Deep Learning for Vision & Language

Computer Vision II: Convolutional Neural Network Architectures

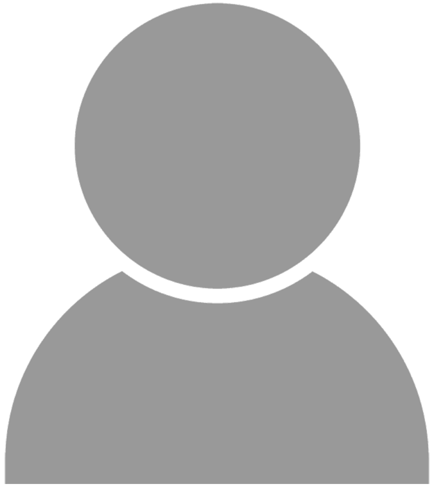
RICE UNIVERSITY

About the class

- COMP 646: Deep Learning for Vision and Language
- Instructor: **Vicente** Ordóñez (Vicente Ordóñez Román)
- Website: https://www.cs.rice.edu/~vo9/deep-vislang
- Location: Zoom – Rice Canvas has the links **OR**
  Duncan Hall 1070
- Times: Mondays, Wednesdays, and Fridays
  from 1pm to 1:50pm Central Time
- Office Hours: Fridays 2 to 3pm
- Teaching Assistants: Brian Hoepfl, Liuba Orlov Savko
- Discussion Forum: Rice Canvas

TAs and Office Hours

Brian Hopfl
Wednesdays 2:30pm to 4:30pm (today)
Next week Location: McMurtry commons
Contact email: beh3@rice.edu



Liuba Orlov Savko
Fridays 4pm to 5pm
Location: Sid's Place (2nd Floor Duncan near fridge)
Contact email: lo13@rice.edu

ILSVRC:
Imagenet Large Scale Visual Recognition Challenge
[Russakovsky et al 2014]

The Problem: Classification

Classify an image into 1000 possible classes:
e.g. Abyssinian cat, Bulldog, French Terrier, Cormorant, Chickadee, red fox, banjo, barbell, hourglass, knot, maze, viaduct, etc.

→

cat, tabby cat (0.71)
Egyptian cat (0.22)
red fox (0.11)
.....

The Data: ILSVRC

Imagenet Large Scale Visual Recognition Challenge (ILSVRC): Annual Competition

1000 Categories

~1000 training images per Category

~1 million images in total for training

~50k images for validation

Only images released for the test set but no annotations,
evaluation is performed centrally by the organizers (max 2 per week)

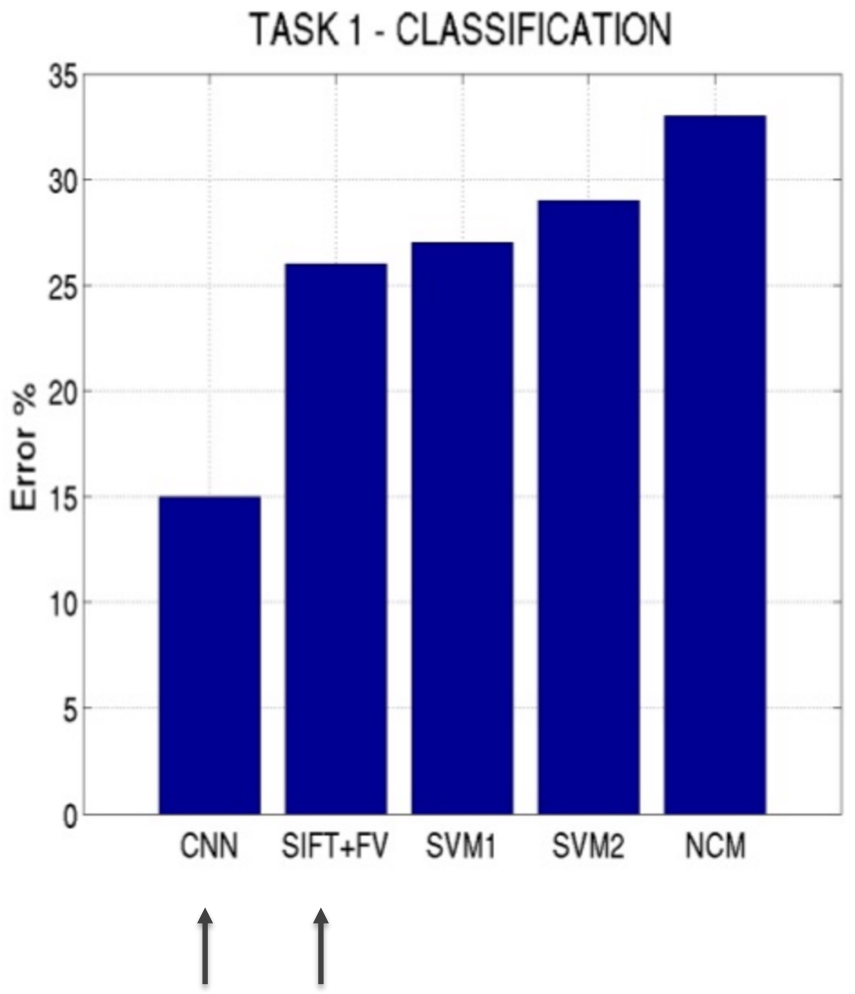
The Evaluation Metric: Top K-error

True label: Abyssinian cat

Top-1 error: 1.0	Top-1 accuracy: 0.0
Top-2 error: 1.0	Top-2 accuracy: 0.0
Top-3 error: 1.0	Top-3 accuracy: 0.0
Top-4 error: 0.0	Top-4 accuracy: 1.0
Top-5 error: 0.0	Top-5 accuracy: 1.0

cat, tabby cat (0.61)
Egyptian cat (0.22)
red fox (0.11)
Abyssinian cat (0.10)
French terrier (0.03)
.....

