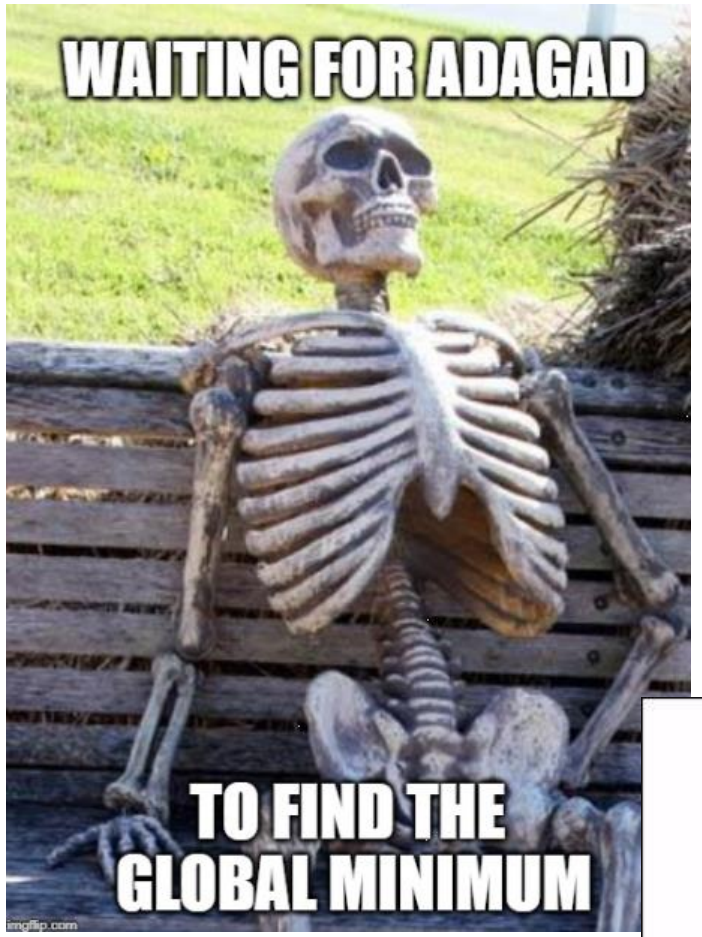
HARDER
BETTER
FASTER
STRONGER

MEMES!

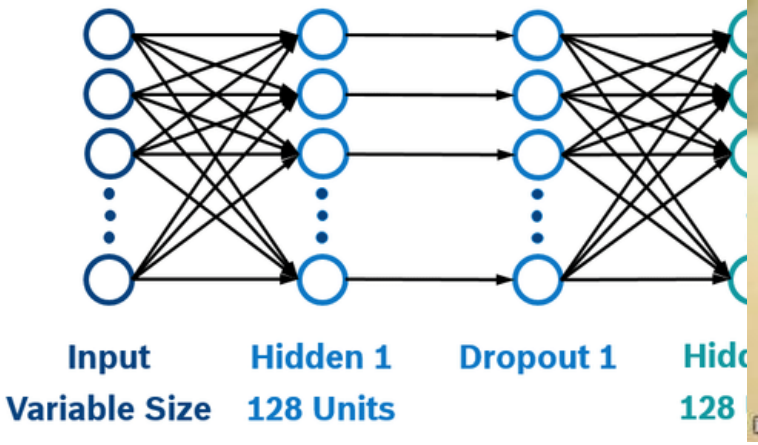

WAITING FOR ADAGAD
TO FIND THE GLOBAL MINIMUM

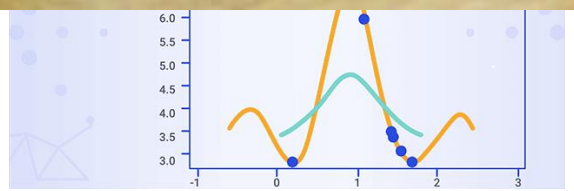

STOCHASTIC GRADIENT DESCENT
SGD WITH MOMENTUM
RMSPROP
ADAM

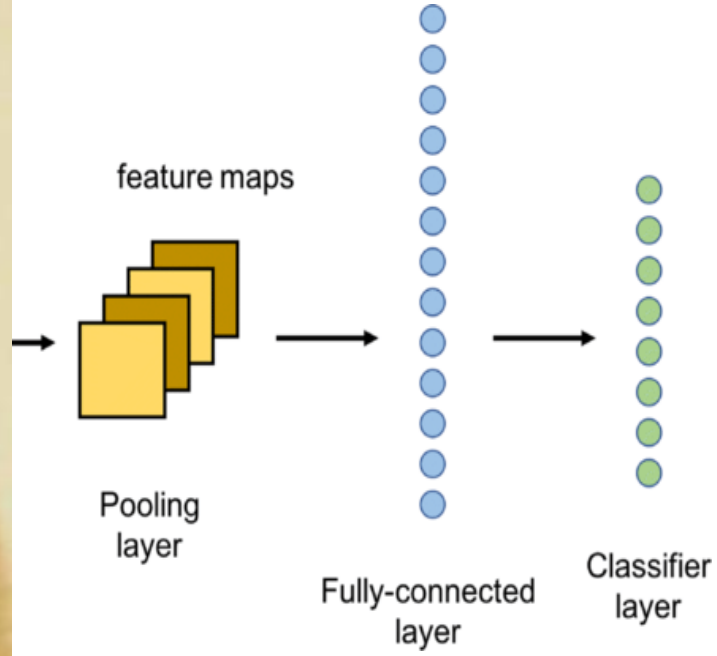

I HAVE MOMENTUM, I HAVE RMSPROP
ADAM
imgflip.com

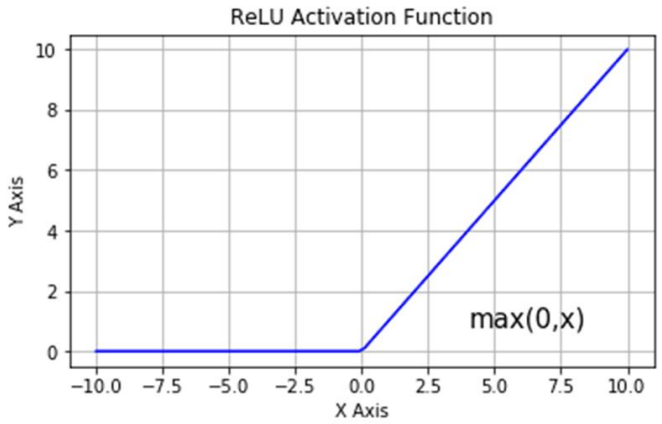
Who is guilty?

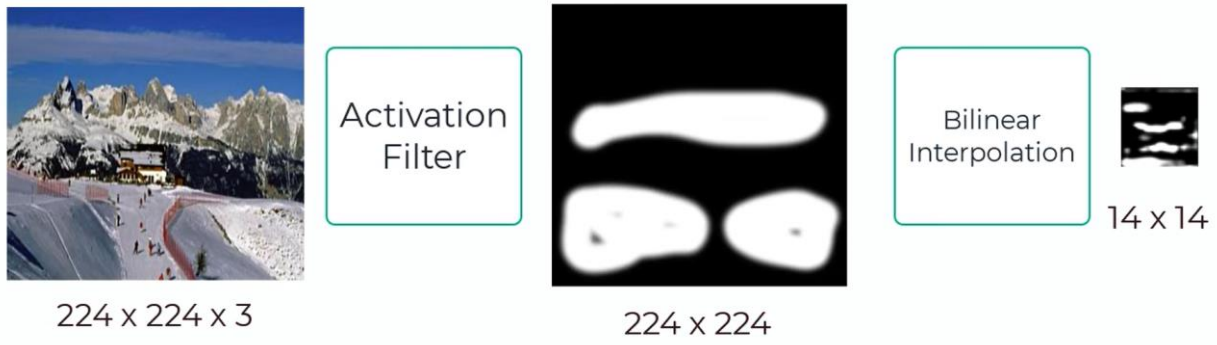
„I know AI"
starter pack





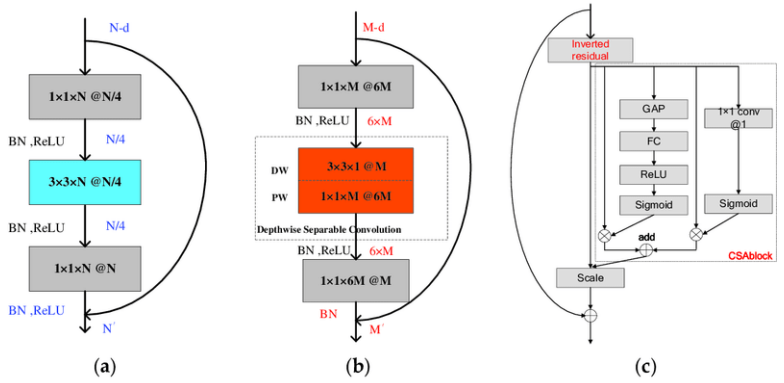


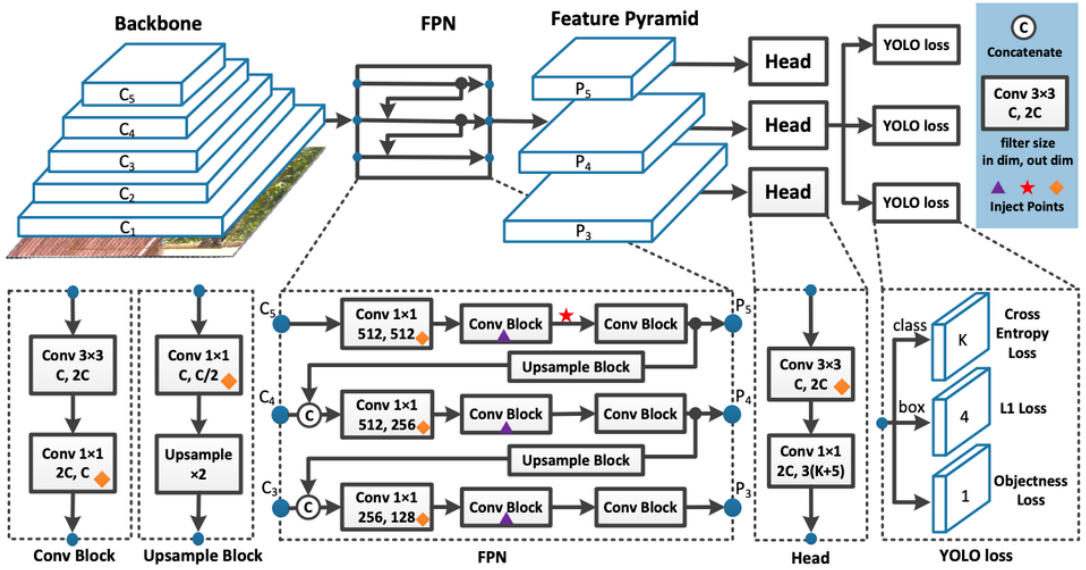




„Yolo v7 is a simple architecture”

Residual blocks are my favourite!

