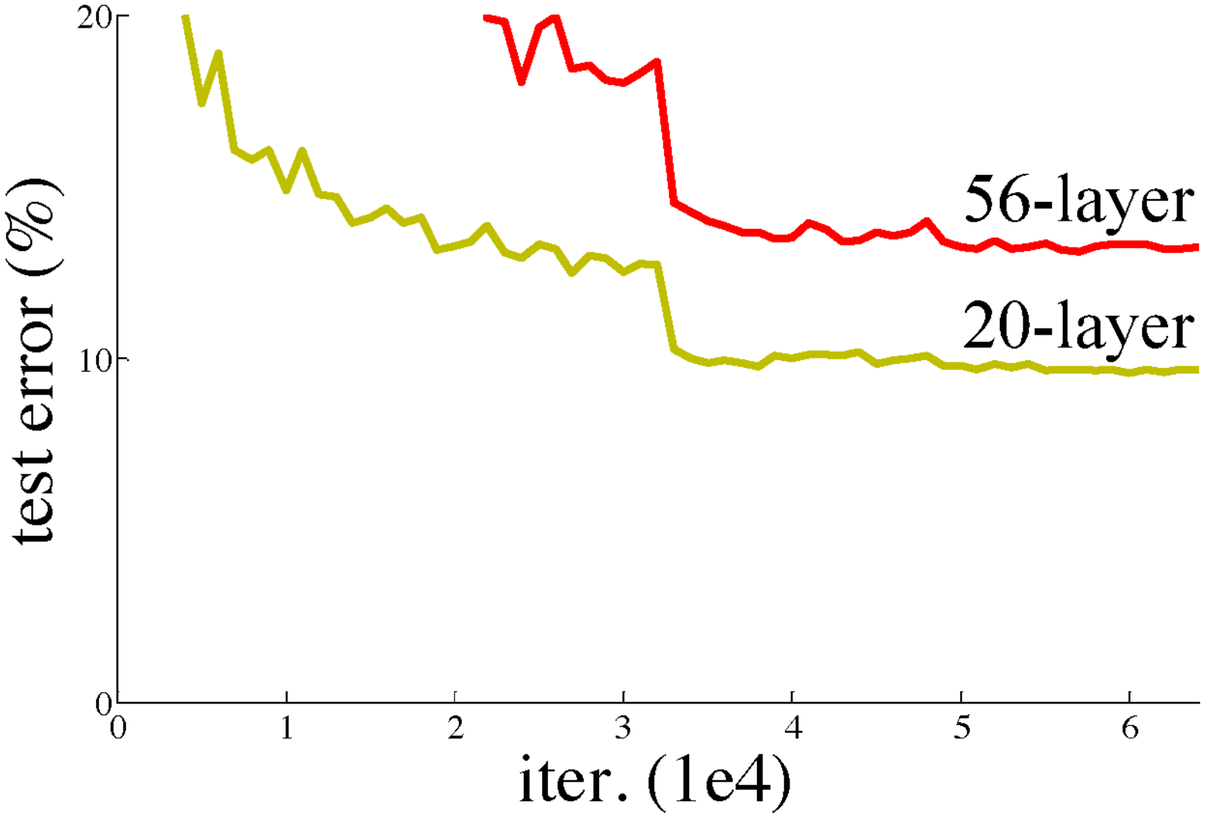


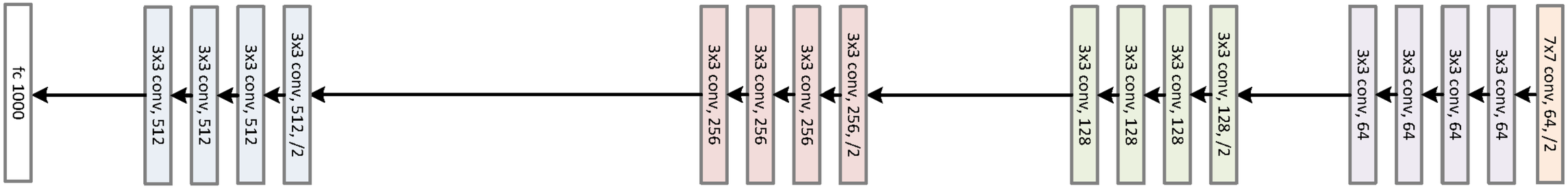
ResNet, 152 layers
(ILSVRC 2015)

Is learning better networks as easy as stacking more layers?