Top-5 error on this competition (2012)



Alexnet (Krizhevsky et al NIPS 2012)

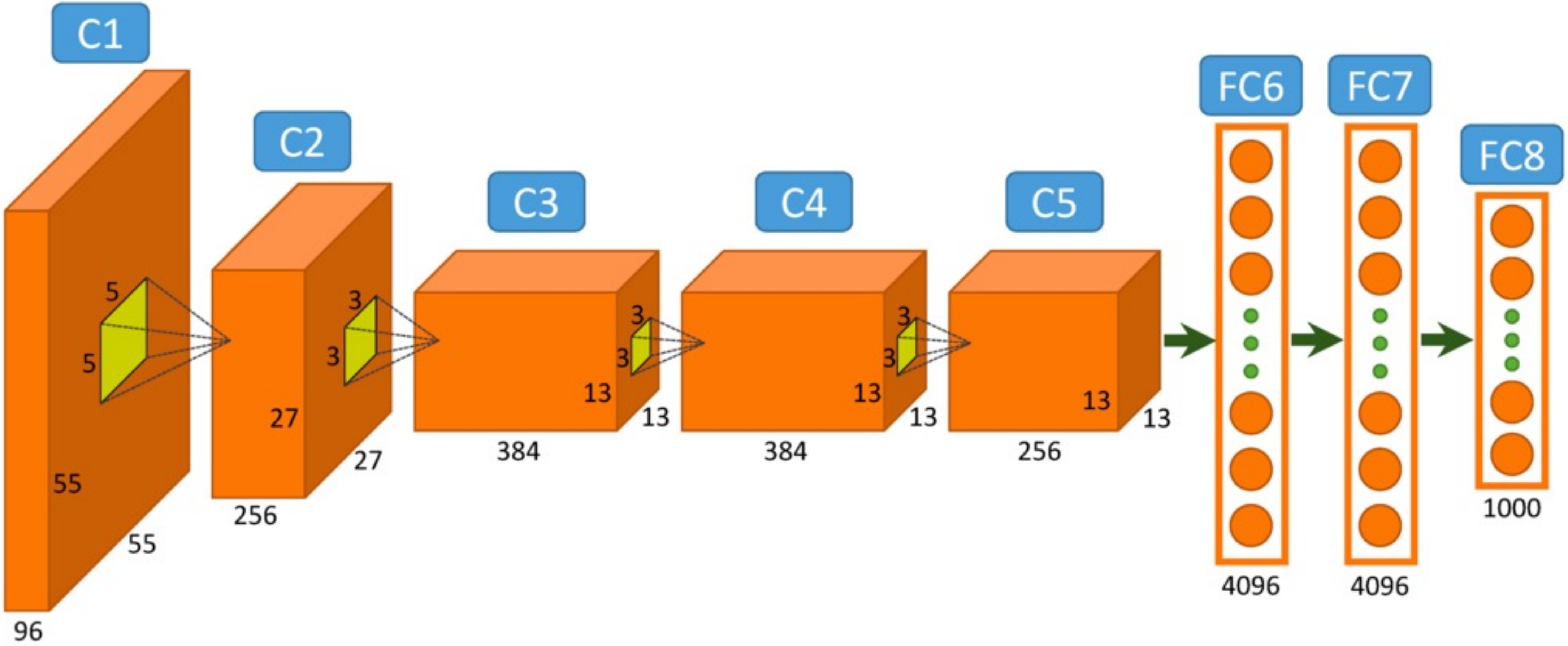
ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Alexnet



https://www.saagie.com/fr/blog/object-detection-part1

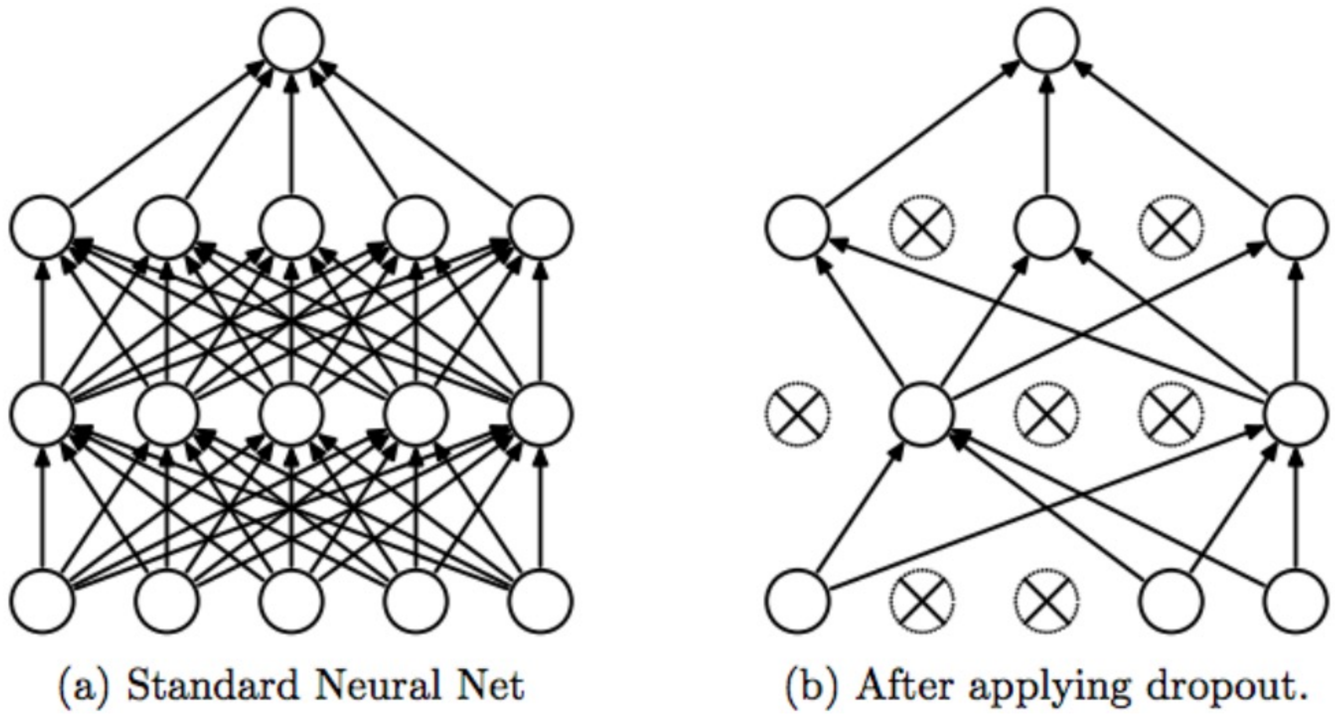
Pytorch Code for Alexnet

- In-class analysis

  https://github.com/pytorch/vision/blob/master/torchvision/models/alexnet.py

Dropout Layer

Happens for every batch for a different set of connections
only during training



Srivastava et al 2014

Important

model.train()

model.eval()

