Lecture 12
Network Tuning

Song Chaoyang

Assistant Professor

Department of Mechanical and Energy Engineering

songcy@sustech.edu.cn

[Slides adapted from CS213n by Fei-Fei Li & CS230 by Andew Ng @ Stanford.]
[Slides adapted from Tensorflow and deep learning without a PhD series by Martin Gorner @ Google.]

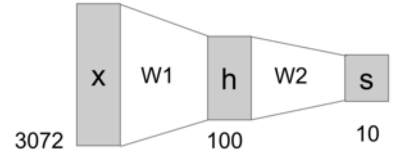
SUSTech
Southern University of Science and Technology

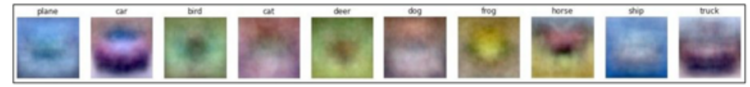
Neural Network

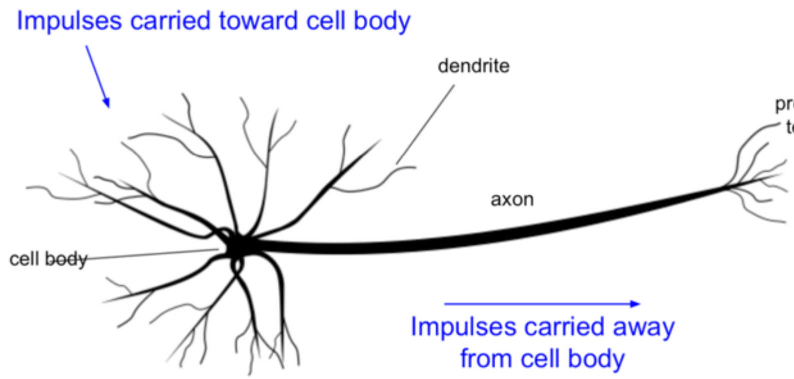
Biological Inspiration

Linear score function: $f = Wx$

2-layer Neural Network $f = W_2 \max(0, W_1 x)$

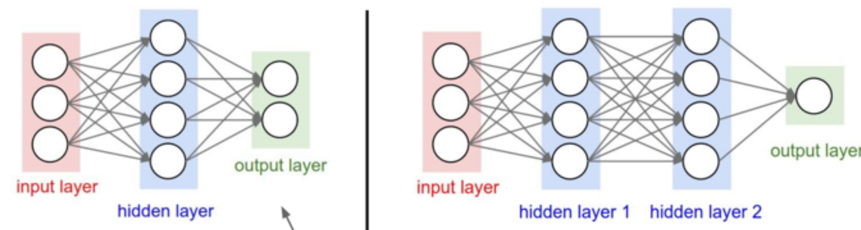






This image by Felipe Perucho is licensed under CC-BY 3.0

Neural networks: Architectures



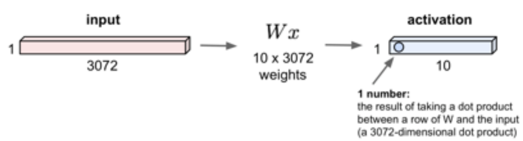
"2-layer Neural Net", or "1-hidden-layer Neural Net"

"Fully-connected" layers

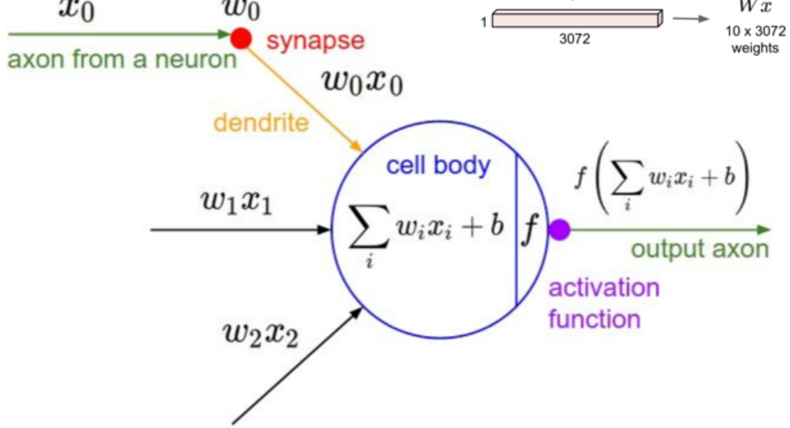
"3-layer Neural Net", or "2-hidden-layer Neural Net"

Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

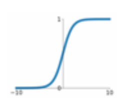



```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

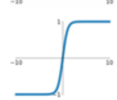


Activation functions

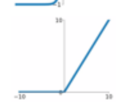
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



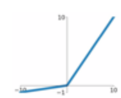
tanh
$\tanh(x)$



ReLU
$\max(0, x)$

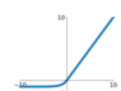


Leaky ReLU
$\max(0.1x, x)$



Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



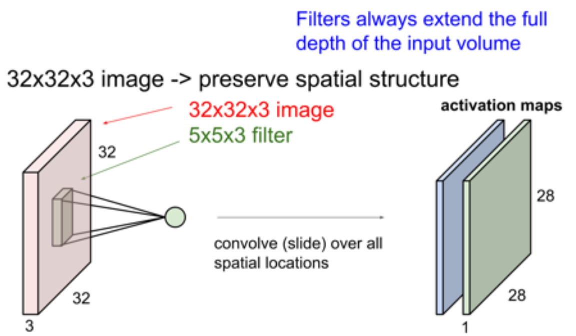
- Use ReLU. Be careful with your learning rates
- Try out Leaky ReLU / Maxout / ELU
- Try out tanh but don't expect much
- Don't use sigmoid

SUSTech
Southern University of Science and Technology

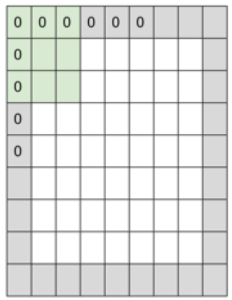
AncoraSIR.com

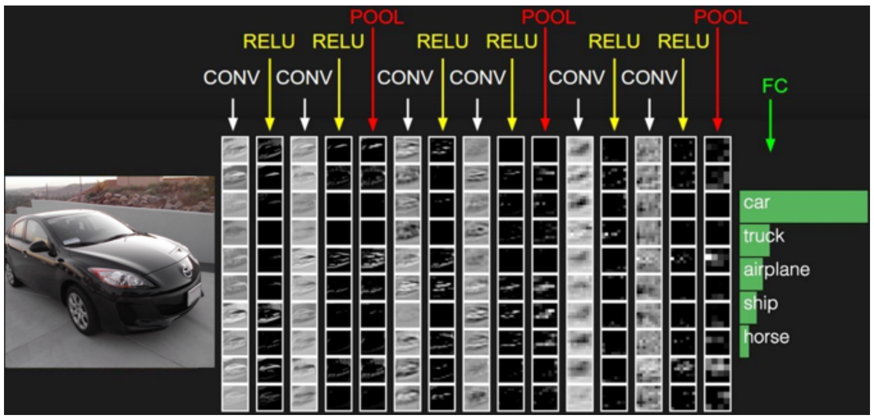
Convolutional Neural Networks

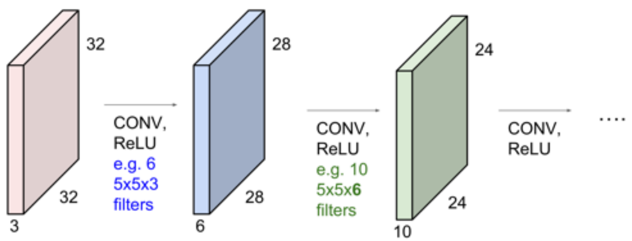
Convolve (slide) over all spatial locations



Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions







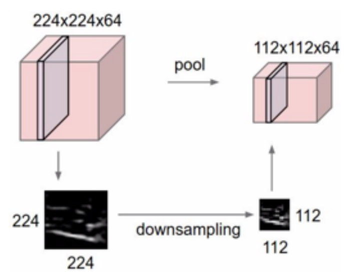
Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

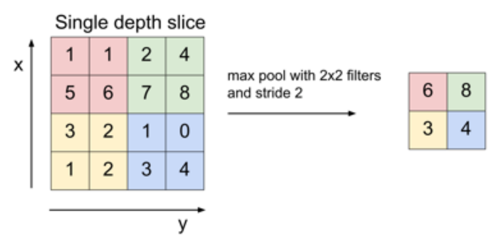
Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Pooling layer
- makes the representations smaller and more manageable
- operates over each activation map independently:



MAX POOLING



- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Typical architectures look like
  [(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K,SOFTMAX
  where N is usually up to ~5, M is large, 0 <= K <= 2.
  - but recent advances such as ResNet/GoogLeNet challenge this paradigm

SUSTech
Southern University of Science and Technology

[ConvNetJS demo: training on CIFAR-10]

https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

ConvNetJS CIFAR-10 demo

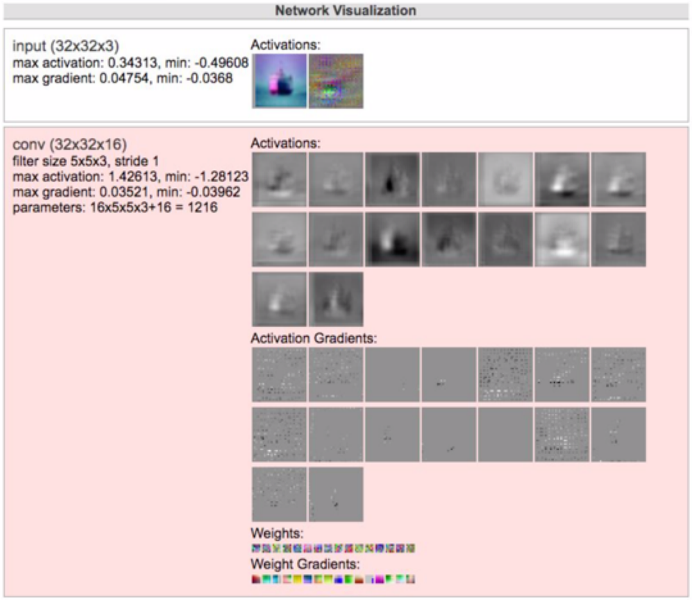
Description

This demo trains a Convolutional Neural Network on the CIFAR-10 dataset in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used this python script to parse the original files (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and verically.

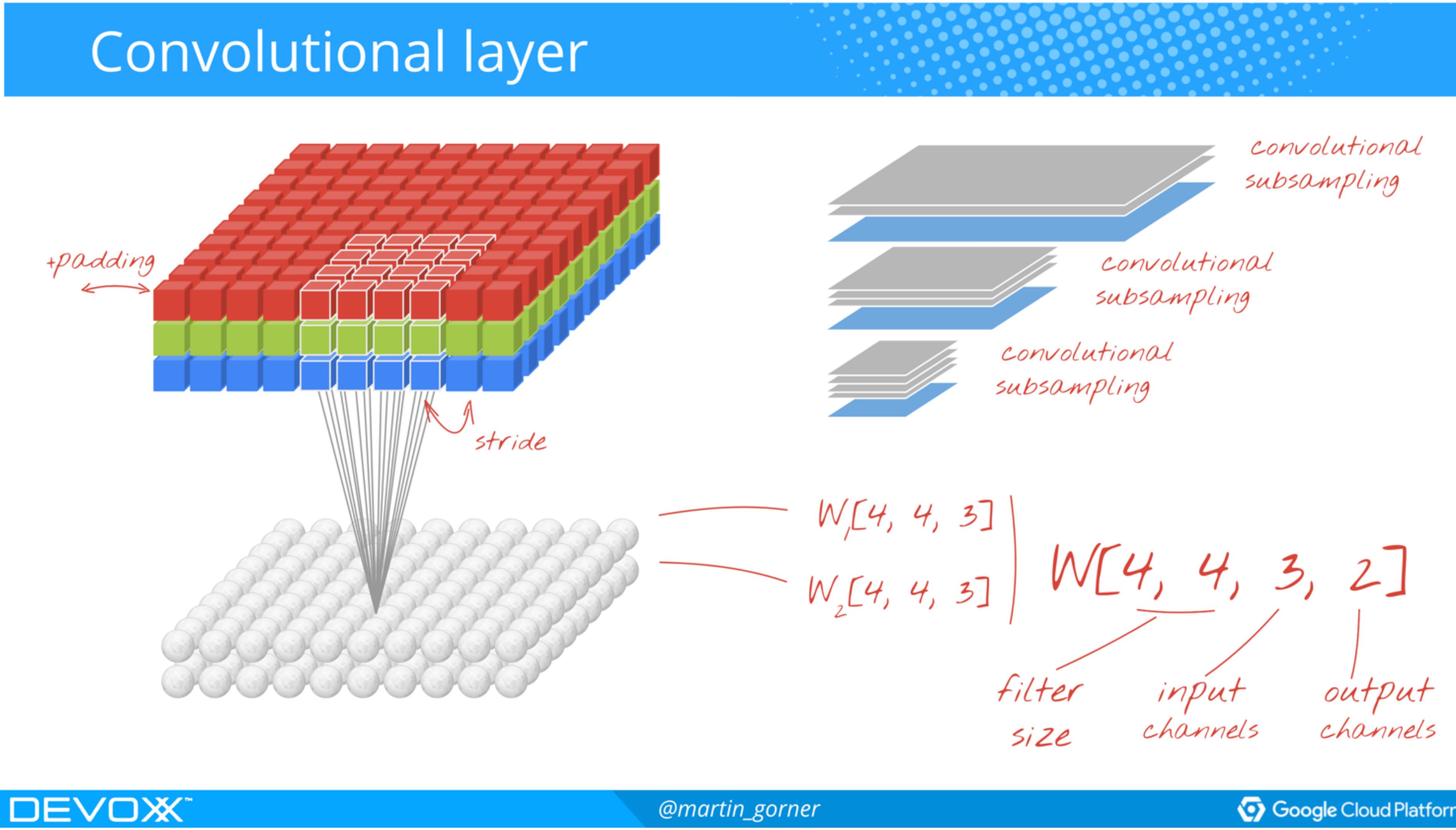
By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to @karpathy.



AncoraSIR.com

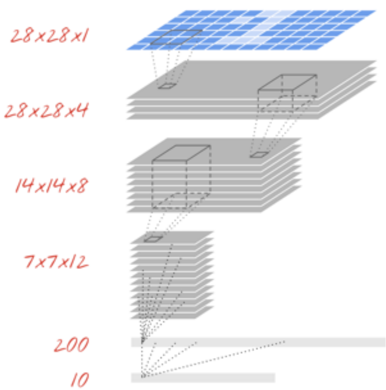
Construct of a Convolutional Layer



python mnist_3.0_convolutional.py => 98.91%

Convolutional neural network

+ biases on all layers



convolutional layer, 4 channels
W1[5, 5, 1, 4] stride 1

convolutional layer, 8 channels
W2[4, 4, 4, 8] stride 2

convolutional layer, 12 channels
W3[4, 4, 8, 12] stride 2

fully connected layer W4[7x7x12, 200]
softmax readout layer W5[200, 10]

Tensorflow - initialisation

filter size — input channels — output channels