🤔 WTF!

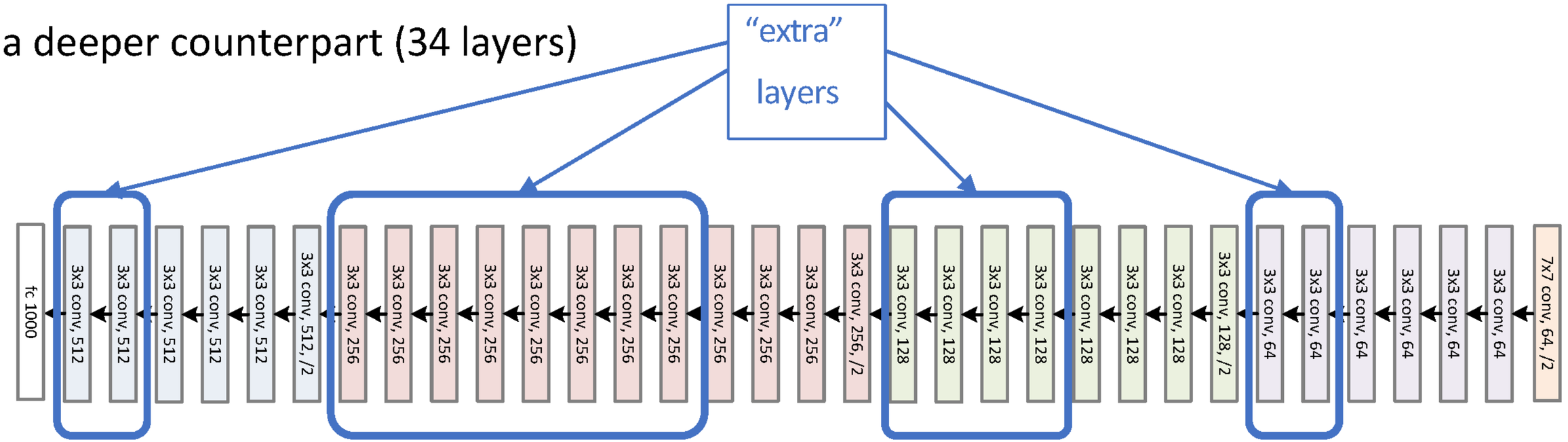

training error (%)
20
10
0
0
1
2
3
4
5
6
iter. (1e4)
56-layer
20-layer


test error (%)
20
10
0
0
1
2
3
4
5
6
iter. (1e4)