Preprocessing and Data Augmentation

Preprocessing and Data Augmentation

256

256

Preprocessing and Data Augmentation

224x224

Preprocessing and Data Augmentation

224x224





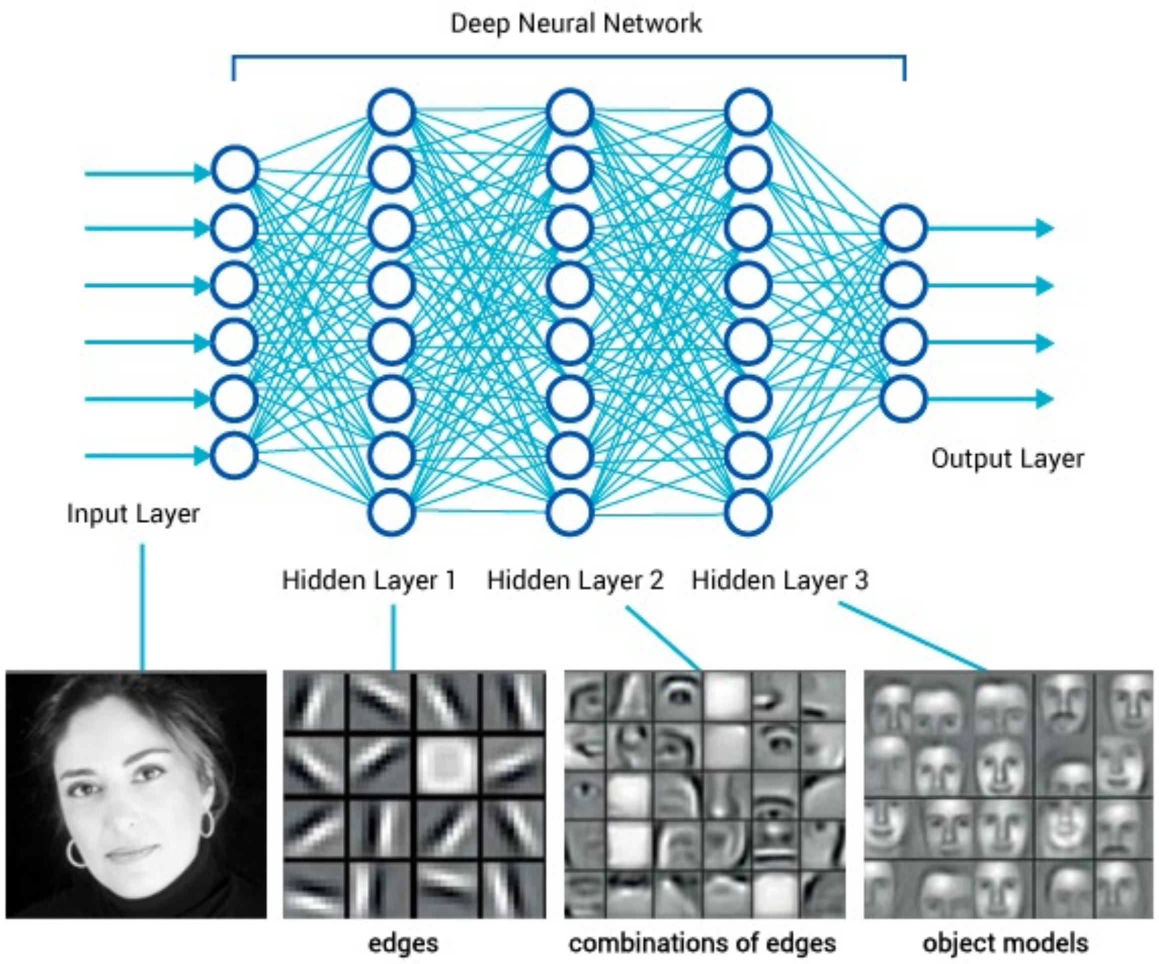
True label: Abyssinian cat

Some Important Aspects

- Using ReLUs instead of Sigmoid or Tanh
- Momentum + Weight Decay
- Dropout (Randomly sets Unit outputs to zero during training)
- GPU Computation!

Model	Top-1	Top-5
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**

What is happening?



https://www.saagie.com/fr/blog/object-detection-part1

SIFT + FV + SVM (or softmax)

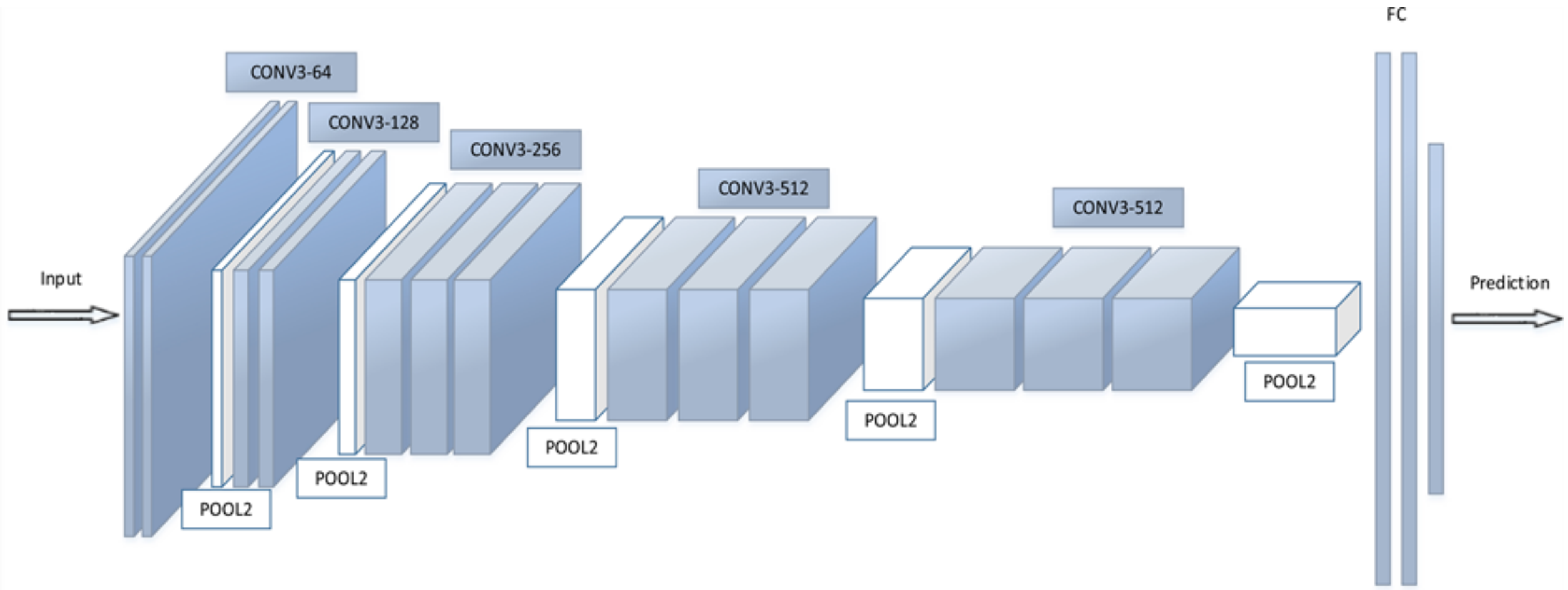
→ Feature extraction (SIFT) → Feature encoding (Fisher vectors) → Classification (SVM or softmax) →