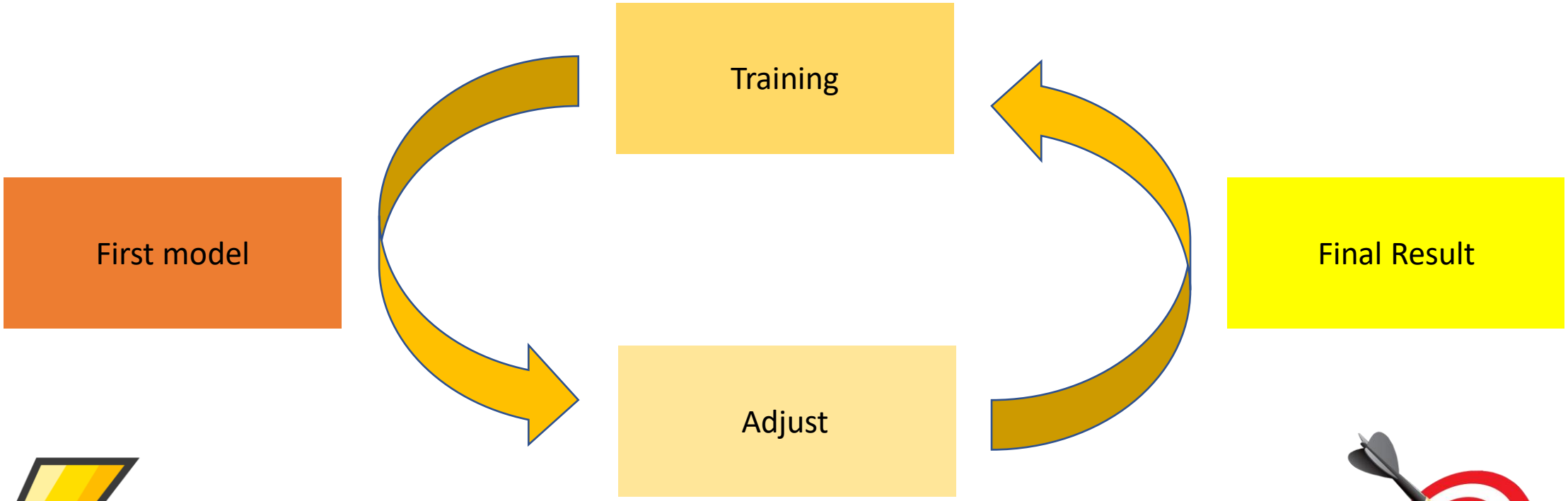




Disclaimer

- While it’s all fun and games, most of the tips come from Work Experience & Research Papers
- Main aim is to close the gap between the „I know AI starter pack” people and recent discoveries in AI modelling, training & optimization
- There will be (non graded) quizes – engage a bit!
- I  CNNs

Modelling & Optimizing Neural Networks



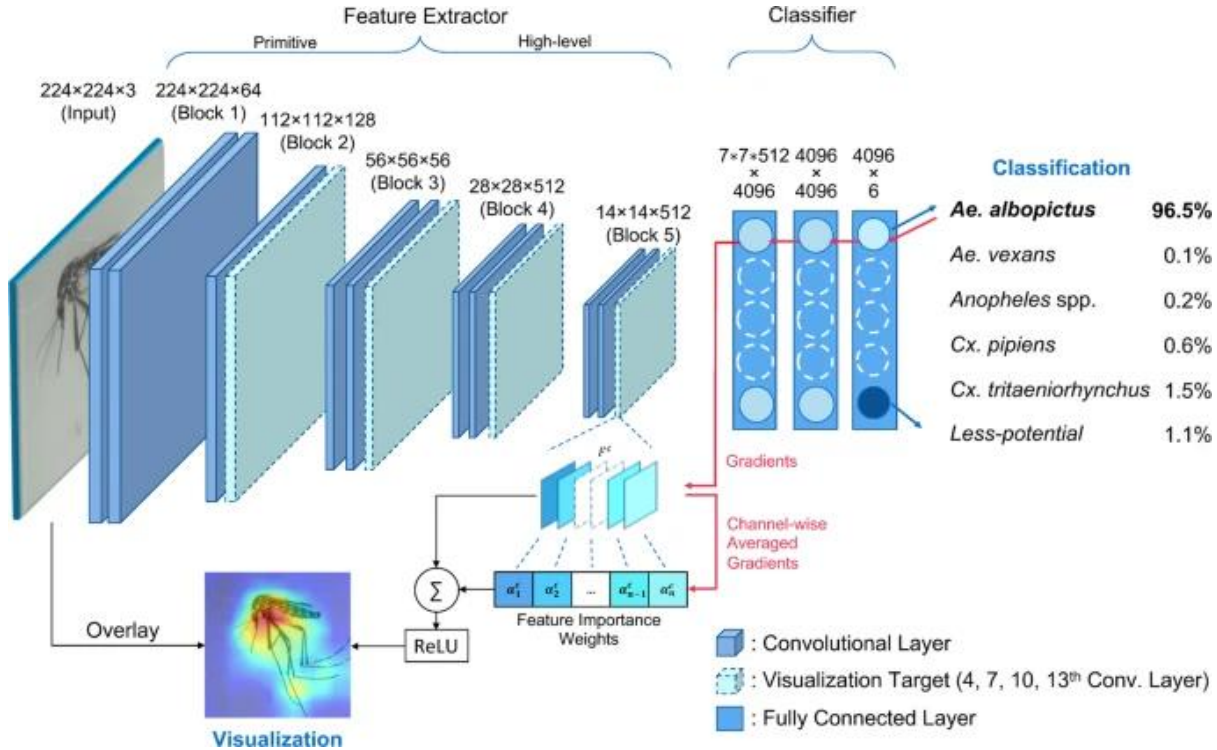
Speed ++

Time Saved
++

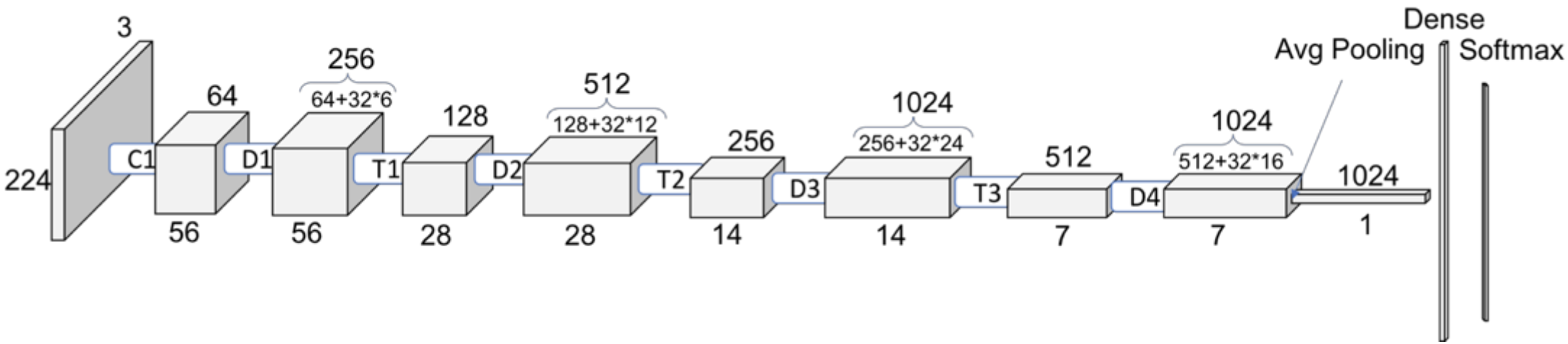
Accuracy ++

#1 Rule of AI Modelling...

Power Of



Model: VGG-16



Model: Dense-121

CPU:

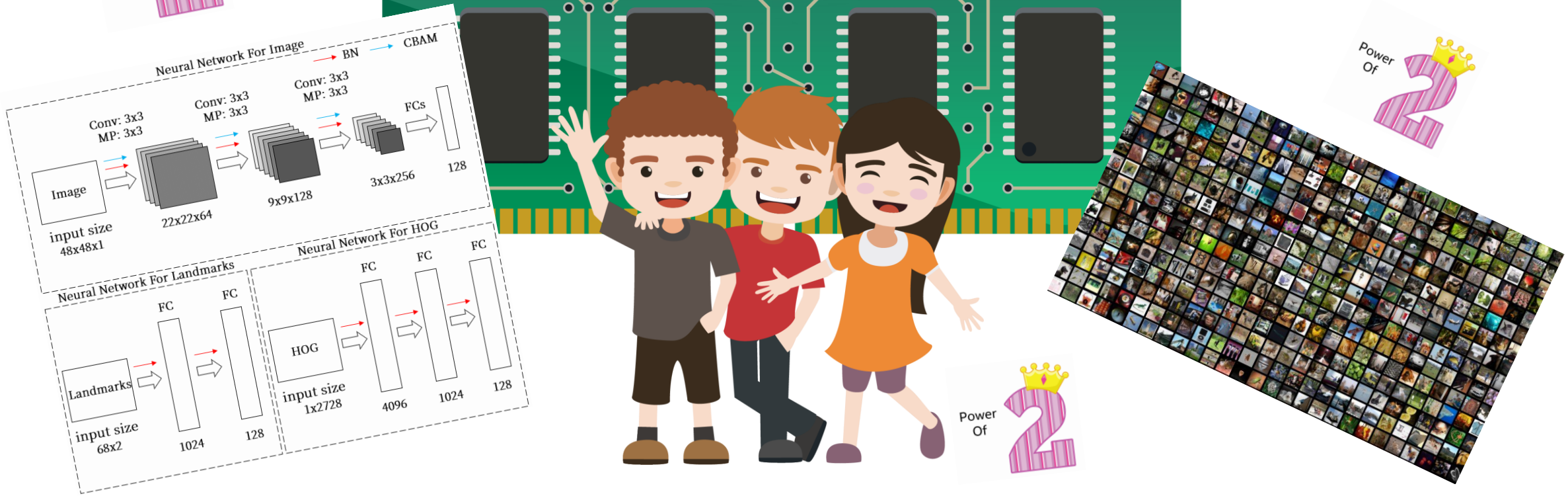
- SIMD (Single Instruction, Multiple Data) are performed in batch sizes, which are powers of 2
- CPU Cores are usually a power of 2 / power of 2 * smth

Name	Codename	Cores
Ryzen 5 3600	Matisse	6 / 12
Core i9-13900K	Raptor Lake-S	24 / 32
Ryzen 5 5600X	Vermeer	6 / 12
Ryzen 5 5600G	Cezanne	6 / 12
Ryzen 7 5800X	Vermeer	8 / 16
Core i7-13700K	Raptor Lake-S	16 / 24

GPU:

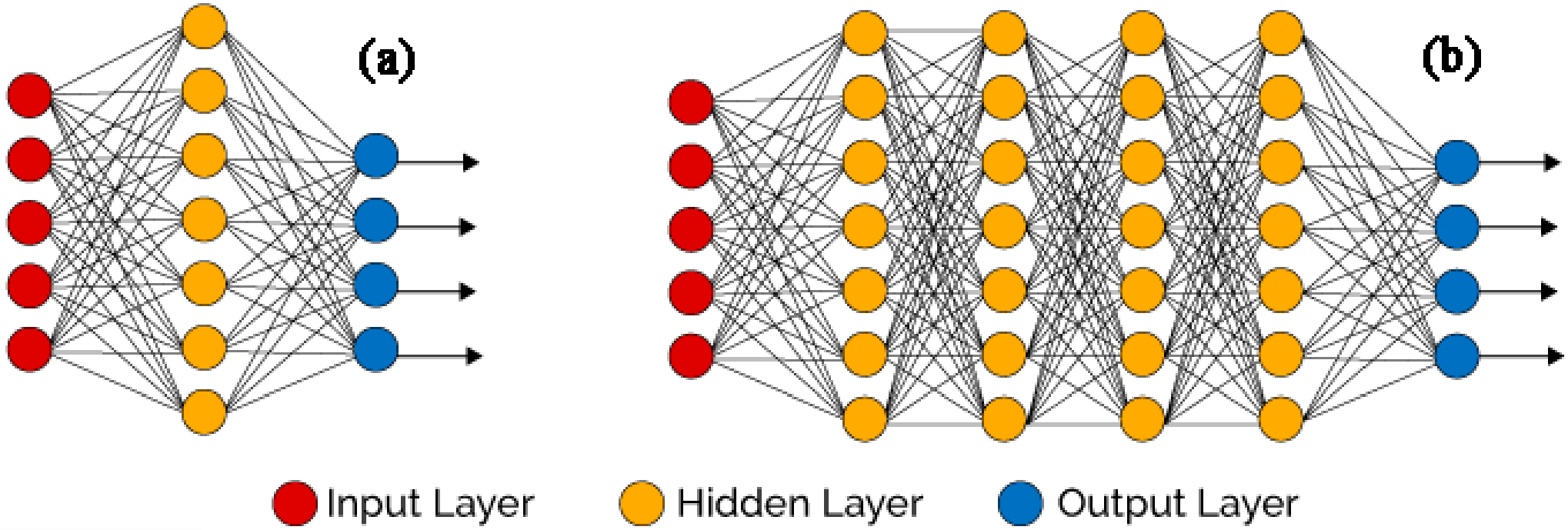
- Better memory bandwith utilization & SIMD
- CUDA cores are also usually a power of 2 / power of 2 * smth