56-layer
20-layer

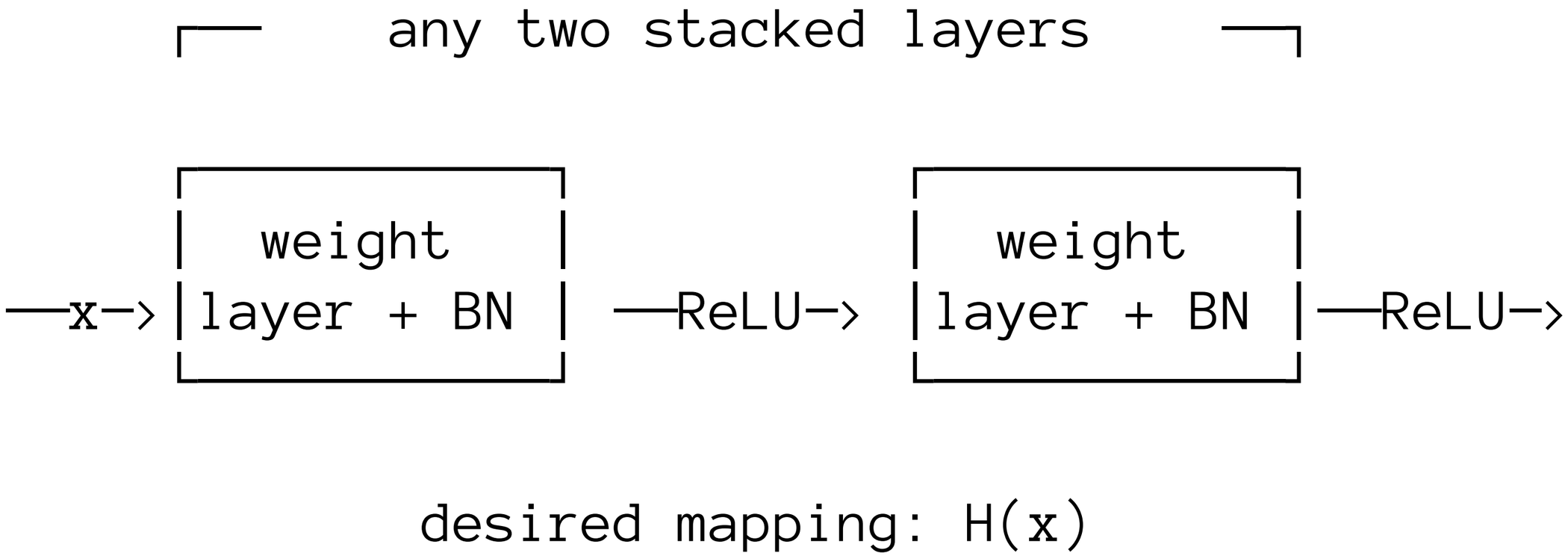


a shallower model (18 layers)

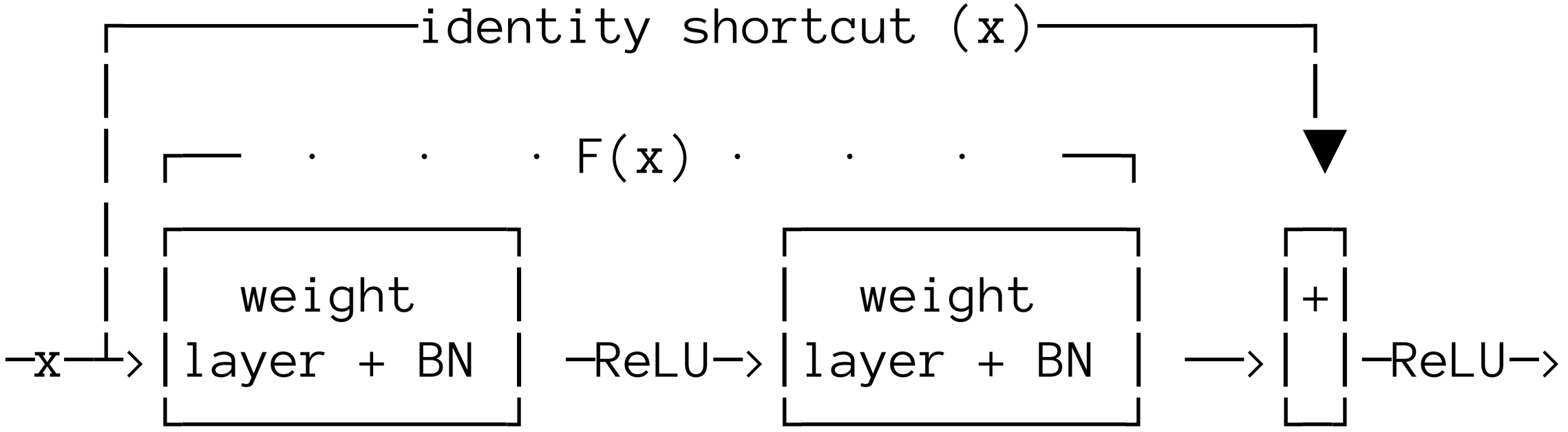
a deeper counterpart (34 layers)