Deep Learning

→ Convolutional Network (includes both feature extraction and classifier) →

VGG Network

Top-5:

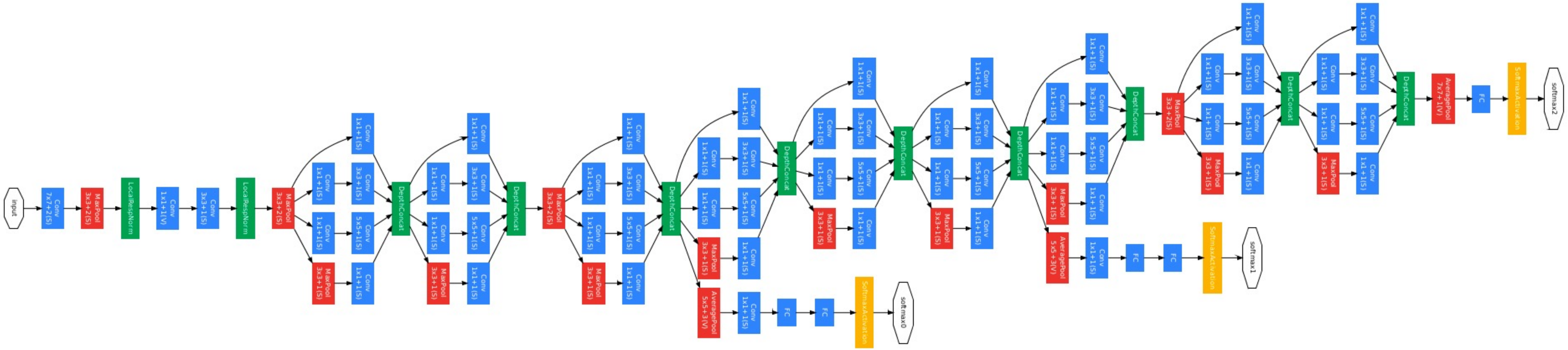


https://github.com/pytorch/vision/blob/master/torchvision/models/vgg.py

Simonyan and Zisserman, 2014.