GPU card	CUDA cores
GeForce GTX 1660 Ti	1536
GeForce GTX 1660 Super	1408
GeForce GTX 1660	1408
GeForce GTX 1650 Super	1408
GeForce GTX 1650	1024
GeForce GTX 1650	896
GeForce GTX 1060 3GB	1280


Power
Of
2
Power
Of
2
Power
Of
2
Power
Of
2
Neural Network For Image
BN
CBAM
Conv: 3x3
MP: 3x3
Conv: 3x3
MP: 3x3
Conv: 3x3
MP: 3x3
FCs
Image
input size
48x48x1
22x22x64
9x9x128
3x3x256
128
Neural Network For Landmarks
FC
FC
Landmarks
input size
68x2
1024
128
Neural Network For HOG
FC
FC
FC
HOG
input size
1x2728
4096
1024
128

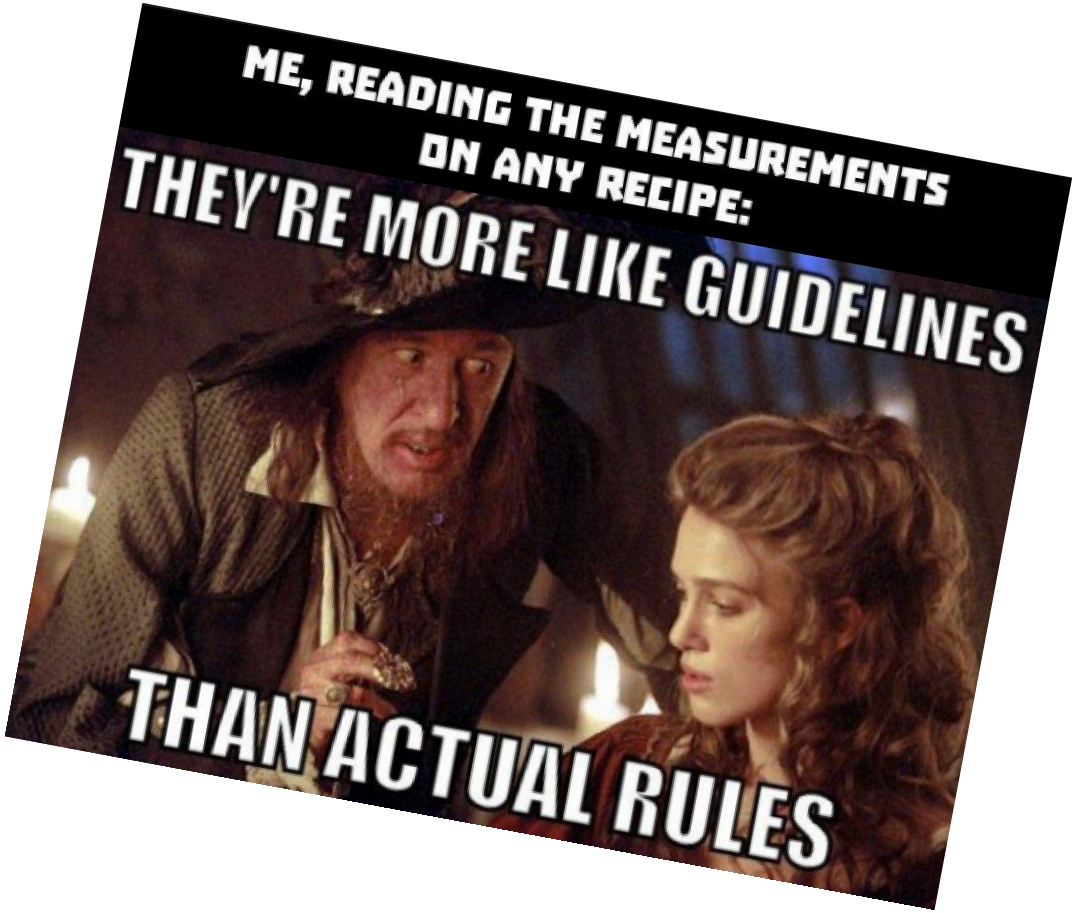
#2

How many Layers & Neurons per Layer?



There is no right answer...

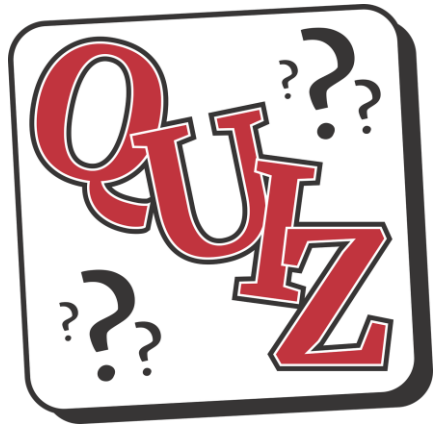
- I can only give you general guidelines
- Best suited for simple uses
- Use them when making your first model before iterating!



First, Hidden Layers count

- Simple Iris dataset with 4 inputs & 3 outputs
- Dog/Cat regressor (0/1)
- Dog breed classifier (on small pictures)
- Dog breed classifier Convolutional Neural network (count only dense layers)
- Understanding women NLP model

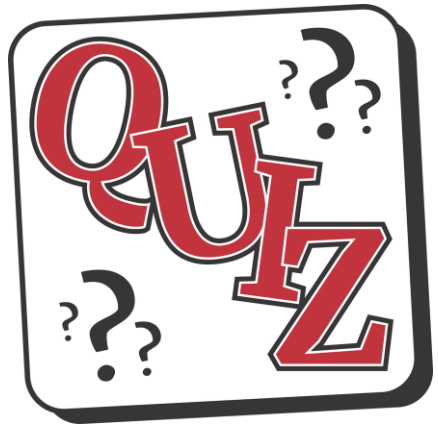
- 1
- 2
- 3
- 4+



First, Hidden Layers count

- Simple Iris dataset with 4 inputs & 3 outputs
- Dog/Cat regressor (0/1)
- Dog breed classifier (on small pictures)
- Dog breed classifier Convolutional Neural network (count only dense layers)
- Understanding women NLP model

- 1
- 2
- 3
- 4+

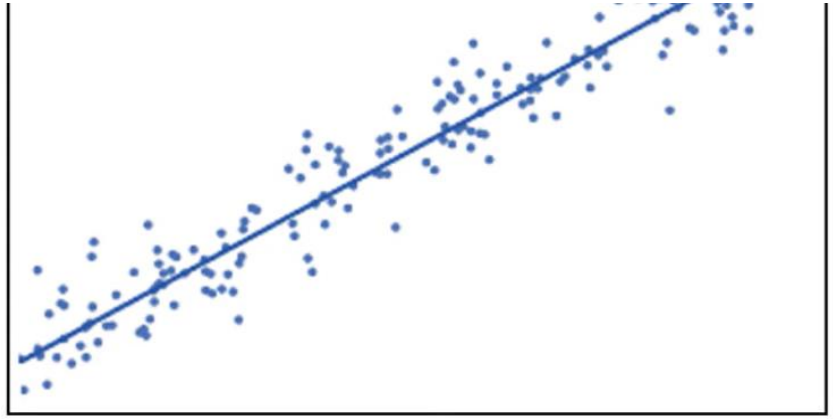


First, Hidden Layers count

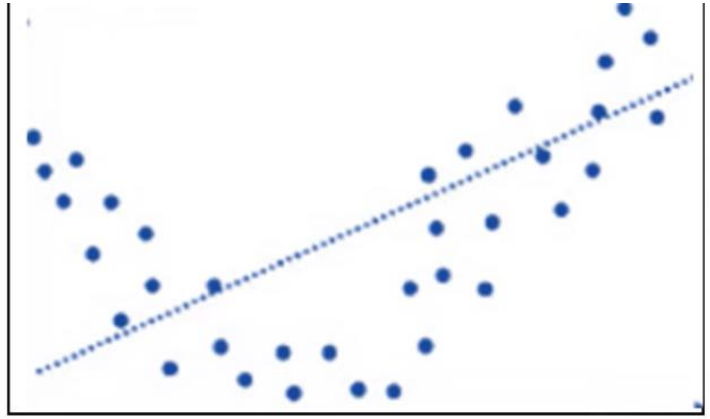
- 2 Hidden layers can map to any R^2 function!
- Too many layers will cause overfitting – your model output function should be simple & smooth

More precisely...

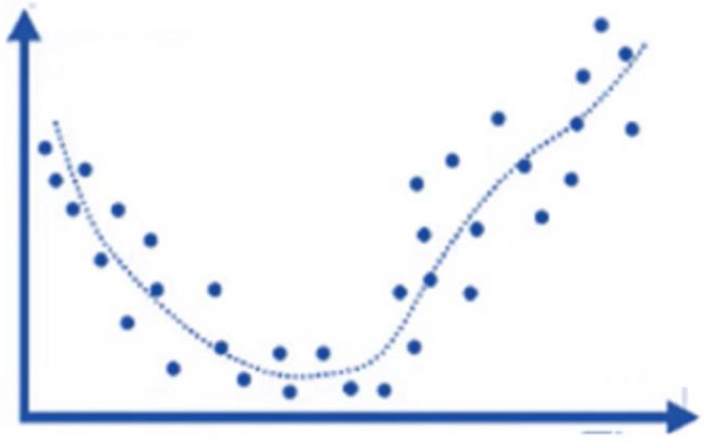
- 0 – linearly separable functions
- 1 – Simple continuous mapping
- 2 – Arbitrary 2 – dimensional decision boundary
- 3 – **VERY** complex functions in more dimensions



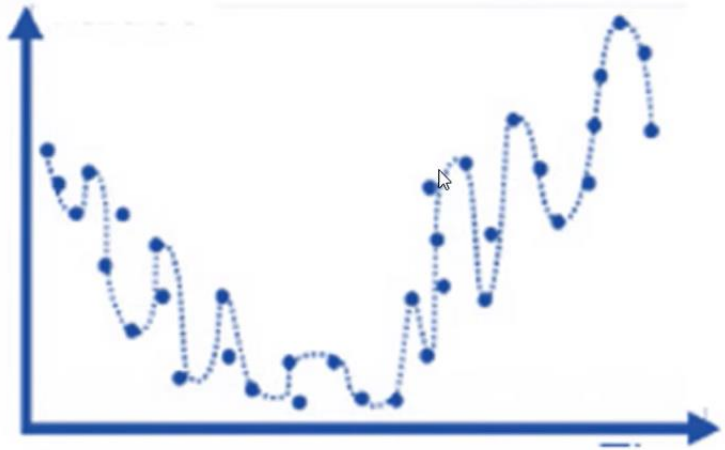
No hidden layers



Need hidden layers, otherwise we underfit the data



Just the right amount



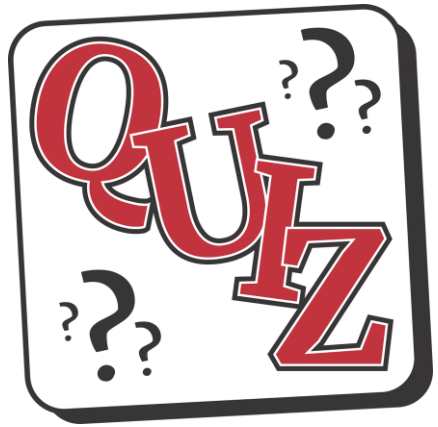
Too many hidden layers may lead to overfitting