```
K=4
L=8
M=12

W1 = tf.Variable(tf.truncated_normal([5, 5, 1, K] ,stddev=0.1))
B1 = tf.Variable(tf.ones([K])/10)
W2 = tf.Variable(tf.truncated_normal([5, 5, K, L] ,stddev=0.1))
B2 = tf.Variable(tf.ones([L])/10)
W3 = tf.Variable(tf.truncated_normal([4, 4, L, M] ,stddev=0.1))
B3 = tf.Variable(tf.ones([M])/10)

N=200

W4 = tf.Variable(tf.truncated_normal([7*7*M, N] ,stddev=0.1))
B4 = tf.Variable(tf.ones([N])/10)
W5 = tf.Variable(tf.truncated_normal([N, 10] ,stddev=0.1))
B5 = tf.Variable(tf.zeros([10])/10)
```

weights initialised with random values

Tensorflow - the model

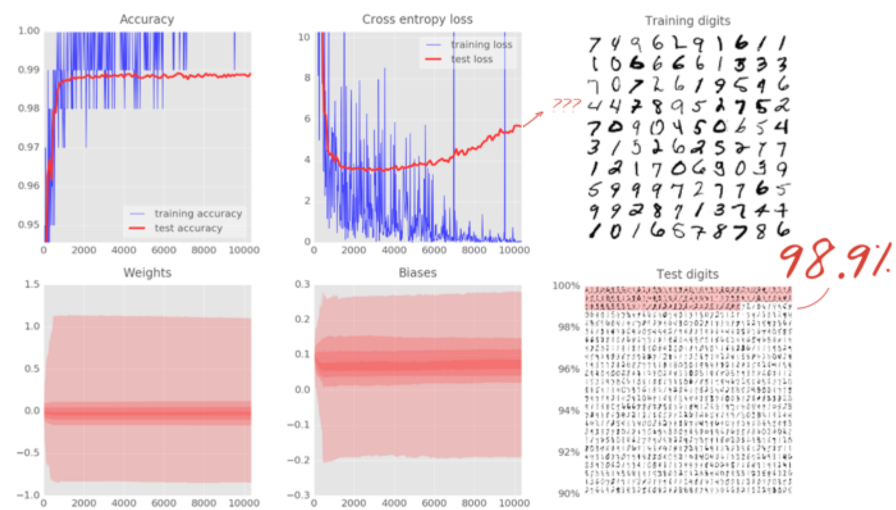
input image batch X[100, 28, 28, 1] — weights — stride — biases

```
Y1 = tf.nn.relu(tf.nn.conv2d(X, W1, strides=[1, 1, 1, 1], padding='SAME') + B1)
Y2 = tf.nn.relu(tf.nn.conv2d(Y1, W2, strides=[1, 2, 2, 1], padding='SAME') + B2)
Y3 = tf.nn.relu(tf.nn.conv2d(Y2, W3, strides=[1, 2, 2, 1], padding='SAME') + B3)

YY = tf.reshape(Y3, shape=[-1, 7 * 7 * M])