https://arxiv.org/pdf/1409.1556.pdf

GoogLeNet

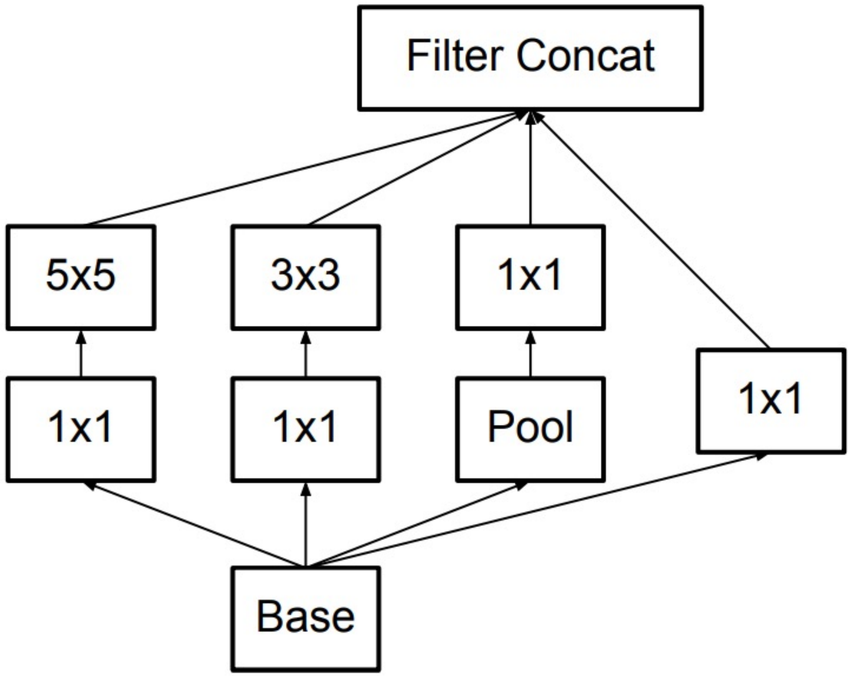


https://github.com/kuangliu/pytorch-cifar/blob/master/models/googlenet.py

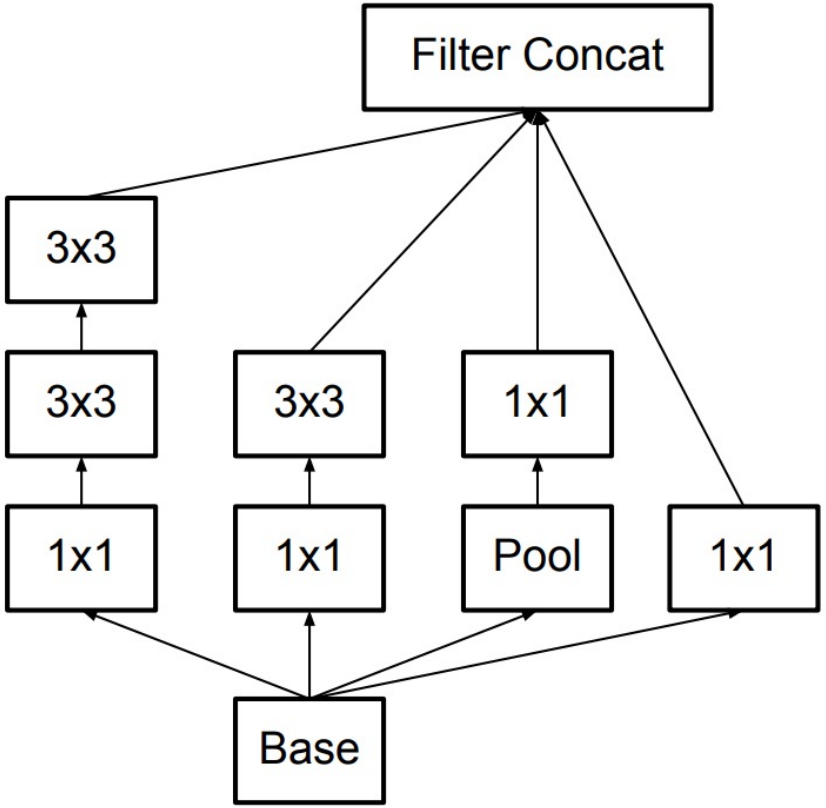
Szegedy et al. 2014

https://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf

Further Refinements – Inception v3, e.g.



GoogLeNet (Inceptionv1)



Inception v3

BatchNormalization Layer

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

https://arxiv.org/abs/1502.03167

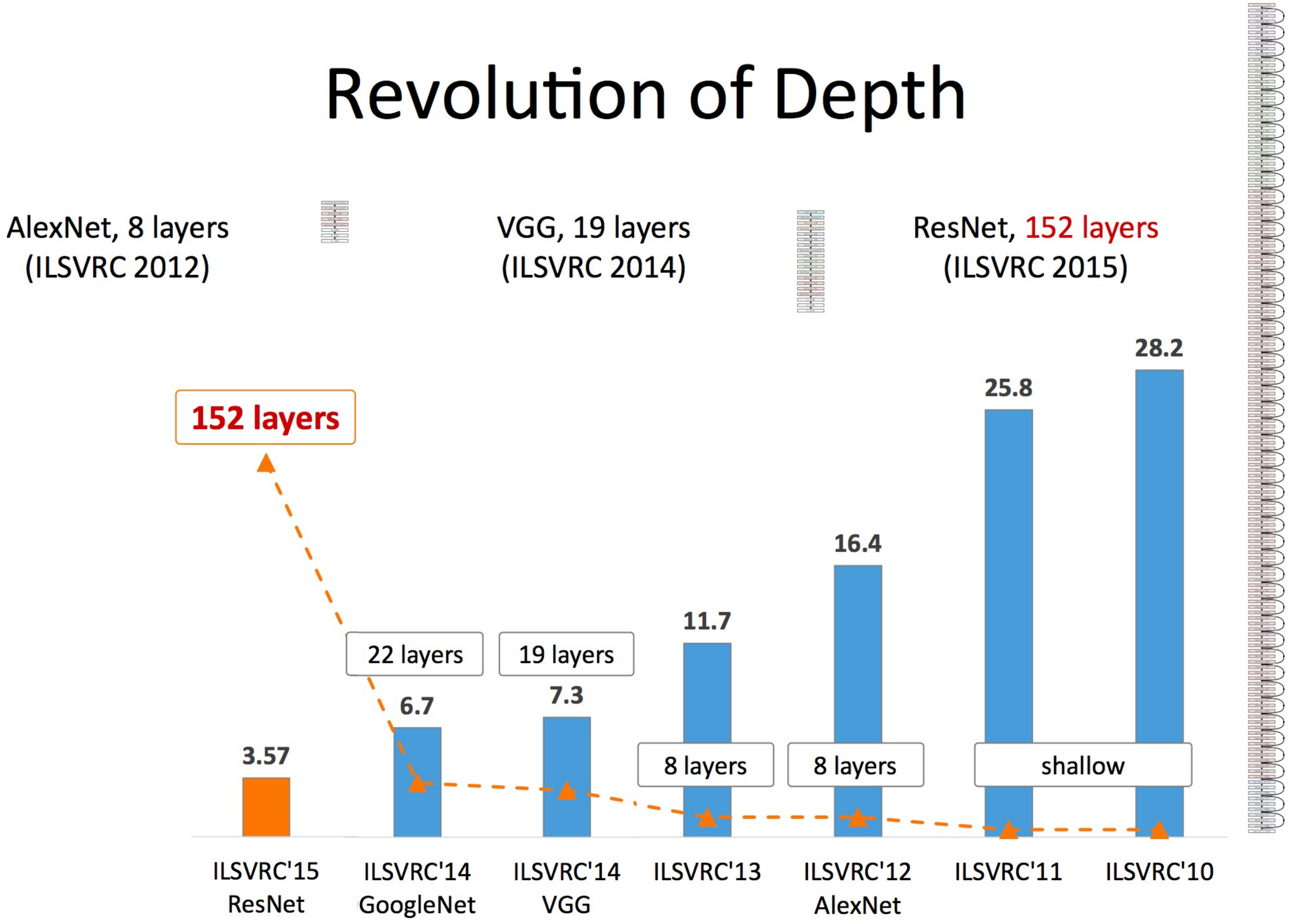
ResNet (He et al CVPR 2016)

Sorry, does not fit in slide.

http://felixlaumon.github.io/assets/kaggle-right-whale/resnet.png

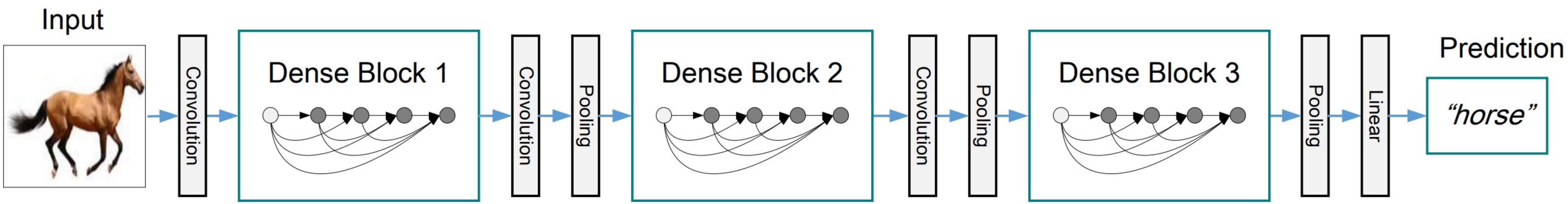
https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py