Neurons count

- Simple dense neural network
- Dataset: S&P 500 stock price
- One row: 50 last candles open, low, close, high – vector of 200 values
- Output: 1 (will go up), 0 (neutral), -1 (will go down)

What should be the (first) layer size?

a) 4000, because 200^200 = combination of all features
b) 1024, because it's a nice number, also teacher said power of 2
c) 256, because it's a power of 2 and close to 200
d) 200, because it's the number of inputs
e) 128, because it's a power of 2 and less than input size

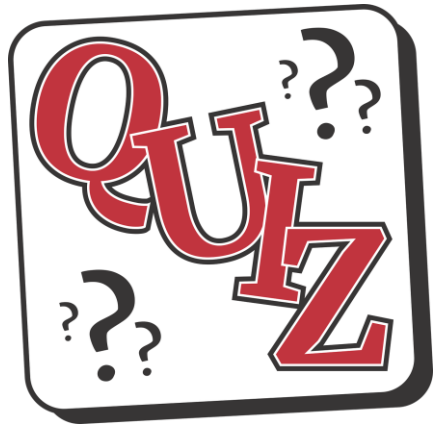


Neurons count

- Simple dense neural network
- Dataset: S&P 500 stock price
- One row: 50 last candles open, low, close, high – vector of 200 values
- Output: 1 (will go up), 0 (neutral), -1 (will go down)

What should be the (first) layer size?

a) 4000, because 200^200 = combination of all features
b) 1024, because it's a nice number, also teacher said power of 2
c) 256, because it's a power of 2 and close to 200
d) 200, because it's the number of inputs
e) 128, because it's a power of 2 and less than input size



Rule of Thumb guidelines

- Number of hidden neurons should be between the number of input and output sizes
- Simple equation: mean(input_size, output_size) + round DOWN to the nearest power of 2
- Round UP in more complex tasks, or when the neuron count is relatively low
- You can use the exact number of input size, but you ~~lose~~ DO NOT gain performance

#3

„You never know what's just enough, until you know what is too much"

- Second iteration: Either go twice as small, or twice as big
- Bigger when overfitting, smaller if not learning
- Do not make small additions hoping for the perfect neuron count to appear
- Learn how to visualize/display weights of your model – if after doubling the size, weights vector for neurons is like „one-hot" vector, reduce size – use „skip connection" instead (foreshadowing?)
- After doubling, freeze half of the weights and see how the validation loss changes – freeze at random at first, or use final layer weights to deremine the importance of each neuron

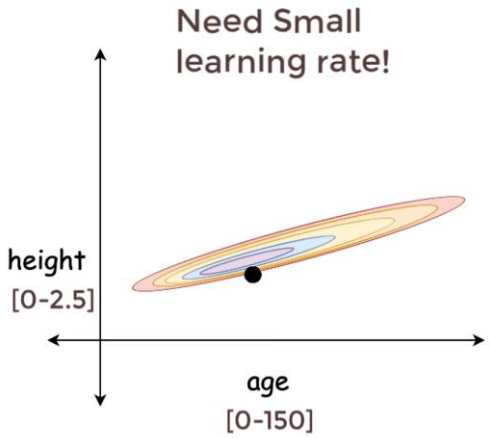
#4

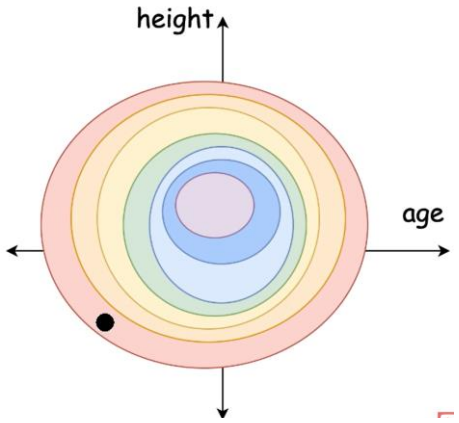
Honorable mentions – General Tips

#4a

Use Batch Normalization – (almost) always!

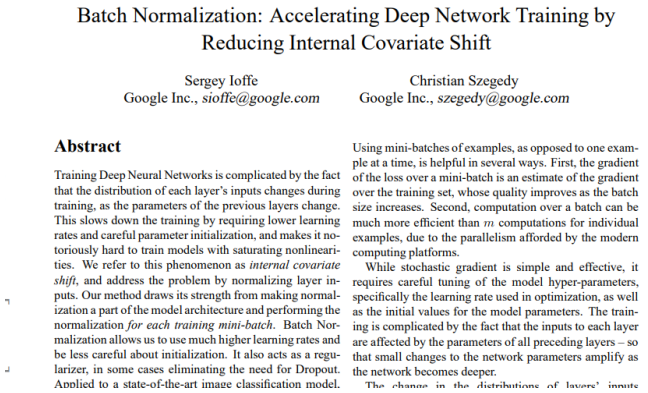
- Speeds up training
- Great with Adaptive Learning Rate optimizers (different learning rate for each parameter – for example, ADAM)
- Non – random regularizer
- Great with Dropout
- Smoothes out loss function – by subtracting the mean and dividing by stddev – allows big learning rates
- Currently under research





https://arxiv.org/pdf/1805.11604.pdf

https://arxiv.org/pdf/1502.03167.pdf



Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe
Google Inc., *sioffe@google.com*

Christian Szegedy
Google Inc., *szegedy@google.com*

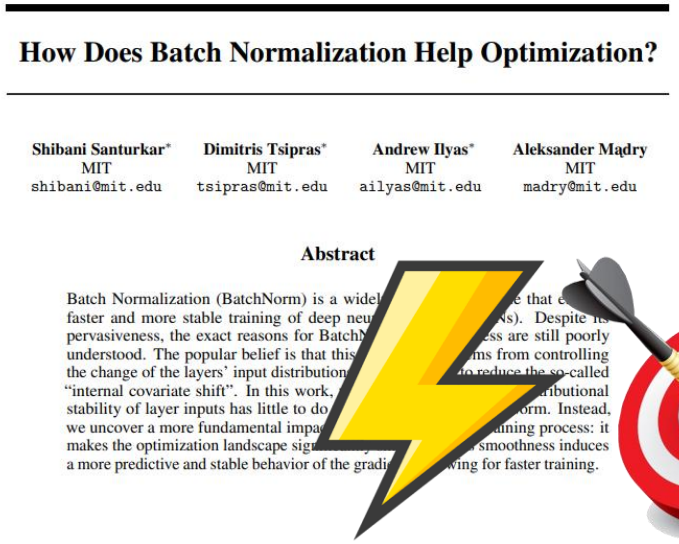
Abstract

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as *internal covariate shift*, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization *for each training mini-batch*. Batch Normalization allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularizer, in some cases eliminating the need for Dropout. Applied to a state-of-the-art image classification model,

Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways. First, the gradient of the loss over a mini-batch is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a batch can be much more efficient than m computations for individual examples, due to the parallelism afforded by the modern computing platforms.

While stochastic gradient is simple and effective, it requires careful tuning of the model hyper-parameters, specifically the learning rate used in optimization, as well as the initial values for the model parameters. The training is complicated by the fact that the inputs to each layer are affected by the parameters of all preceding layers – so that small changes to the network parameters amplify as the network becomes deeper.

The change in the distributions of layers' inputs



How Does Batch Normalization Help Optimization?

Shibani Santurkar* MIT shibani@mit.edu

Dimitris Tsipras* MIT tsipras@mit.edu

Andrew Ilyas* MIT ailyas@mit.edu

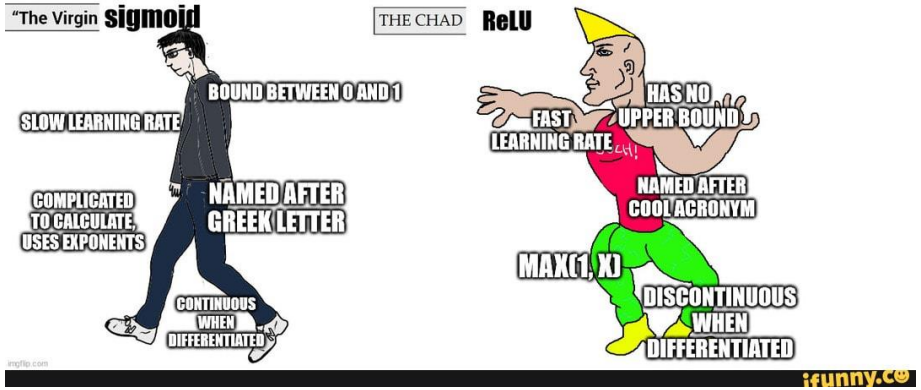
Aleksander Mądry MIT madry@mit.edu

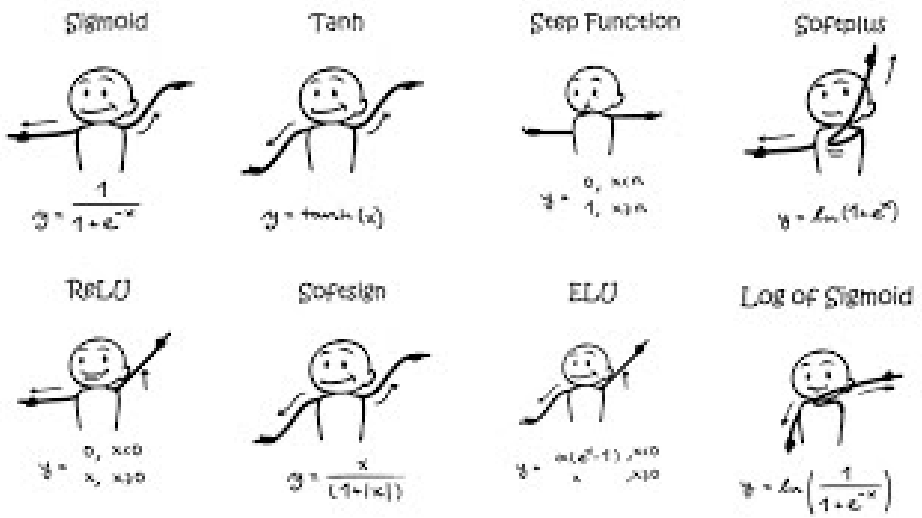
Abstract

Batch Normalization (BatchNorm) is a widel… that e… faster and more stable training of deep neu… Ns). Despite its… pervasiveness, the exact reasons for Batch… …ess are still poorly understood. The popular belief is that this… …ms from controlling the change of the layers' input distribution… …o reduce the so-called "internal covariate shift". In this work, … …ributional stability of layer inputs has little to do… …orm. Instead, we uncover a more fundamental impa… …aining process: it makes the optimization landscape sig… …smoothness induces a more predictive and stable behavior of the gradi… …wing for faster training.