Y4 = tf.nn.relu(tf.matmul(YY, W4) + B4)
Y  = tf.nn.softmax(tf.matmul(Y4, W5) + B5)
```

flatten all values for fully connected layer

Y3 [100, 7, 7, 12]

YY [100, 7x7x12]

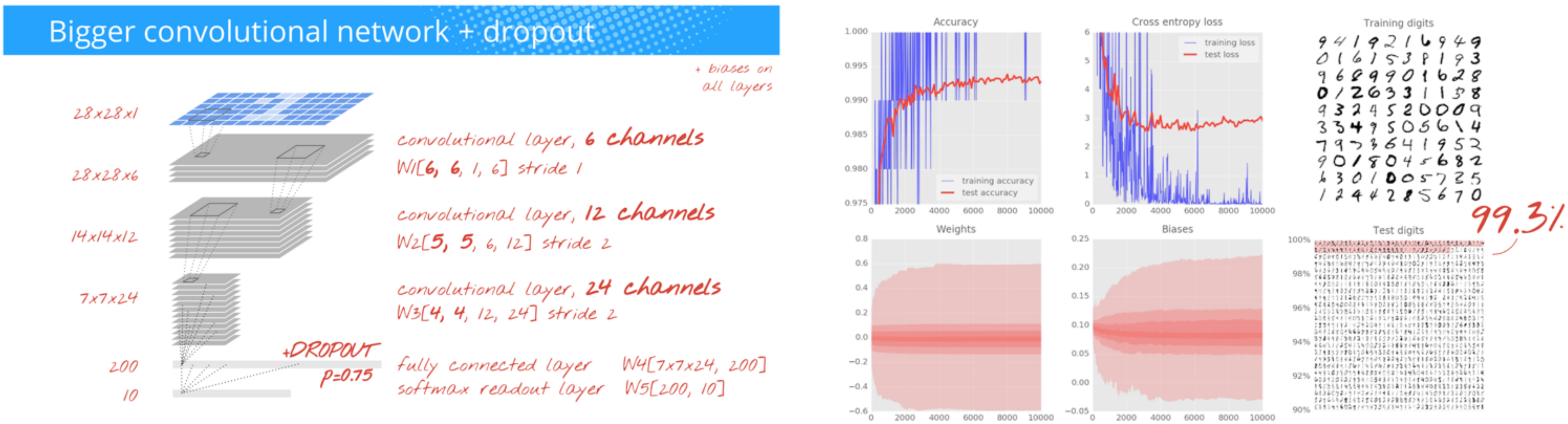


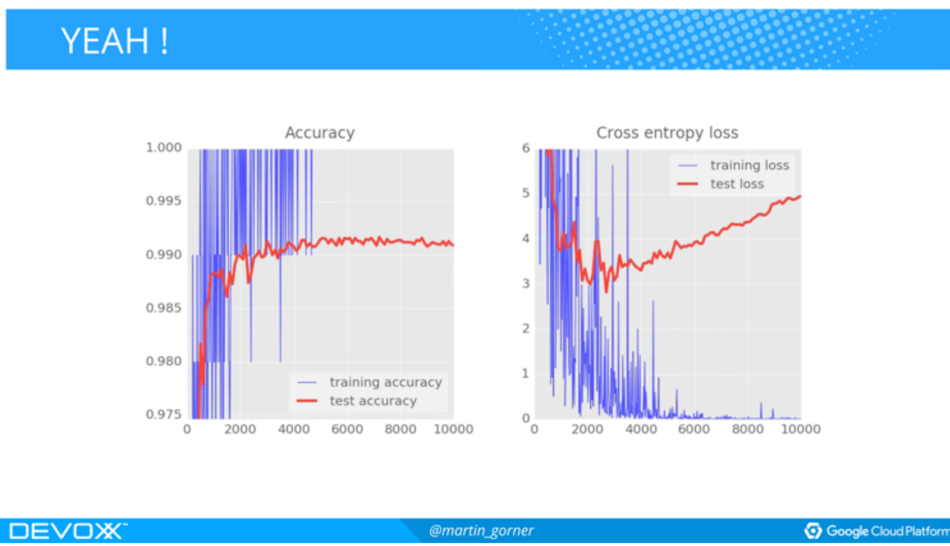
DEVOXX @martin_gorner Google Cloud Platform

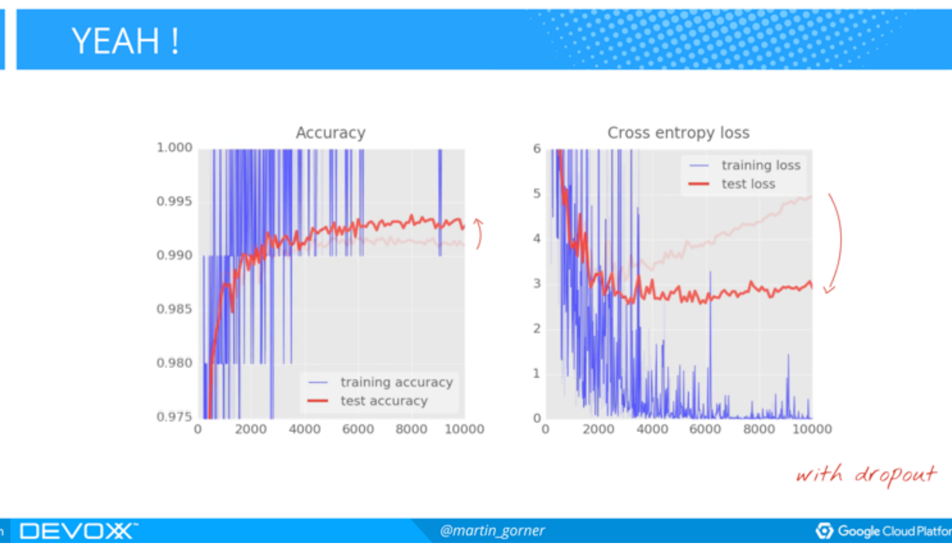
AncoraSIR.com

Southern University of Science and Technology

python mnist_3.1_convolutional_bigger_dropout.py => 99.3%



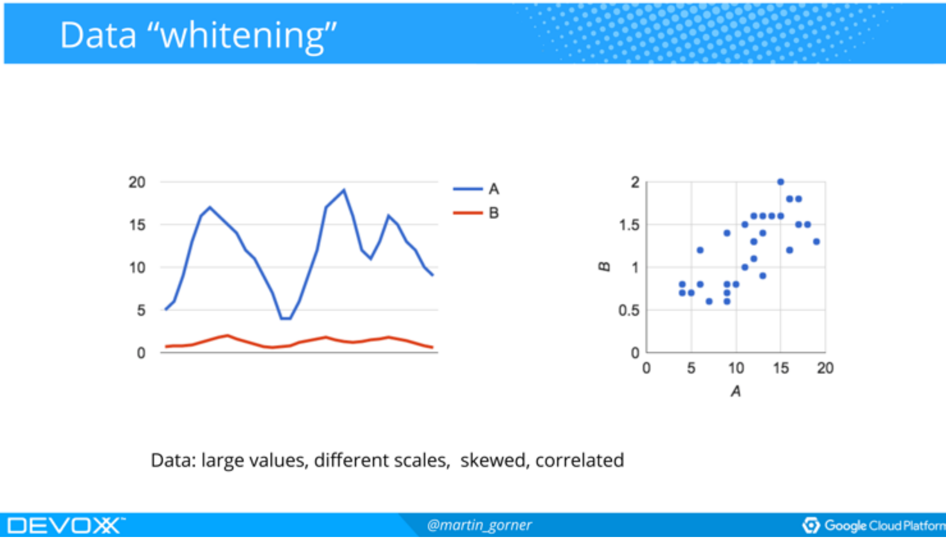


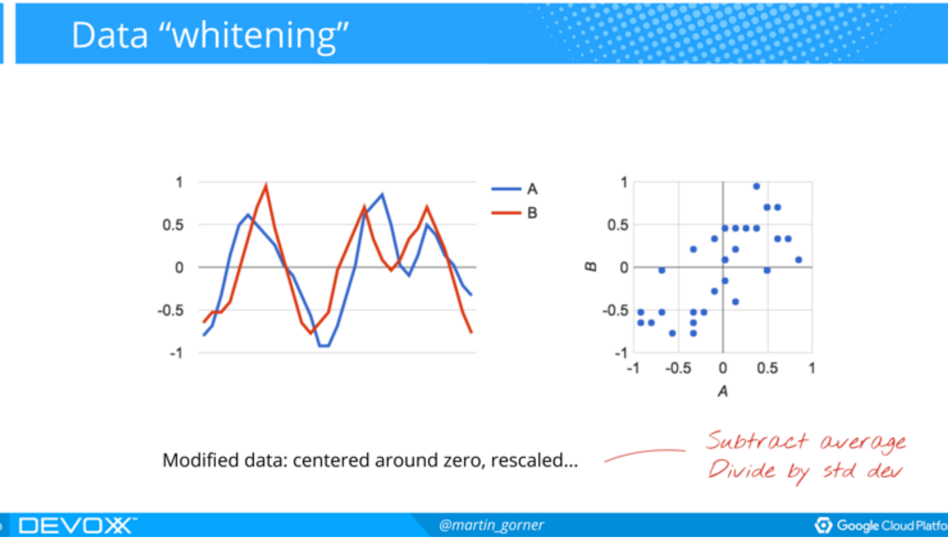


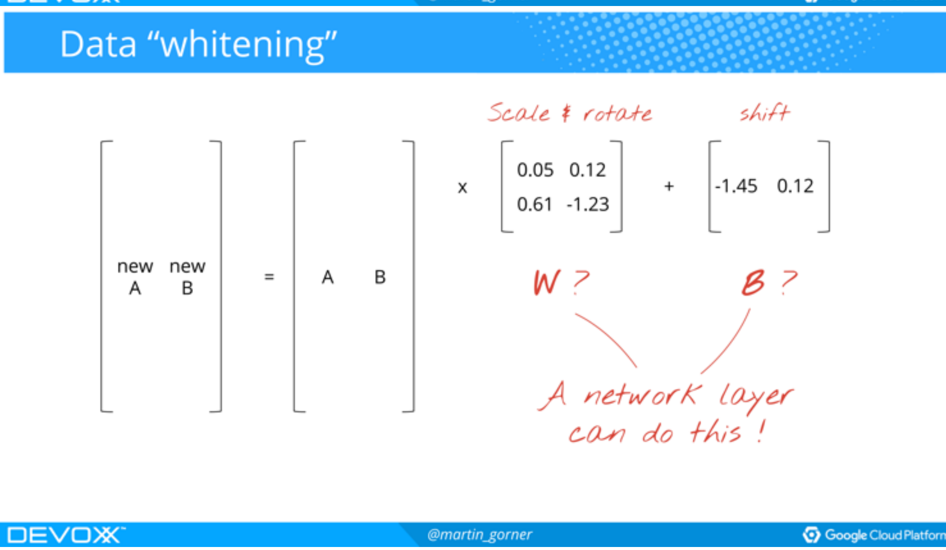
AncoraSIR.com

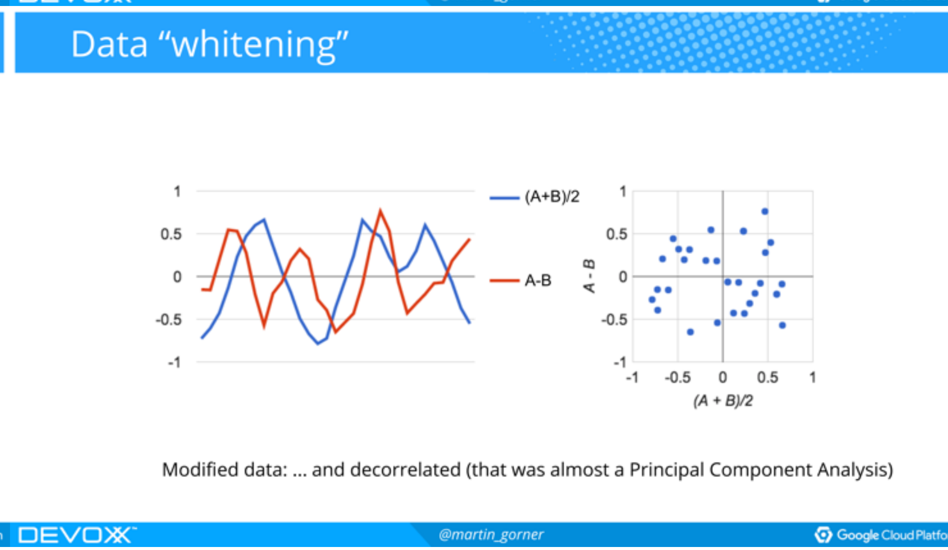
Southern University of Science and Technology