Revolution of Depth

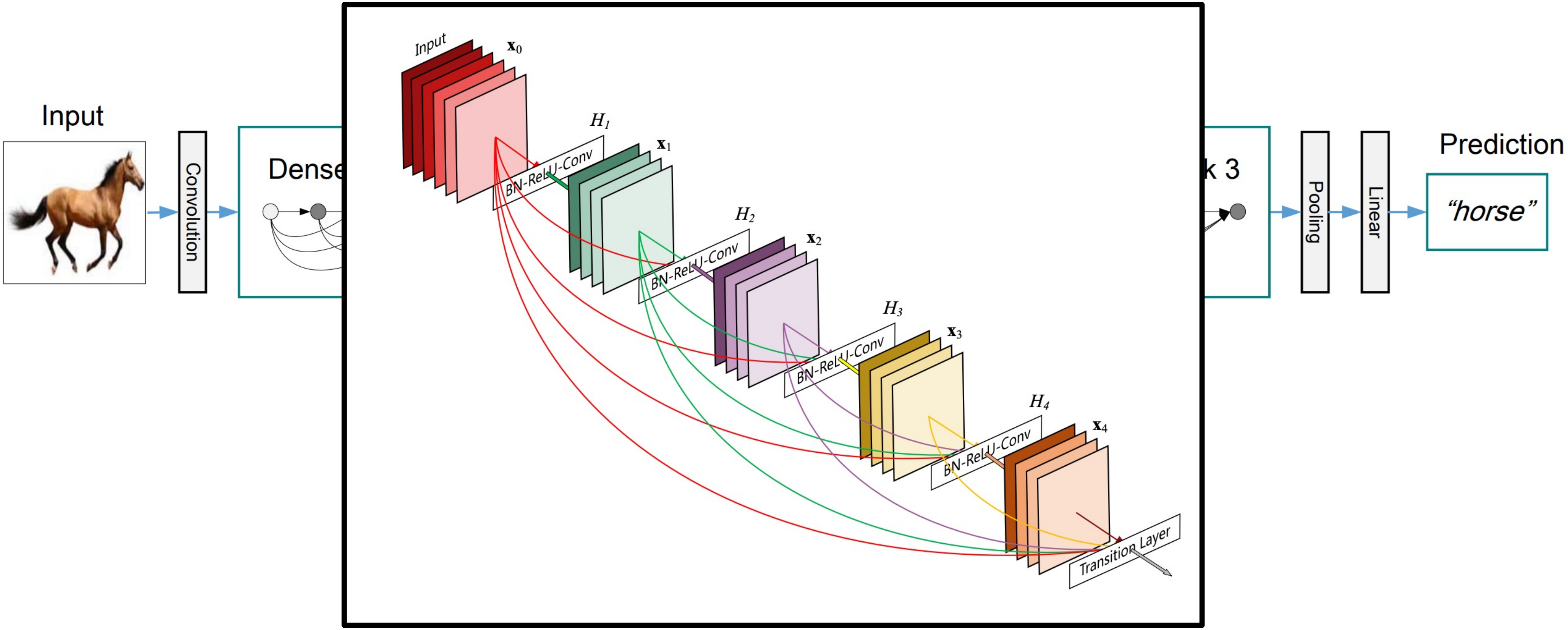


Slide by Mohammad Rastegari

Densenet

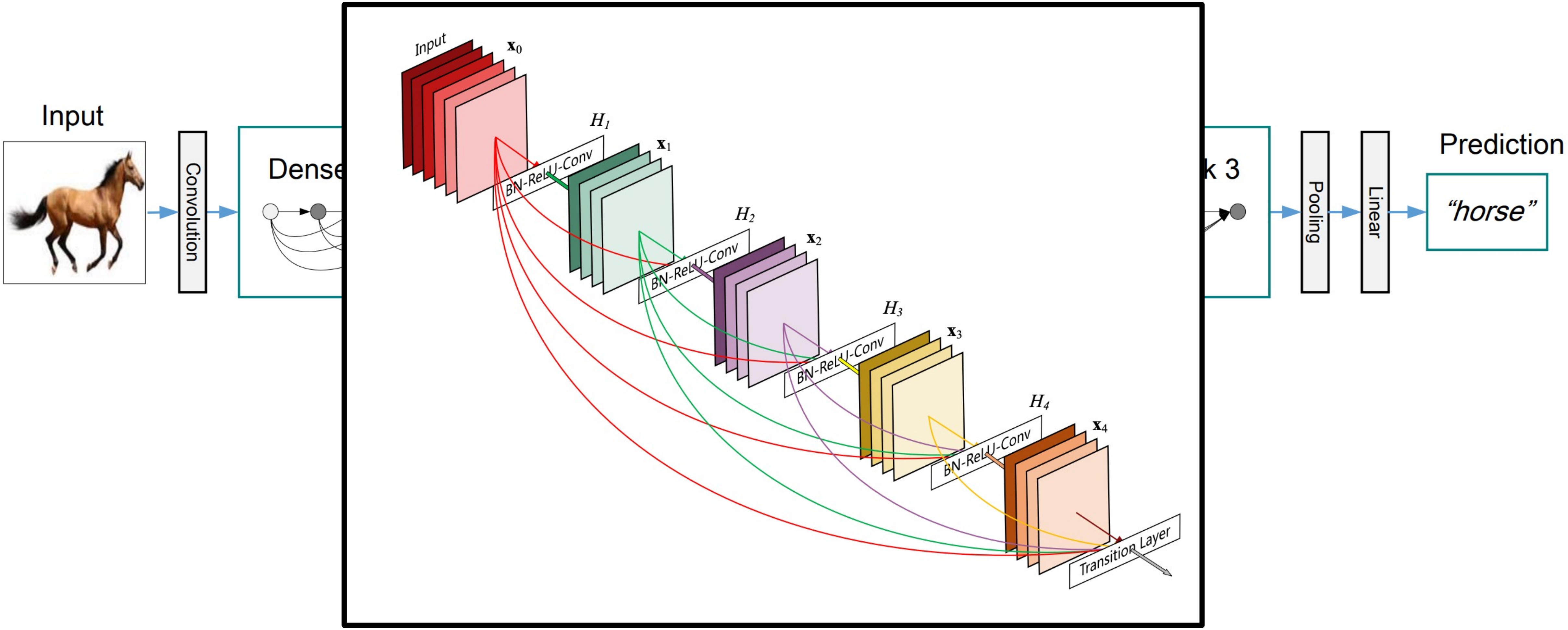

Input
Convolution
Dense Block 1
Convolution
Pooling
Dense Block 2
Convolution
Pooling
Dense Block 3
Pooling
Linear
Prediction
"horse"