#4b

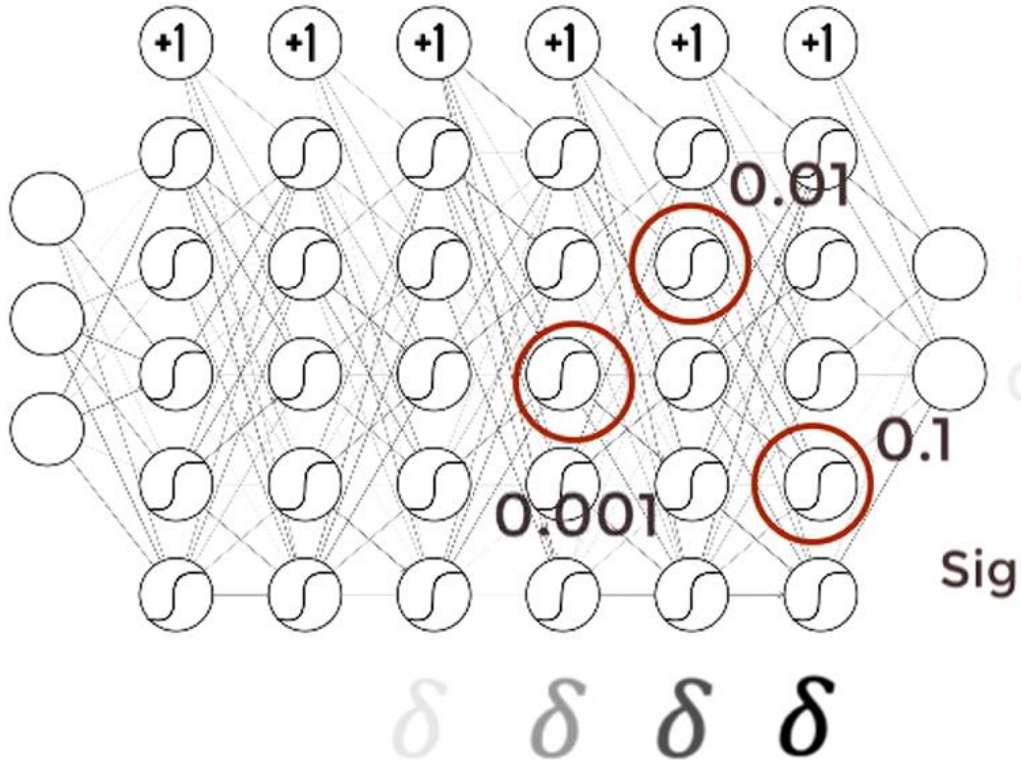
Learn modern activation functions!





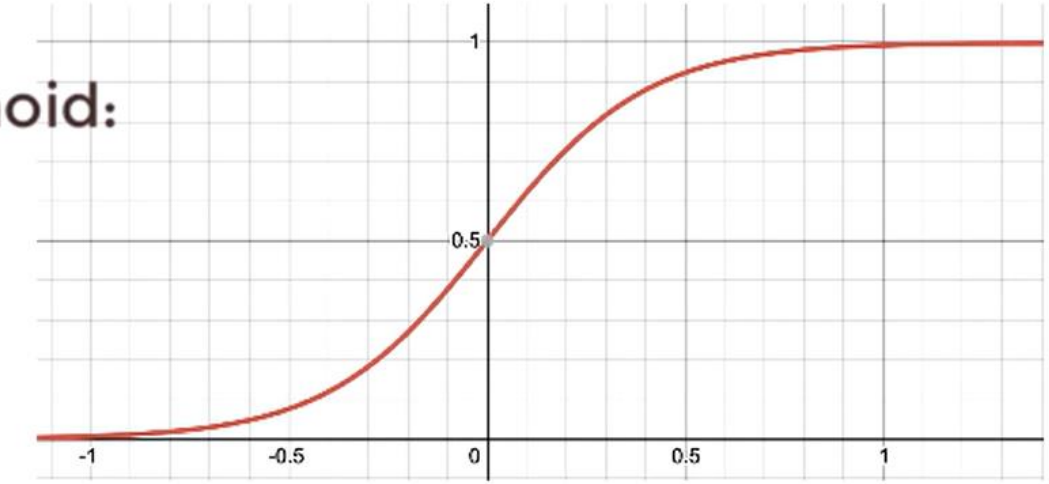
WHY IS RELU BETTER?

Why do gradient vanish with sigmoid?

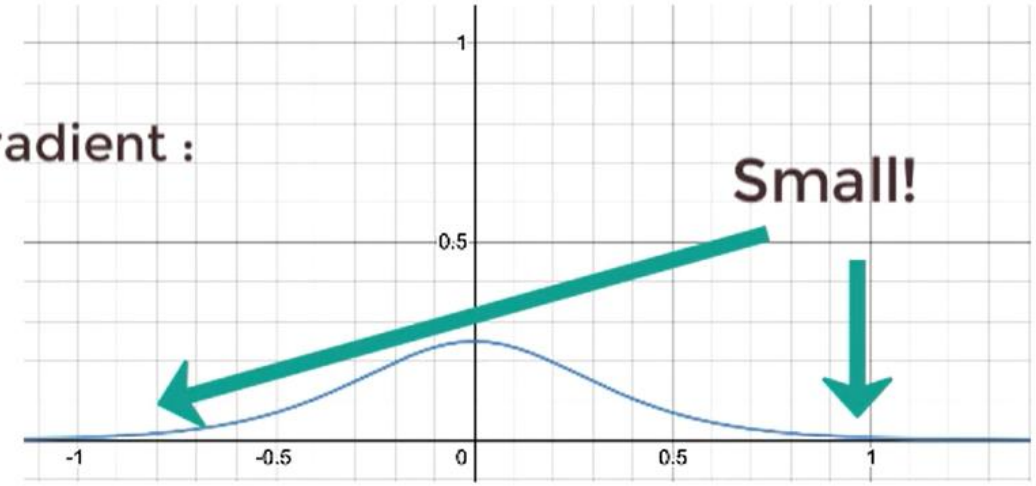


Vanishing Gradient Problem

Sigmoid:



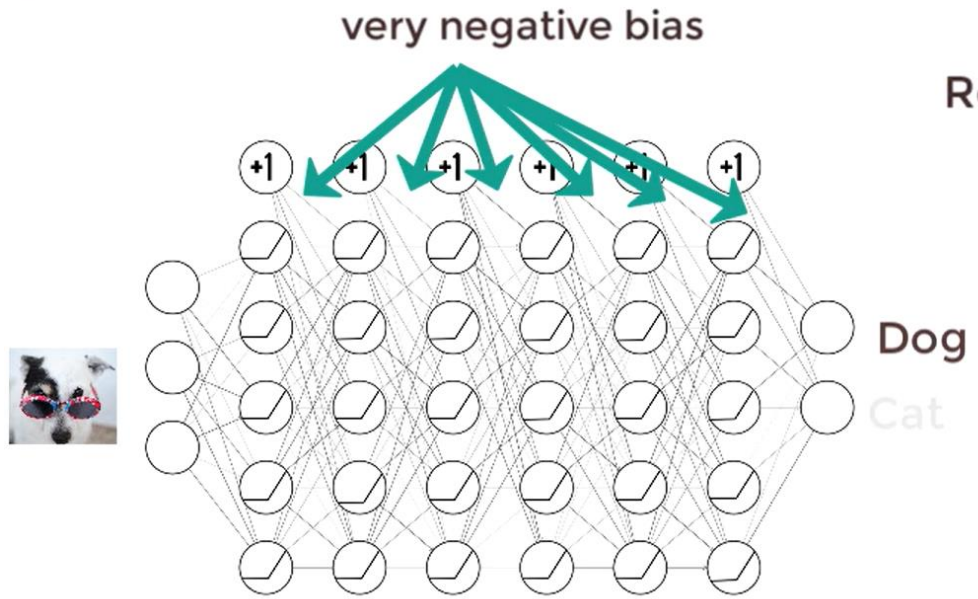
Sigmoid Gradient :



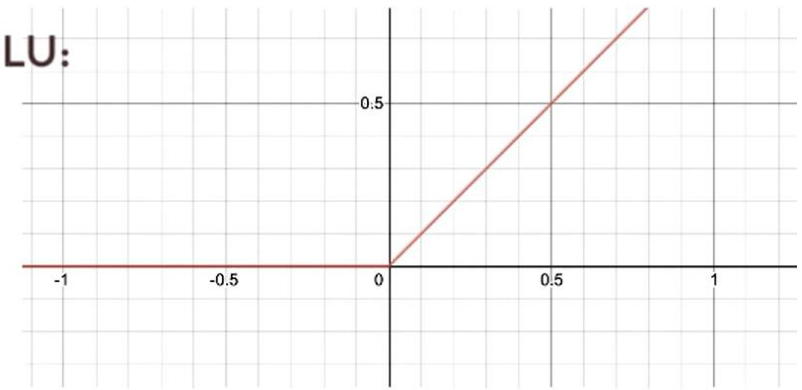
Why not spam ReLU?

- It should NOT be used in input and output layers
- It can „kill” your data
- Regardless of the value inputted, gradient is always one – can „kill” neurons – Dying ReLU Problem with momentum, or heavily negative bias
- Not really „logical” – gradient value is also an information
- TanH, Sigmoid, Softmax – can normalize data

Why do neurons "die"?



ReLU:



Dying ReLU Problem

$$ReLU(Wx + b) = 0$$

$$if \ \max(Wx + b, 0) = 0$$

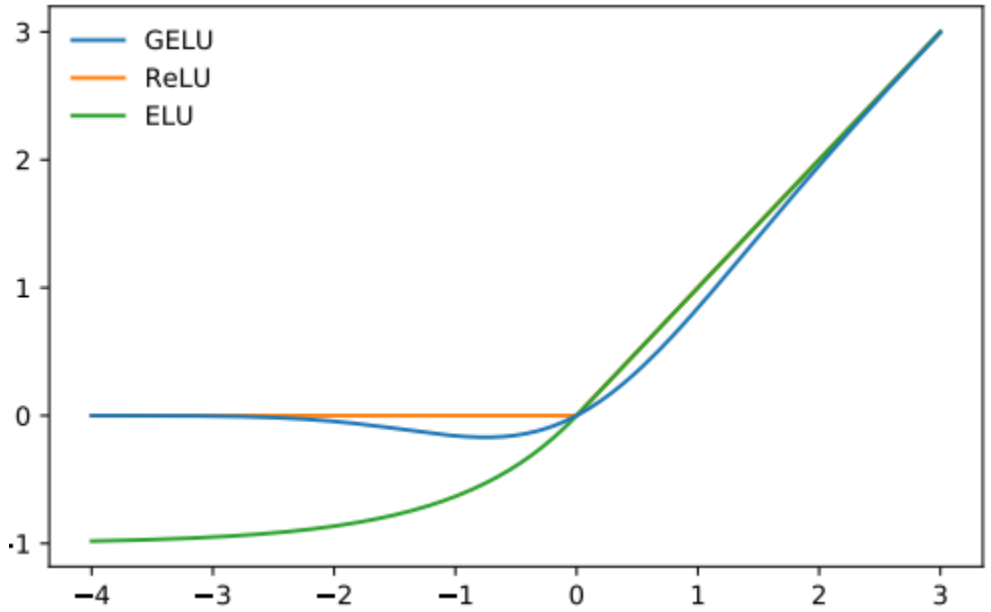
$$or \ Wx + b < 0 \ \ or \ b < -Wx$$

This happens with very neagtive bias when W,x > 0

Do not be scared to experiment!