Data Preprocessing





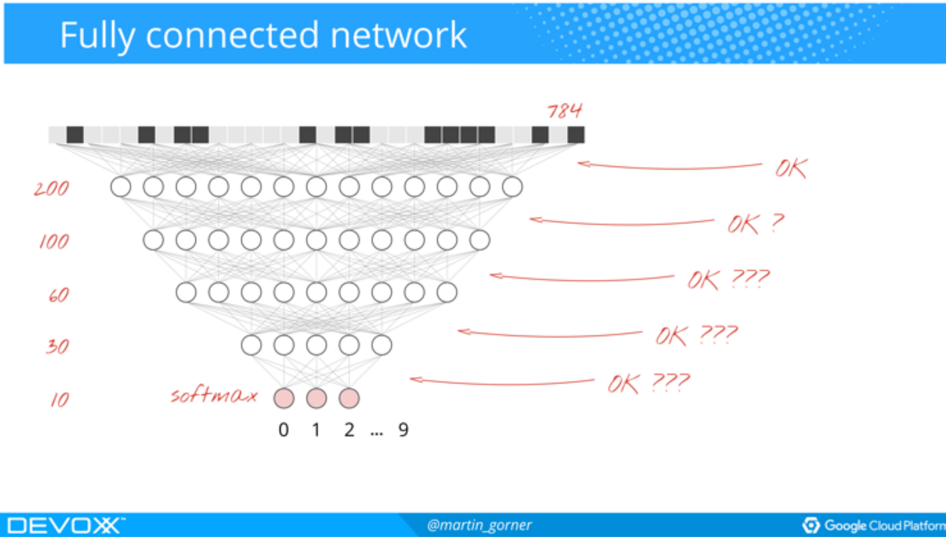


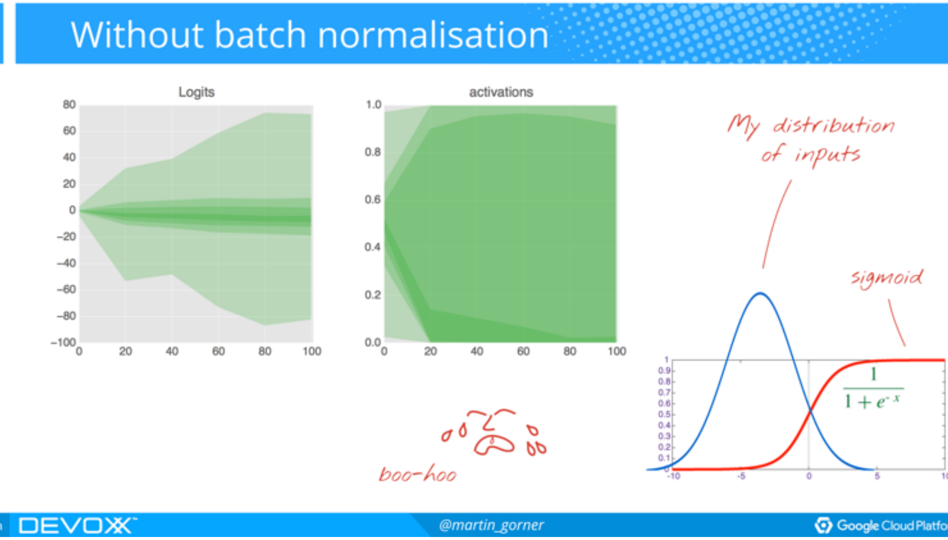


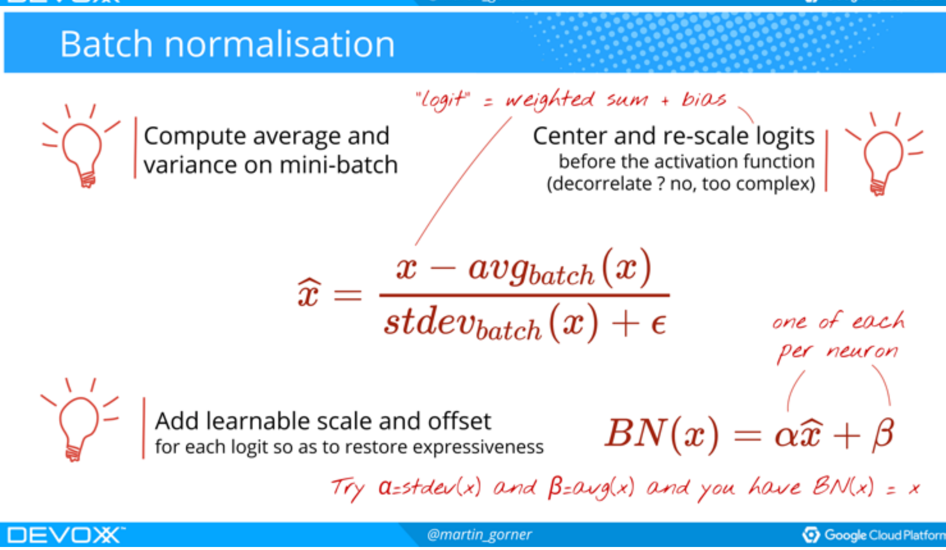
AncoraSIR.com

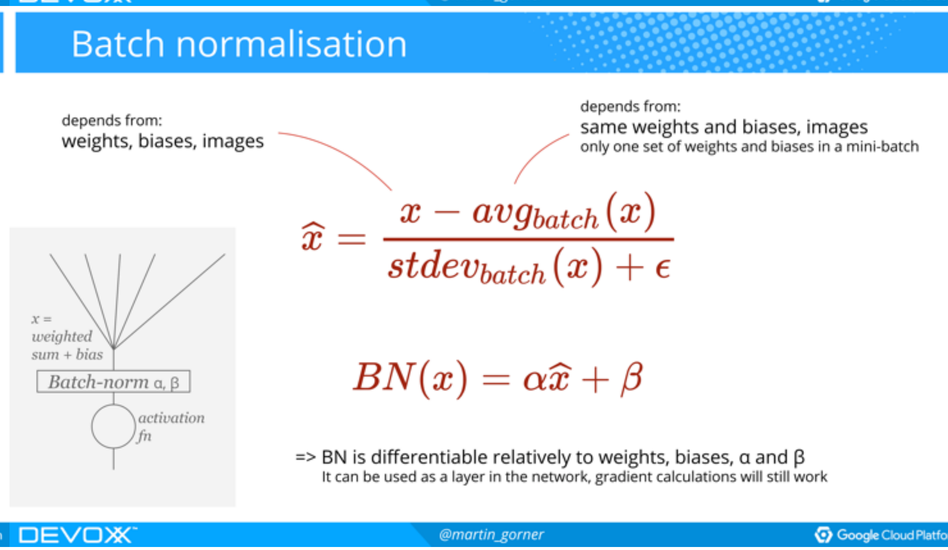
Southern University of Science and Technology

Batch Normalization Theory







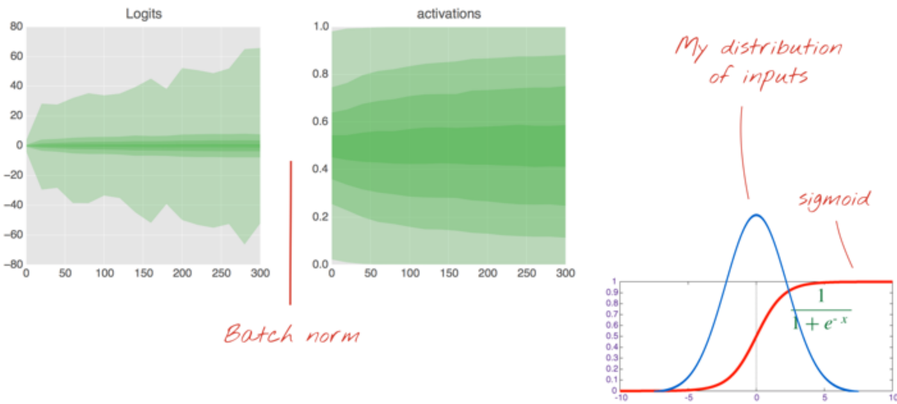


AncoraSIR.com

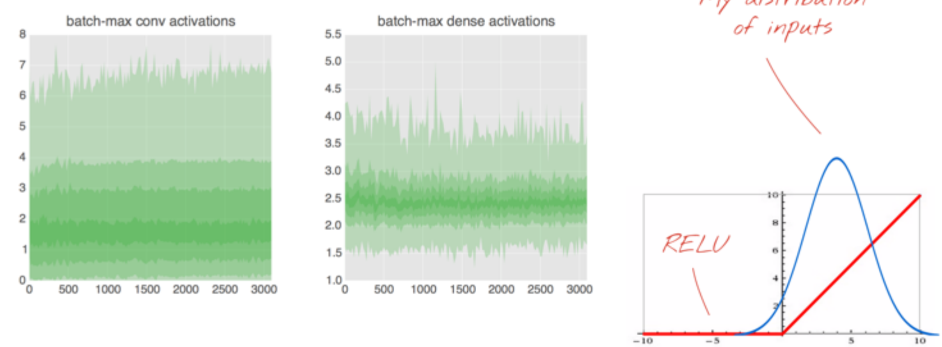
Southern University of Science and Technology

Batch Normalization Done Right

With batch normalisation (sigmoid)



With batch normalisation (RELU)

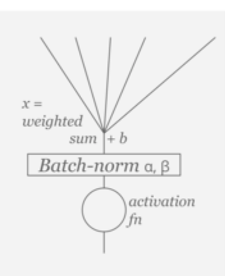


DEVOXX @martin_gorner Google Cloud Platform

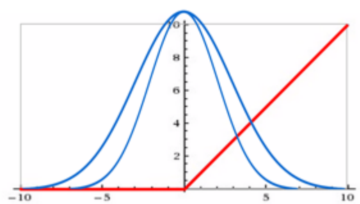
DEVOXX @martin_gorner Google Cloud Platform