Densenet



https://arxiv.org/pdf/1608.06993.pdf

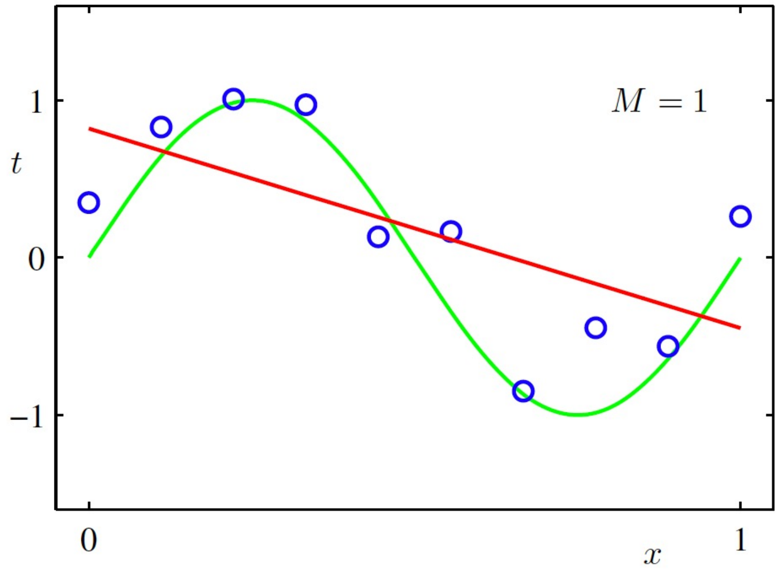
Densenet



https://arxiv.org/pdf/1608.06993.pdf

Pending Topic: Regularization

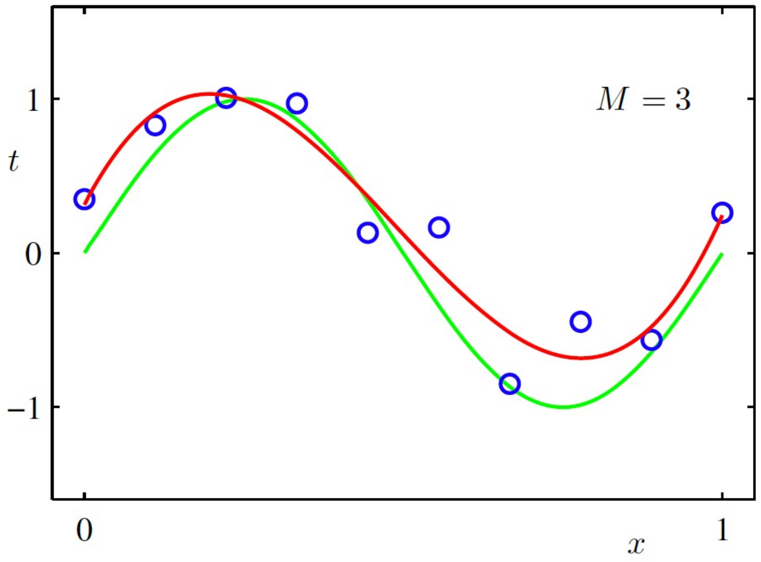
f is linear



$Loss(w)$ is high

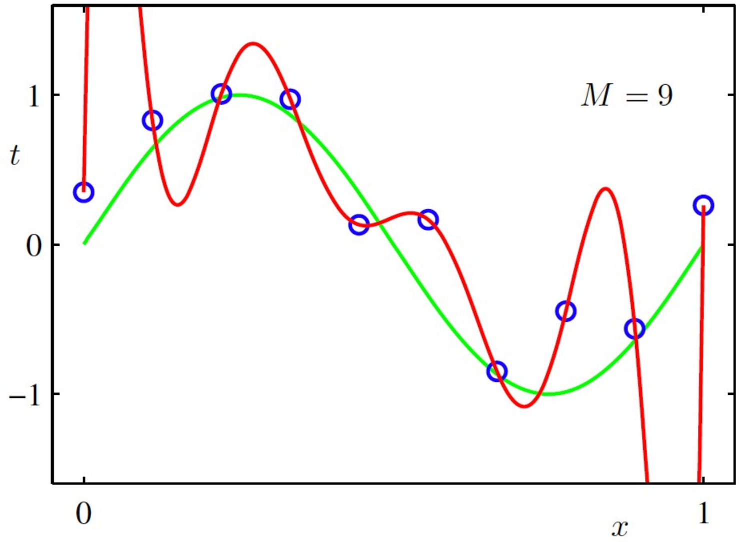
Underfitting

f is cubic



$Loss(w)$ is low

f is a polynomial of degree 9



$Loss(w)$ is zero!

Overfitting

(mini-batch) Stochastic Gradient Descent (SGD)

$\lambda = 0.01$

$$l(w,b) = \sum_{i \in B} Cost(w,b)$$

Initialize w and b randomly

for e = 0, num_epochs **do**

for b = 0, num_batches **do**

Compute: $dl(w,b)/dw$ and $dl(w,b)/db$

Update w: $w = w - \lambda \, dl(w,b)/dw$

Update b: $b = b - \lambda \, dl(w,b)/db$

Print: $l(w,b)$ // Useful to see if this is becoming smaller or not.

end

end

Regularization

- Large weights lead to large variance. i.e. model fits to the training data too strongly.
- Solution: Minimize the loss but also try to keep the weight values small by doing the following:

$$\text{minimize} \qquad L(w, b) + \alpha \sum_i |w_i|^2$$

Regularization

- Large weights lead to large variance. i.e. model fits to the training data too strongly.