Examples of functions to learn:

- GELU (Gaussian Error Linear Unit) – modern ReLU – you can spam this one instead
- SiLU - Swish activation function (x * sigmoid(beta * x)), usually beta is 1
- ELU - Exponential linear function
- Leaky ReLU – I assume most know about this one but probably never tried



All of them are better than ReLU! (In most cases)

#4c

Learn Optimizers Pros & Cons!

Gradient Descent:

$$\theta = \theta - \alpha \,.\nabla_\theta J(\theta)$$

Stochastic Gradient Descent

$$\theta = \theta - \alpha \,.\nabla_\theta J(\theta; sample)$$

Mini-Batch Gradient Descent

$$\theta = \theta - \alpha \,.\nabla_\theta J(\theta; N\ samples)$$

SGD + Momentum

$$v = \gamma . v + \eta . \nabla_\theta J(\theta)$$
$$\theta = \theta - \alpha\, v$$

SGD + Momentum + Acceleration

$$v = \gamma . v + \eta . \nabla_\theta J(\theta - \gamma . v)$$
$$\theta = \theta - \alpha\, v$$

Adagrad

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii}} + \epsilon} \nabla_{\theta_{t,i}} J(\theta_{t,i})$$

Adadelta

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{E[G_{t,ii}] + \epsilon}} \nabla_{\theta_{t,i}} J(\theta_{t,i})$$

Adam

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{E[G_{t,ii}] + \epsilon}} \times \mathrm{E}[g_{t,i}]$$

#4c

Learn Optimizers Pros & Cons!

- Gradient Descent:
  - Batch gradient descent – never use
  - Stochastic gradient descent – can be used with better results than mini-batch when dataset is simple & without many „wild outliers”
  - Mini-batch gradient descent – best choice, but will require learning rate tuning. Not good for small datasets. Might get stuck in suboptimal local minima
  - SGD + Momentum + Acceleration – reduces the „noisiness”, resistant to random outliers
- Adaptative:
  - Adagrad – prioritizes the „rare” features by giving them bigger updates. Uses the past gradients values to determine which ones are „rare” – might give extremely low learning rate to „frequent” features – never use
  - Adadelta – Adagrad that uses a history window
  - RMSProp – Basically a bit different Adadelta, uses a different update method
  - Adam –RMSProp + Adadelta (momentum – like) – Best overall, use in most cases
  - Nadam - Adam with Nesterov momentum – newest toy in the arsenal, try it

Quiz time!

When to use Batch Normalization?

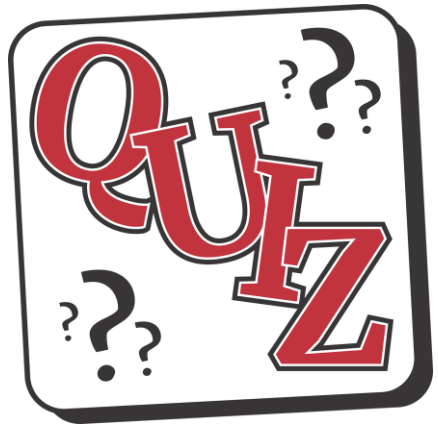
a) Always
b) When your activation function does not normalize data
c) When your teacher asks you to do so
d) Never

What is currently the best activation function?

a) ReLU
b) TanH
c) Softmax
d) GELU

What should be your default optimizer?

a) Adam
b) Nadam
c) SGD
d) SGD with Momentum



Quiz time!

When to use Batch Normalization?

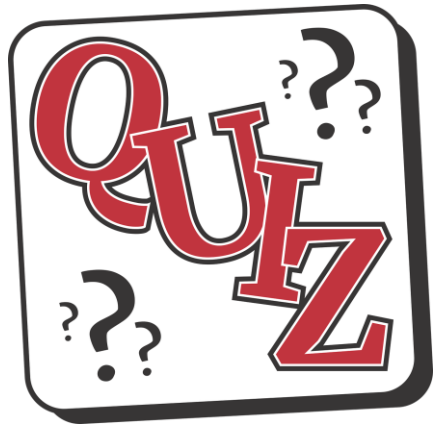
a) Always
b) When your activation function does not normalize data
c) When your teacher asks you to do so
d) Never

What is currently the best activation function?

a) ReLU
b) TanH
c) Softmax
d) GELU

What should be your default optimizer?

a) Adam
b) Nadam
c) SGD
d) SGD + Momentum + Acceleration



Why even mention that? Why not just use Adam/Nadam?

- It turns out SGD with precisely selected learning rate is better
- Topic of current research
- Adam is sufficient in most cases, but reduces the need for YOU to learn – hyperparameter tuning is a valuable skill
- Why is SGD better? Even they do not know... Yet
- Low momentum + Acceleration can cause overfitting

Towards Theoretically Understanding Why SGD Generalizes Better Than ADAM in Deep Learning

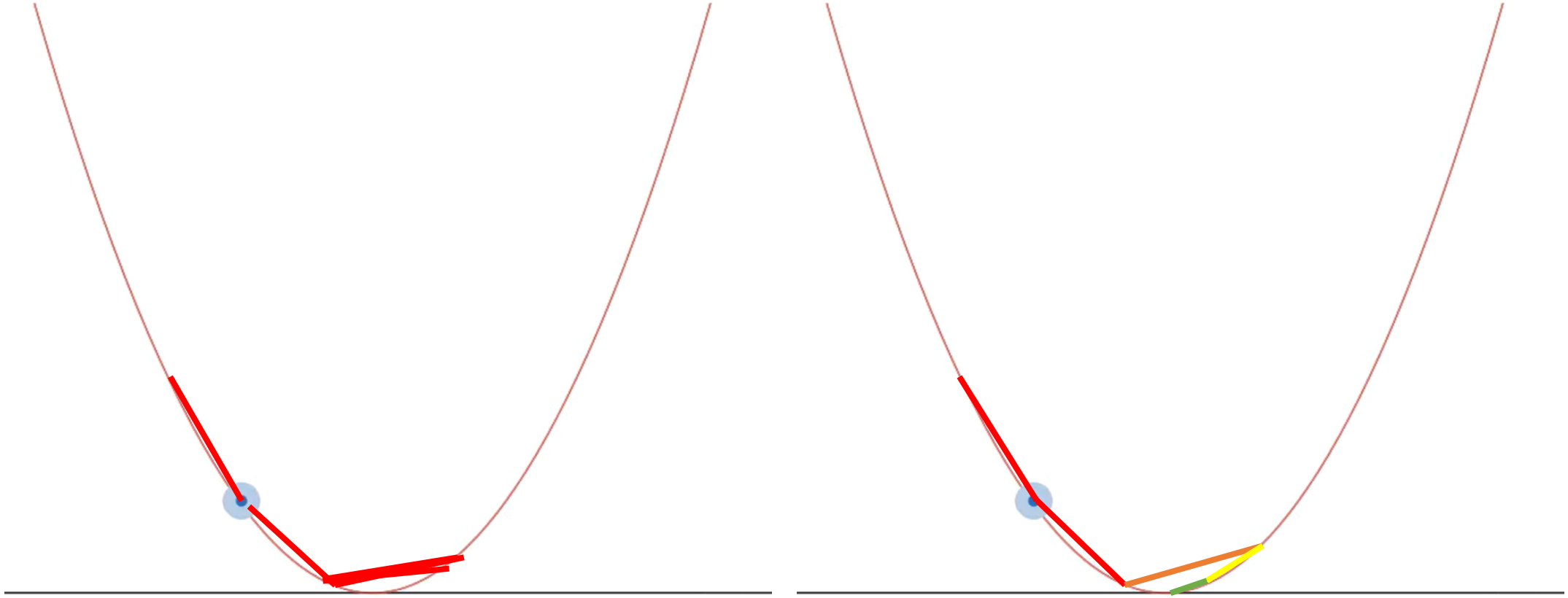
Pan Zhou*, Jiashi Feng†, Chao Ma‡, Caiming Xiong*, Steven HOI*, Weinan E‡
*Salesforce Research, † National University of Singapore, ‡ Princeton University
{pzhou,shoi,cxiong}@salesforce.com elefjia@nus.edu.sg {chaom@, weinan@math.}princeton.edu

Abstract

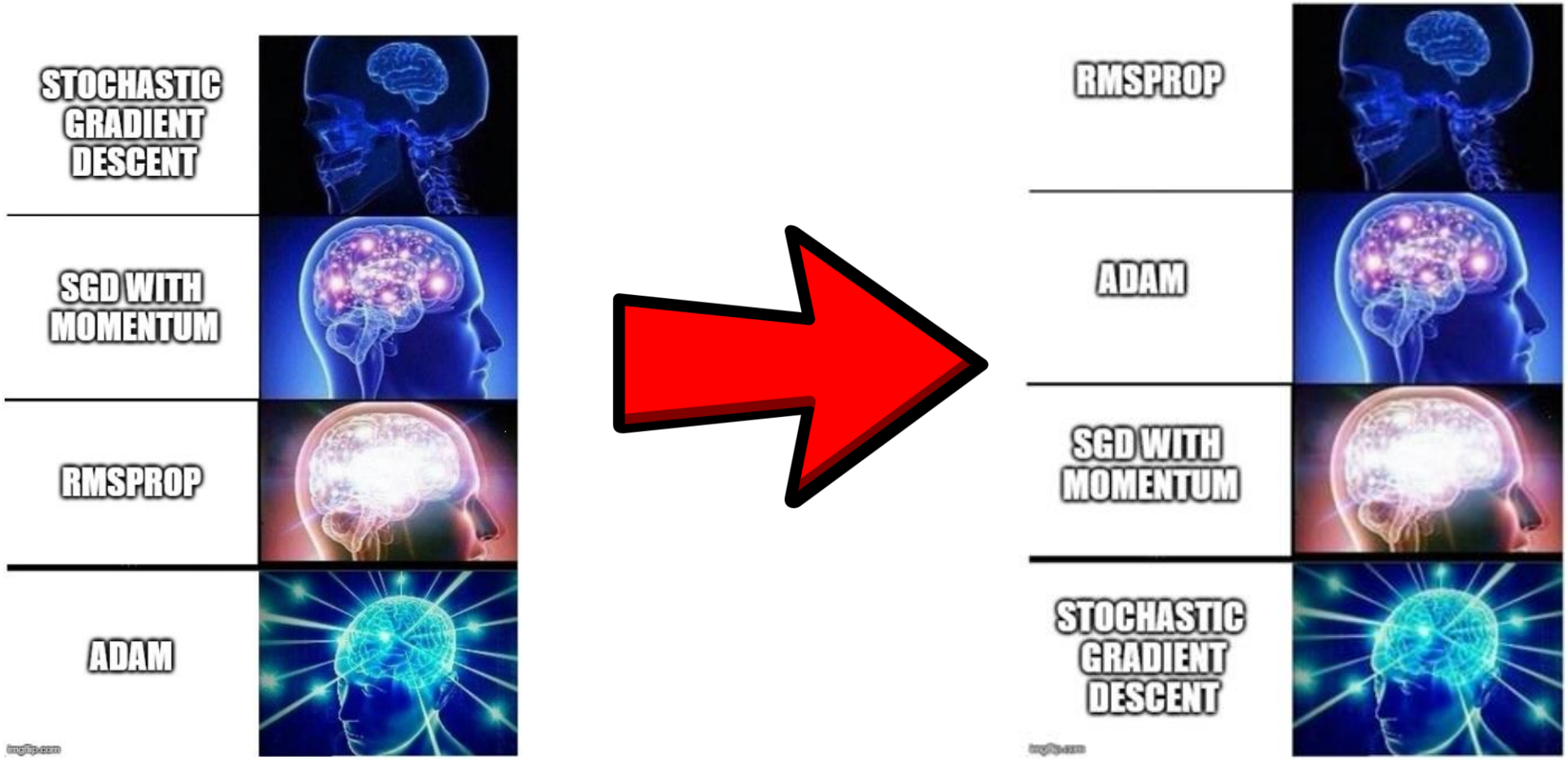
It is not clear yet why ADAM-alike adaptive gradient algorithms suffer from worse generalization performance than SGD despite their faster training speed. This work aims to provide understandings on this generalization gap by analyzing their local convergence behaviors. Specifically, we observe the heavy tails of gradient noise in these algorithms. This motivates us to analyze these algorithms through their Lévy-driven stochastic differential equations (SDEs) because of the similar convergence behaviors of an algorithm and its SDE. Then we establish the escaping time of these SDEs from a local basin. The result shows that (1) the escaping time of both SGD and ADAM depends on the Radon measure of the basin positively and the heaviness of gradient noise negatively; (2) for the same basin, SGD enjoys smaller escaping time than ADAM, mainly because (a) the geometry adaptation in ADAM via adaptively scaling each gradient coordinate well diminishes the anisotropic structure in gradient noise and results in larger Radon measure of a basin; (b) the exponential gradient average in ADAM smooths its gradient and leads to lighter gradient noise tails than SGD. So SGD is more locally unstable than ADAM at sharp minima defined as the minima whose local basins have small Radon measure, and can better escape from them to flatter ones with larger Radon measure. As flat minima here which often refer to the minima at flat or asymmetric basins/valleys often generalize better than sharp ones [1, 2], our result explains the better generalization performance of SGD over ADAM. Finally, experimental results confirm our heavy-tailed gradient noise assumption and theoretical affirmation.