Plain Network



Residual Network (2015)

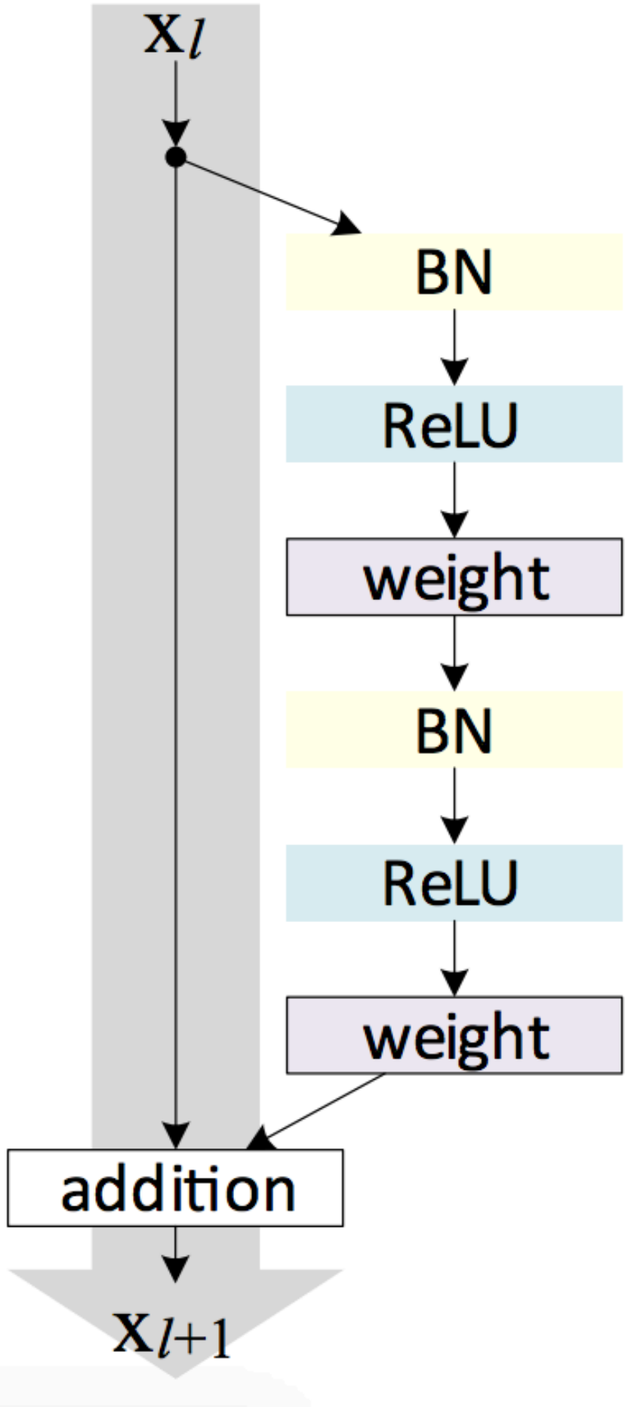


Shortcut Connections

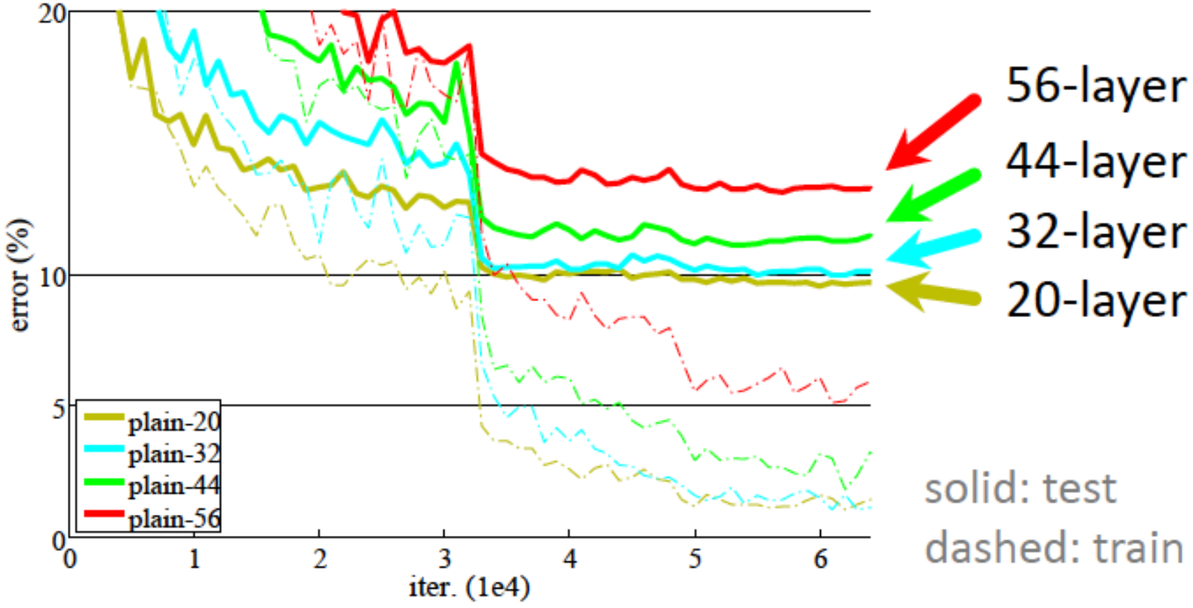
- Add a linear layer connected from the network input to the output by Ripley 1996
- A few intermediate layers are directly connected to auxiliary classifiers by Lee et al. 2014 or Szegedy et al. 2015
- "Inception" layer composed of a shortcut branch and a few deeper branches by Szegedy et al. 2015
- Highway networks: shortcuts with gating functions by Schmidhuber et al. 2015

New ResNet (2016)

- **Deep** and VGG-Style:
  - All convolutions proceeded by Batch Normalization and ReLU
  - When spatial size /2 then increase number of filters x2
  - Xavier2 initialization
  - SGD + Momentum (0.9)
- No: Max pooling, hidden fully connected layers or Dropout