Batch normalisation done right

biases :
no longer useful



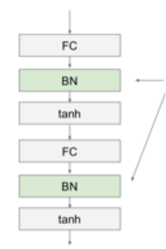
Per neuron:	relu	sigmoid
without BN	**bias**	**bias**
With BN	**β**	**α, β**



when activation fn is RELU
α is not useful
It does not modify output distrib.

+You can go faster: use higher learning rate
+BN also regularises: lower or remove dropout

DEVOXX @martin_gorner Google Cloud Platform



Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

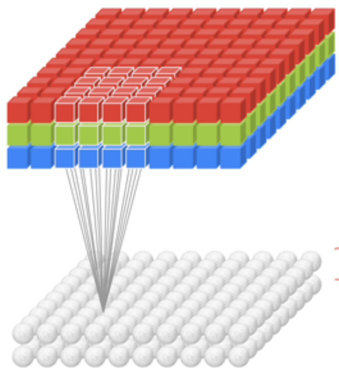
- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe

SUSTech
Southern University of Science and Technology

AncoraSIR.com

Convolutional Batch Normalization

Convolutional batch normalisation



Each neuron or patch has a value:
- per image in the batch
- per x position
- per y position

=> compute avg and stdev across all
`batchsize x width x height` values

$W_1[4, 4, 3]$ — b_1 α_1 β_1
$W_2[4, 4, 3]$ — b_2 α_2 β_2

Still, one bias, scale or offset per neuron

DEVOXX @martin_gorner Google Cloud Platform

Batch normalisation at test time

$$\widehat{x} = \frac{x - avg_?(x)}{stdev_?(x) + \epsilon}$$

Stats on what ?
- Last batch: no
- all images: yes (but not practical)
- => Exponential moving average during training

DEVOXX @martin_gorner Google Cloud Platform

Batch normalisation with Tensorflow

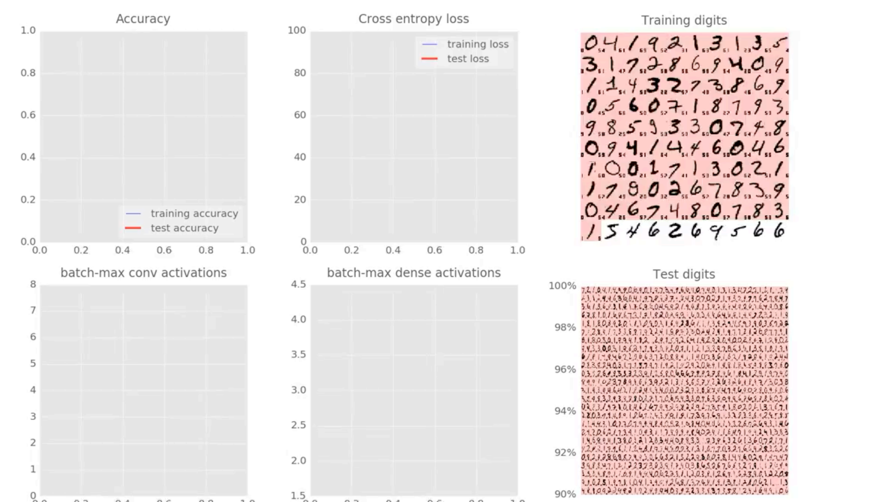
Define one offset and/or scale per neuron