https://proceedings.neurips.cc/paper/2020/file/f3f27a324736617f20abbf2ffd806f6d-Paper.pdf

SGD vs Adam – lower loss is not always better



Even memes can lie



#5

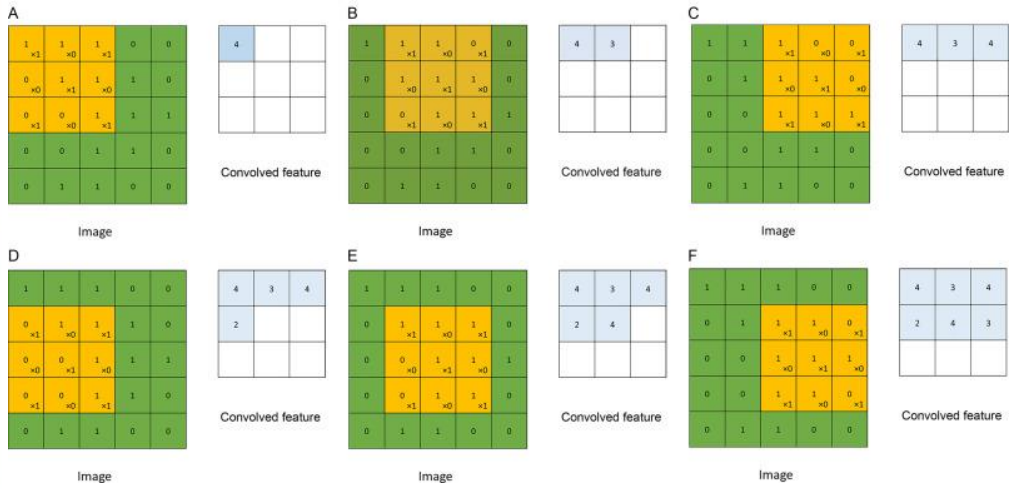
CNN Modelling:

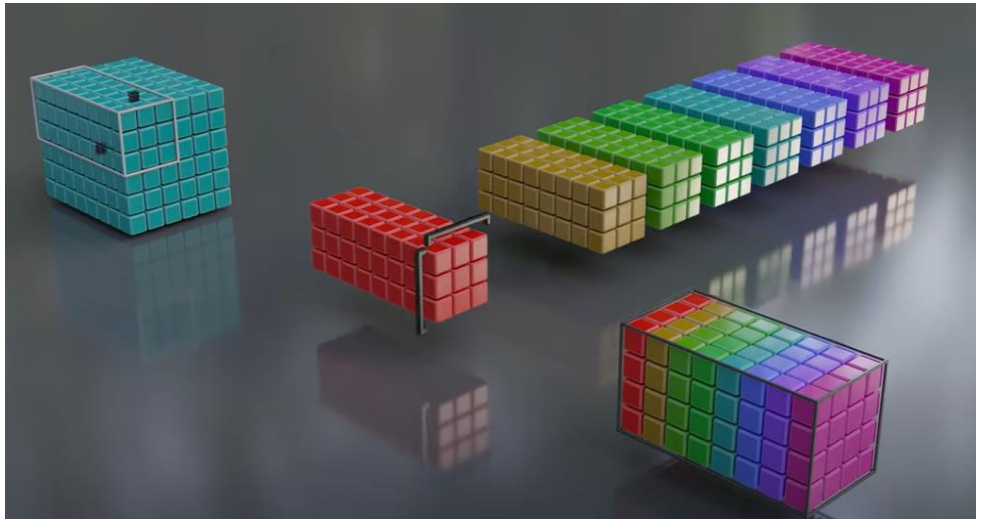
How many layers, filters per layer & filter size?

The ~~Bigger~~ The Better

Smaller

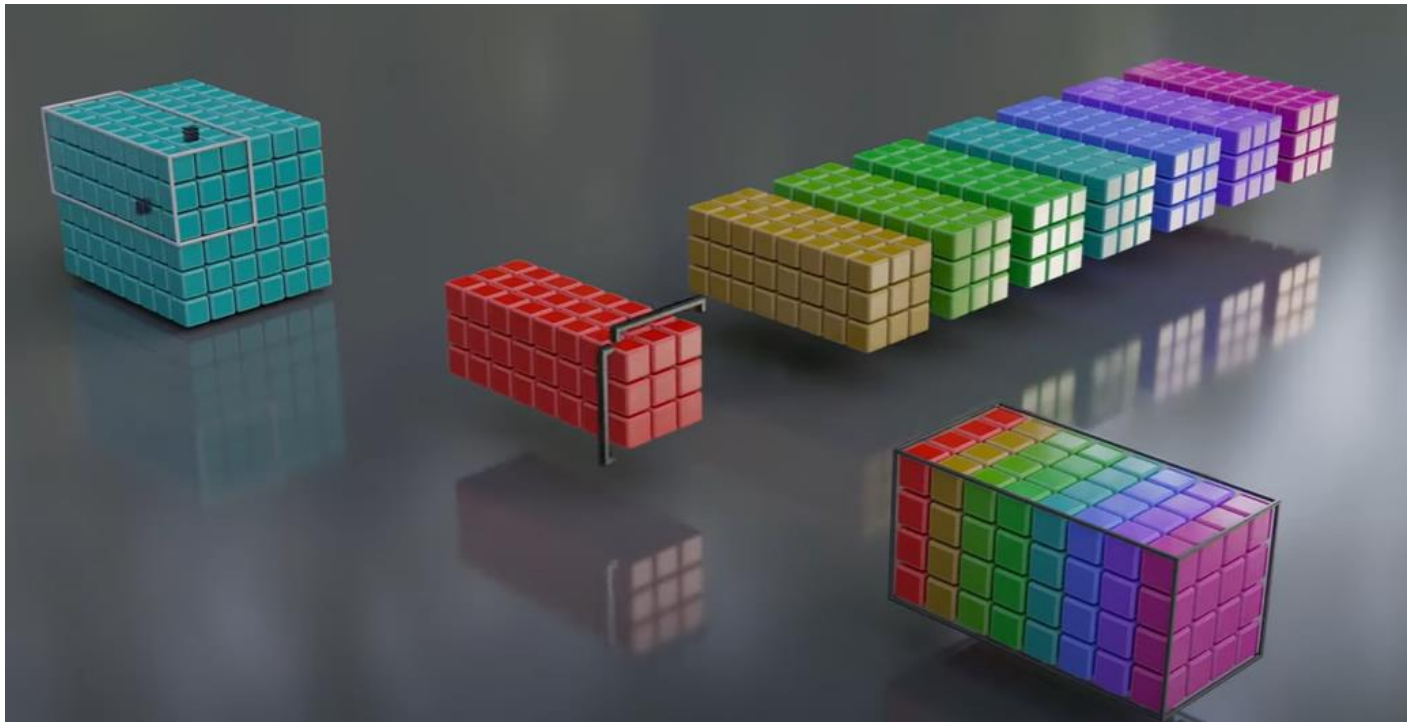
What is a „Conv2D”





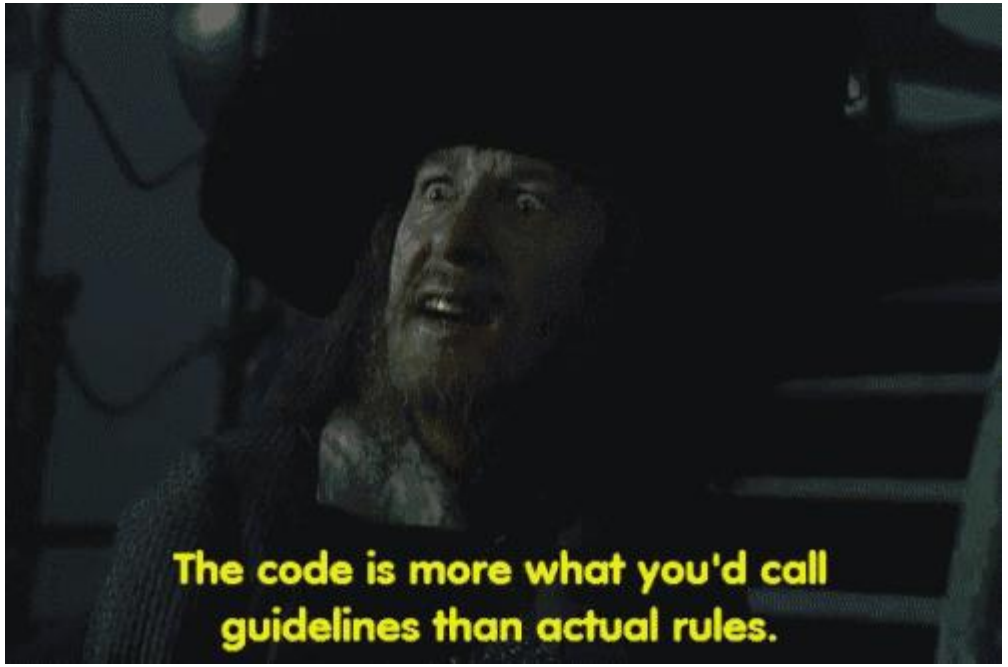
When using Conv2D...