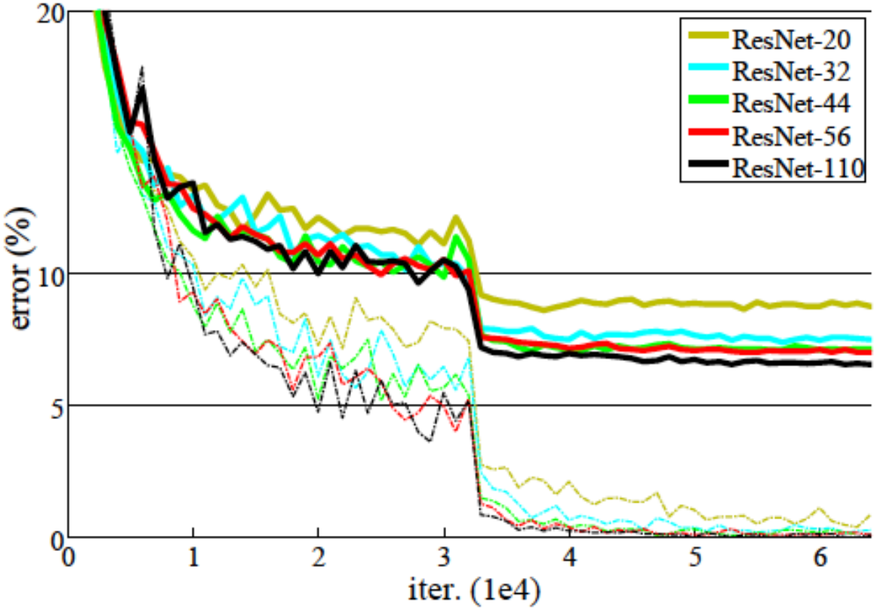
CIFAR-10 plain nets



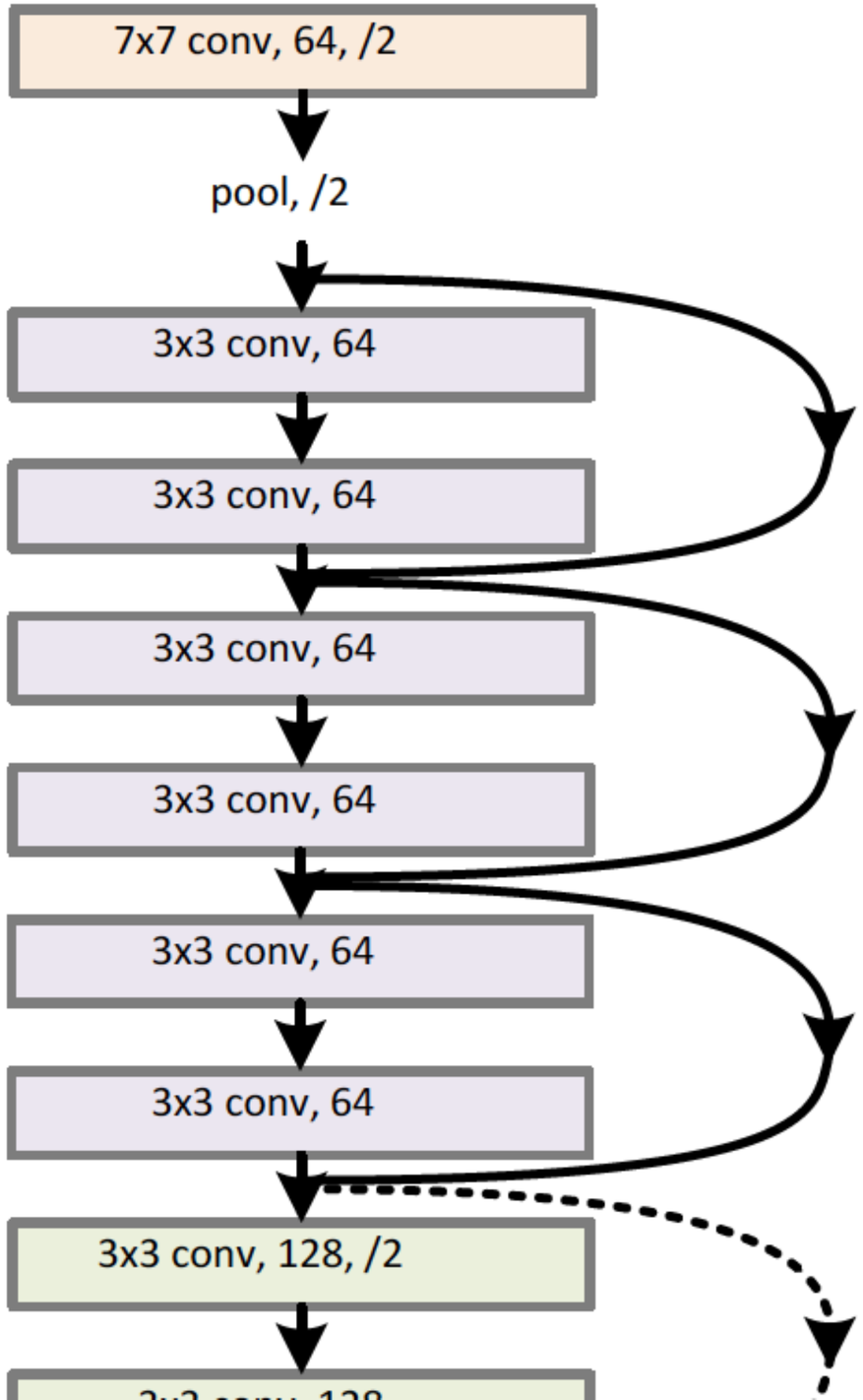
56-layer
44-layer
32-layer
20-layer

solid: test
dashed: train

CIFAR-10 ResNets



20-layer
32-layer
44-layer
56-layer
110-layer


7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2

```
def bottleneck(x, count, nb_in_filt, nb_out_filt, subsample):
  # first one we change the number of filters
  x = bottleneck_unit(x, nb_in_filt, nb_out_filt, subsample)

  # for the rest there is no change
  for _ in range(1, count):
    x = bottleneck_unit(x, nb_out_filt, nb_out_filt)

  return x
```

```
def bottleneck_unit(x, nb_in_filt, nb_out_filt, subsample=(1,1)):
  nb_bottleneck_filt = nb_out_filt/4

  # the case when there is no change in number of in/out filters
  if nb_in_filt == nb_out_filt:
    # conv1x1
    y = BatchNormalization(axis=1)(x)
    y = Activation('relu')(y)
    y = Convolution2D(nb_bottleneck_filt, nb_row=1, nb_col=1,
          subsample=subsample, init='he_normal',
          border_mode='same')(y)
    ...
    return merge([x, y], mode='sum')
  else: # Residual Units for increasing dimensions
    ...
```

```
img_input = Input(shape=(img_channels, img_rows, img_cols))

# one conv at the beginning (spatial size: 32x32)
x = ZeroPadding2D((1,1))(img_input)
x = Convolution2D(16, nb_row=3, nb_col=3, subsample=(1,1))(x)

n = 3
# Stage 1 (spatial size: 32x32)
x = bottleneck(x, n, 16, 64, subsample=(1,1))
# Stage 2 (spatial size: 16x16)
x = bottleneck(x, n, 64, 128, subsample=(2,2))