- Solution: Minimize the loss but also try to keep the weight values small by doing the following:

minimize $$L(w,b) + \alpha \sum_i |w_i|^2$$

Regularizer term
e.g. L2- regularizer

SGD with Regularization (L-2)

$\lambda = 0.01$

$l(w, b) = l(w, b) + \alpha \sum_i |w_i|^2$

Initialize w and b randomly

for e = 0, num_epochs **do**

for b = 0, num_batches **do**

Compute: $dl(w, b)/dw$ and $dl(w, b)/db$

Update w: $w = w - \lambda \, dl(w, b)/dw \boxed{- \lambda \alpha w}$

Update b: $b = b - \lambda \, dl(w, b)/db \boxed{- \lambda \alpha w}$

Print: $l(w, b)$ // Useful to see if this is becoming smaller or not.

end

end

Revisiting Another Problem with SGD

$\lambda = 0.01$

$$l(w,b) = l(w,b) + \alpha \sum_i |w_i|^2$$

Initialize w and b randomly

for e = 0, num_epochs **do**

for b = 0, num_batches **do**

Compute: $dl(w,b)/dw$ and $dl(w,b)/db$

Update w: $w = w - \lambda\, dl(w,b)/dw - \lambda \alpha w$

Update b: $b = b - \lambda\, dl(w,b)/db - \lambda \alpha w$

Print: $l(w,b)$ // Useful to see if this is becoming smaller or not.

end

end

These are only approximations to the true gradient with respect to $L(w,b)$

Revisiting Another Problem with SGD

$\lambda = 0.01$

$$l(w,b) = l(w,b) + \alpha \sum_i |w_i|^2$$

Initialize w and b randomly

for e = 0, num_epochs **do**

for b = 0, num_batches **do**

Compute: $dl(w,b)/dw$ and $dl(w,b)/db$

Update w: $w = w - \lambda\, dl(w,b)/dw - \lambda \alpha w$

Update b: $b = b - \lambda\, dl(w,b)/db - \lambda \alpha w$

Print: $l(w,b)$ // Useful to see if this is becoming smaller or not.

end

end

This could lead to "un-learning" what has been learned in some previous steps of training.

Solution: Momentum Updates

$\lambda = 0.01$

$l(w, b) = l(w, b) + \alpha \sum_i |w_i|^2$

Initialize w and b randomly

for e = 0, num_epochs **do**

for b = 0, num_batches **do**

Compute: $dl(w, b)/dw$ and $dl(w, b)/db$

Update w: $w = w - \lambda\, dl(w, b)/dw - \lambda \alpha w$

Update b: $b = b - \lambda\, dl(w, b)/db - \lambda \alpha w$

Print: $l(w, b)$ // Useful to see if this is becoming smaller or not.

end

end

Keep track of previous gradients in an accumulator variable! and use a weighted average with current gradient.

Solution: Momentum Updates

$\lambda = 0.01 \qquad \tau = 0.9$

$$l(w, b) = l(w, b) + \alpha \sum_i |w_i|^2$$

Initialize w and b randomly

global v

for e = 0, num_epochs **do**

for b = 0, num_batches **do**

Compute: $dl(w, b)/dw$

Compute: $v = \tau v + dl(w, b)/dw + \alpha w$

Update w: $w = w - \lambda v$

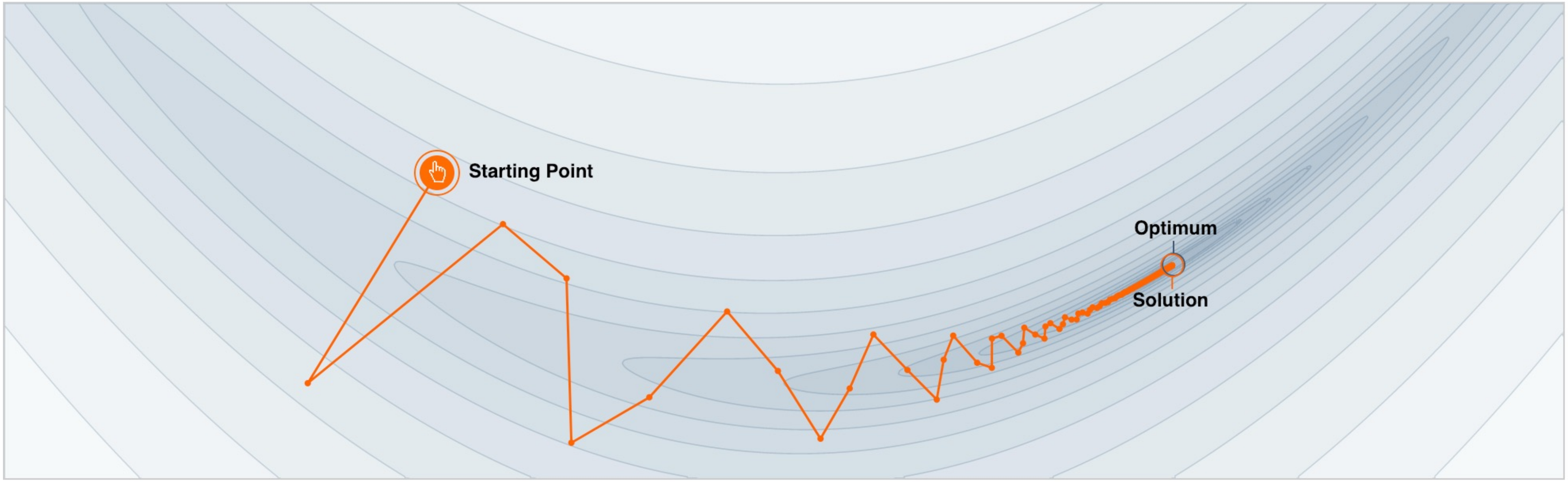
Print: $l(w, b)$ // Useful to see if this is becoming smaller or not.

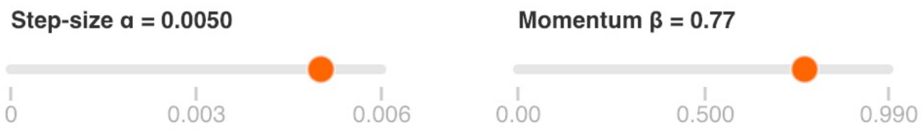
end

end

Keep track of previous gradients in an accumulator variable! and use a weighted average with current gradient.

More on Momentum





We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

https://distill.pub/2017/momentum/

Questions