```
def batchnorm_layer(Ylogits, is_test, Offset, Scale, iteration, convolutional=False):
    exp_moving_avg = tf.train.ExponentialMovingAverage(0.9999, iteration)
    if convolutional:    # avg across batch, width, height
        mean, variance = tf.nn.moments(Ylogits, [0, 1, 2])
    else:
        mean, variance = tf.nn.moments(Ylogits, [0])
    update_moving_averages = exp_moving_avg.apply([mean, variance])
    m = tf.cond(is_test, lambda: exp_moving_avg.average(mean), lambda: mean)
    v = tf.cond(is_test, lambda: exp_moving_avg.average(variance), lambda: variance)
    Ybn = tf.nn.batch_normalization(Ylogits, m, v, Offset, Scale, variance_epsilon=1e-5)
    return Ybn, update_moving_averages — don't forget to execute this (sess.run)
```

apply activation fn on Ybn

The code is on GitHub: goo.gl/DEOe7Z

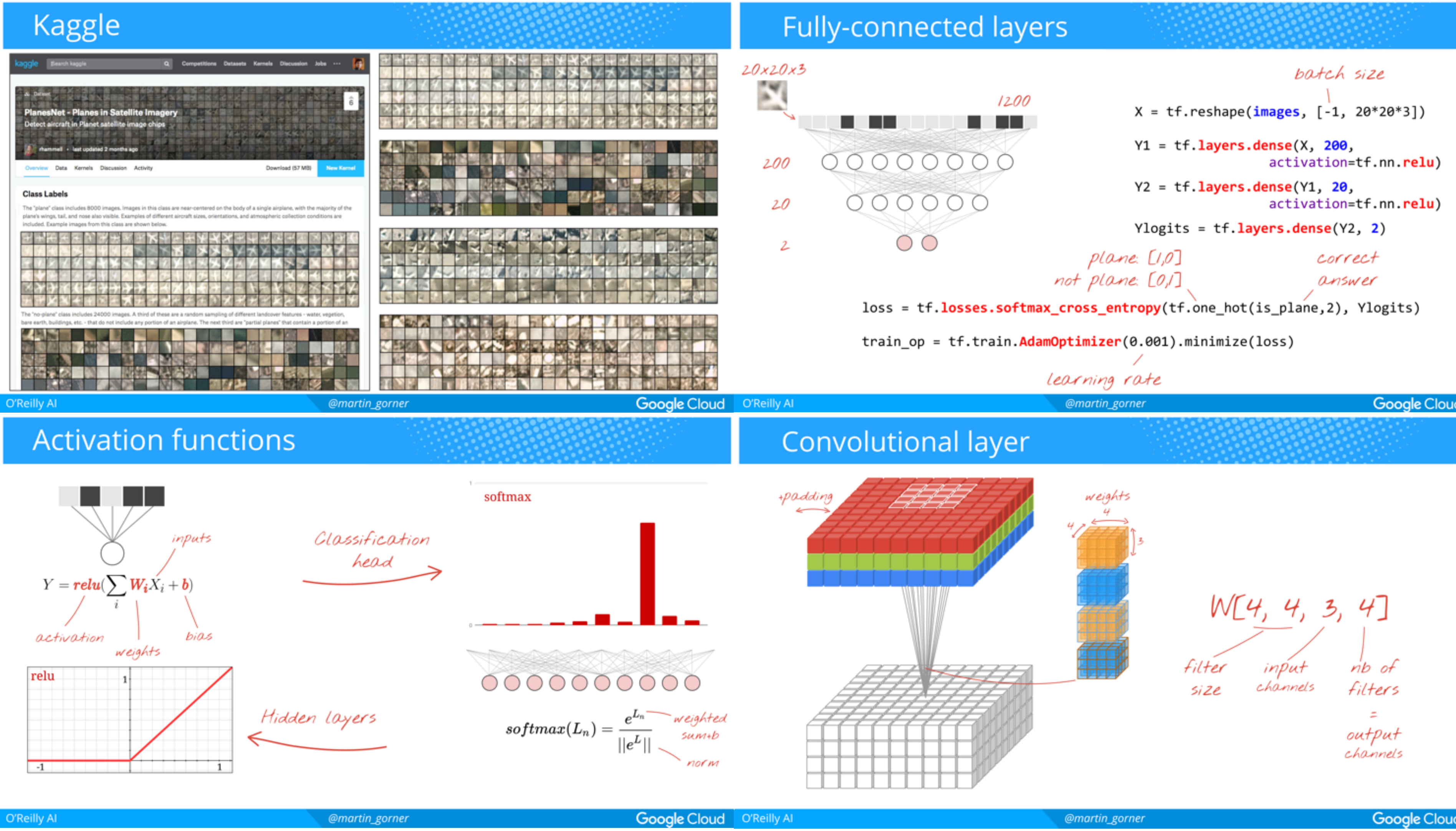
DEVOXX @martin_gorner Google Cloud Platform



AncoraSIR.com

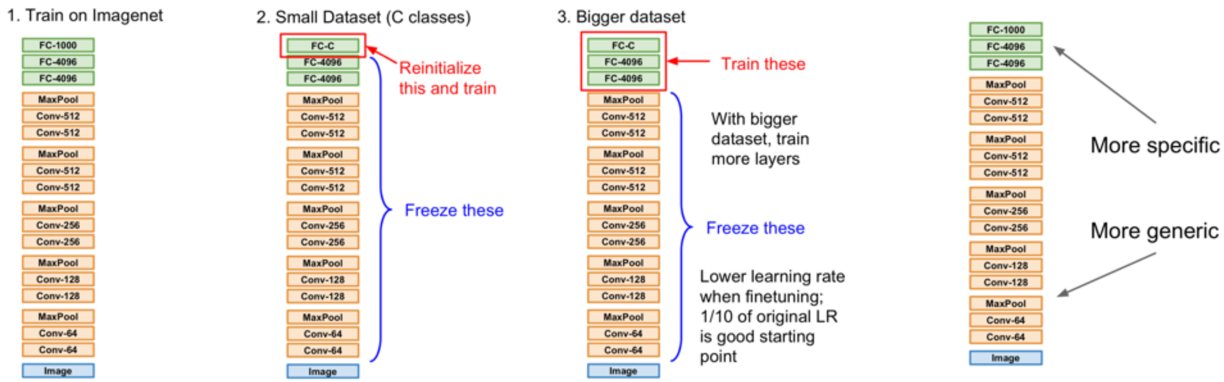
Southern University of Science and Technology

Example: Airplane or not?



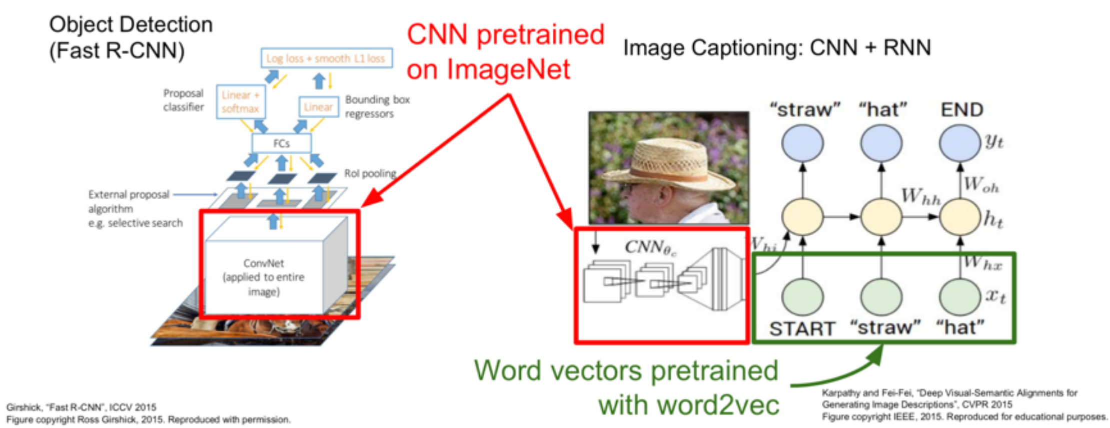
Transfer Learning

You need a lot of data if you want to train/use CNNs, really?



	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Transfer learning with CNNs is pervasive…
(it's the norm, not an exception)



Next class:

- Transfer learning for arcade claw machine robot

AncoraSIR.com

Thank you!

Prof. Song Chaoyang

- Dr. Wan Fang (sophie.fwan@hotmail.com)

AncoraSIR.com