# Stage 3 (spatial size: 8x8)
x = bottleneck(x, n, 128, 256, subsample=(2,2))
```

```
x = BatchNormalization(axis=1)(x)
x = Activation('relu')(x)
x = AveragePooling2D((8,8), strides=(1,1))(x)
x = Flatten()(x)
preds = Dense(nb_classes, activation='softmax')(x)

model = Model(input=img_input, output=preds)
model.compile(optimizer='adam', loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(X_train, Y_train, batch_size=batch_size,
          nb_epoch=nb_epoch, validation_data=(X_test, Y_test),
          shuffle=True)
```

```
Epoch 1/200
50000/50000 [==============================] - 233s - loss: 1.6311 - acc: 0.3946
- val_loss: 2.2076 - val_acc: 0.2346
Epoch 2/200
50000/50000 [==============================] - 233s - loss: 1.2752 - acc: 0.5363
- val_loss: 2.7894 - val_acc: 0.2408
Epoch 3/200
50000/50000 [==============================] - 231s - loss: 1.1015 - acc: 0.6026
- val_loss: 2.5626 - val_acc: 0.2953
Epoch 4/200
50000/50000 [==============================] - 233s - loss: 0.9716 - acc: 0.6507
- val_loss: 4.1231 - val_acc: 0.1928
Epoch 5/200
50000/50000 [==============================] - 233s - loss: 0.8825 - acc: 0.6863
- val_loss: 2.7574 - val_acc: 0.1543
Epoch 6/200
 6144/50000 [==>...........................] - ETA: 171s - loss: 0.8195 - acc: 0.7041
```

Summary

- Assumptions of initialization
- Issues with learning
- Use Batch Normalization
- Issues with very deep networks
- Residual networks
- Code: https://bit.ly/cifar10-resnet

Thank you!

Questions?