- Do NOT mess with its output – every output matrix is a 2-dimensional feature map, that has positional information
- Do NOT „reshape” the output
- Use odd values for filter size
- Use „same” padding
- Learn how to extract these arrays from your models!
- Nowadays it’s common to use pre-trained Conv2Ds, since first few always learn some low-level features – learn how to do it and apply it
- Recommended models: YOLO, RetinaNet, DarkNet, AlexNet



Filter Count

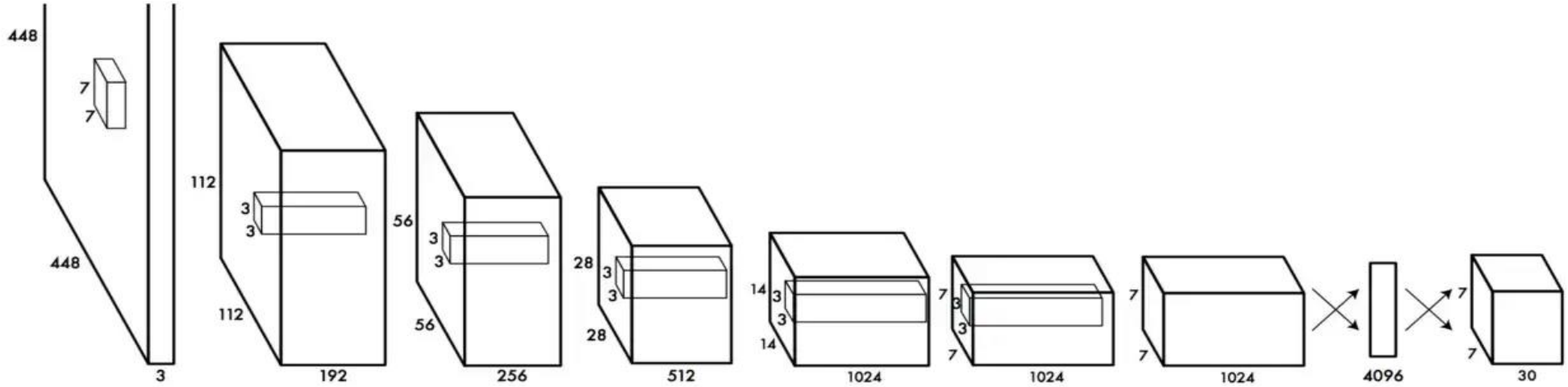
- It’s actually quite hard to OVERESTIMATE this number
- It could go as high as ½ of the input width/height
- Filter count should DOUBLE every time you reduce the image size by a half



Rule of Thumb – CNN Models

- Add Convolutional layers until the input size is below 10
- After each Convolution, either use MaxPool2D 2x2 with stride 2, or use stride 2 in the Convolution – more about this next
- Add the final Convolution layer, with filter size set to the dimensions of the previous layer
- Flatten the output (should be 1x1x..... till now)
- 2 Dense Layers as mentioned before
- Output layer with size = number of classes

YOLO v1 Example:

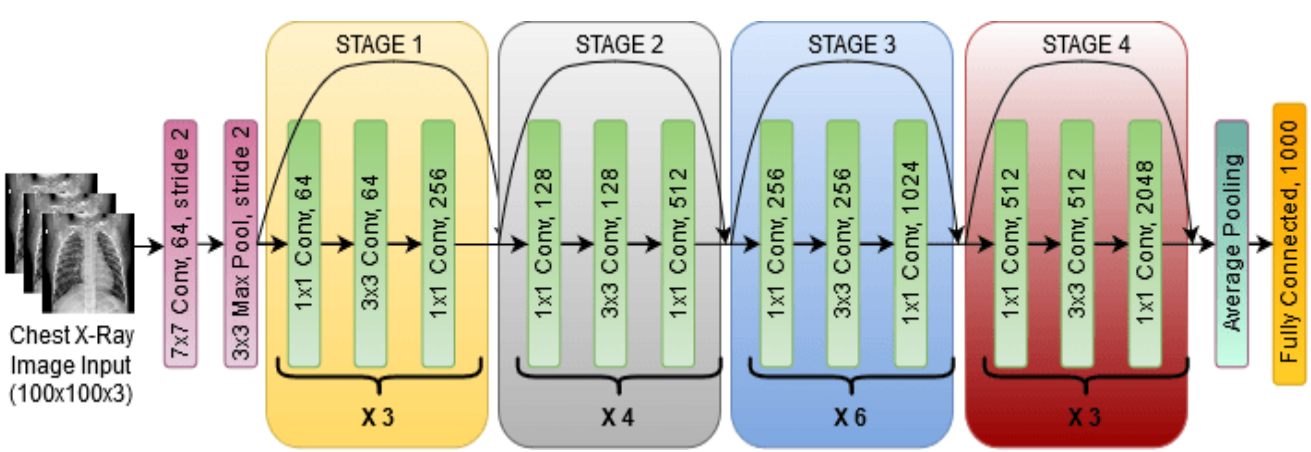


#6
„Yin-Yang principle”

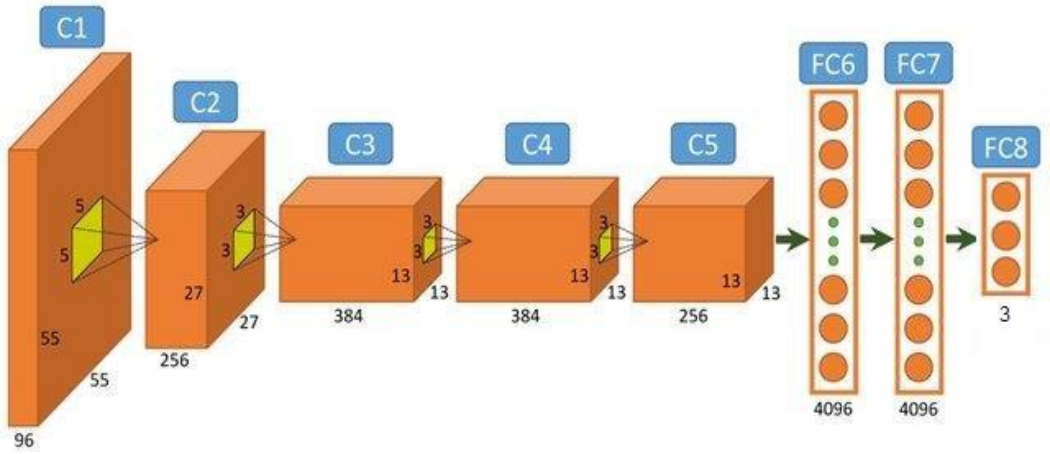
- Before using MaxPool2D, use two Convolutions with equal shape and ‚same’ padding – first usually learns fresh features, second one refines them
- Second Convolution can be used as MaxPool if used with stride = 2 – it can save a lot of computations
- Both options (double Connvolutional Layer & MaxPool, double Convolutional layer with second layer’s stride = 2) are used.

Examples:

ResNet (stride trick)



FishNet (maxpool)



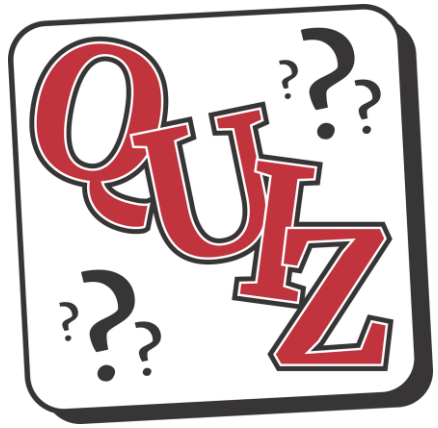
Convolution Size (first few layers)

Image sizes:

- 64x64
- 128x128
- 256x256
- 512x512
- 1024x1024

Convolution sizes:

- 2x2
- 3x3
- 4x4
- 5x5
- 6x6
- 7x7



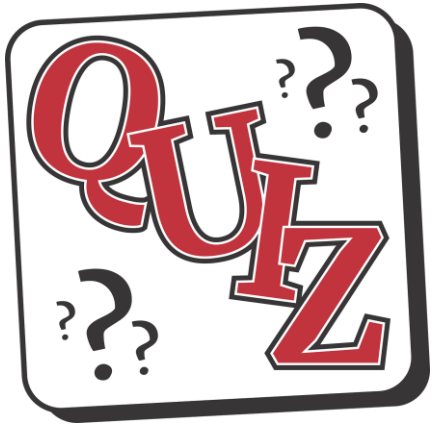
Convolution Size (first few layers)

Image sizes:

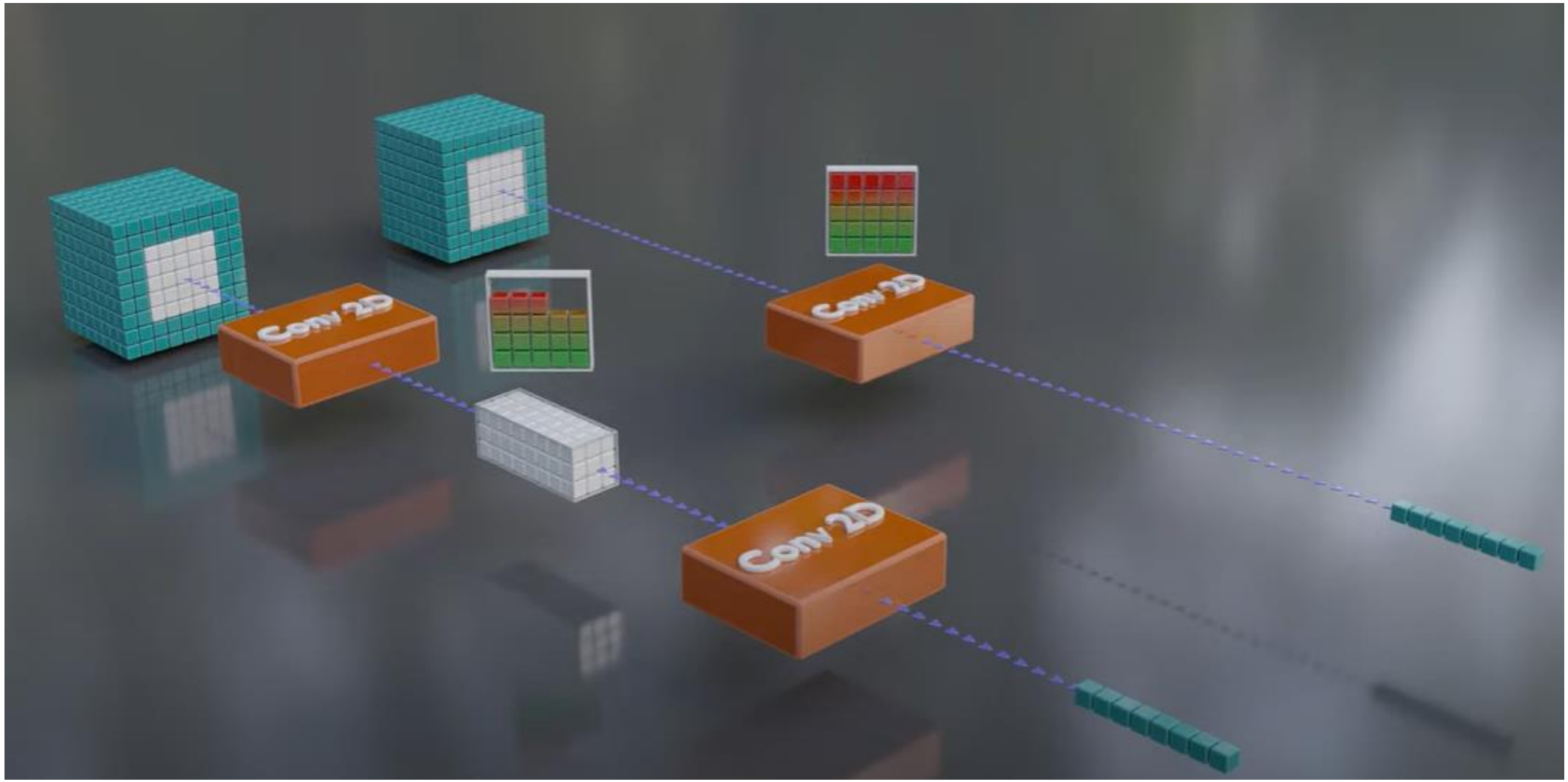
- 64x64
- 128x128
- 256x256
- 512x512
- 1024x1024

Convolution sizes:

- 2x2
- 3x3
- 4x4
- 5x5
- 6x6
- 7x7



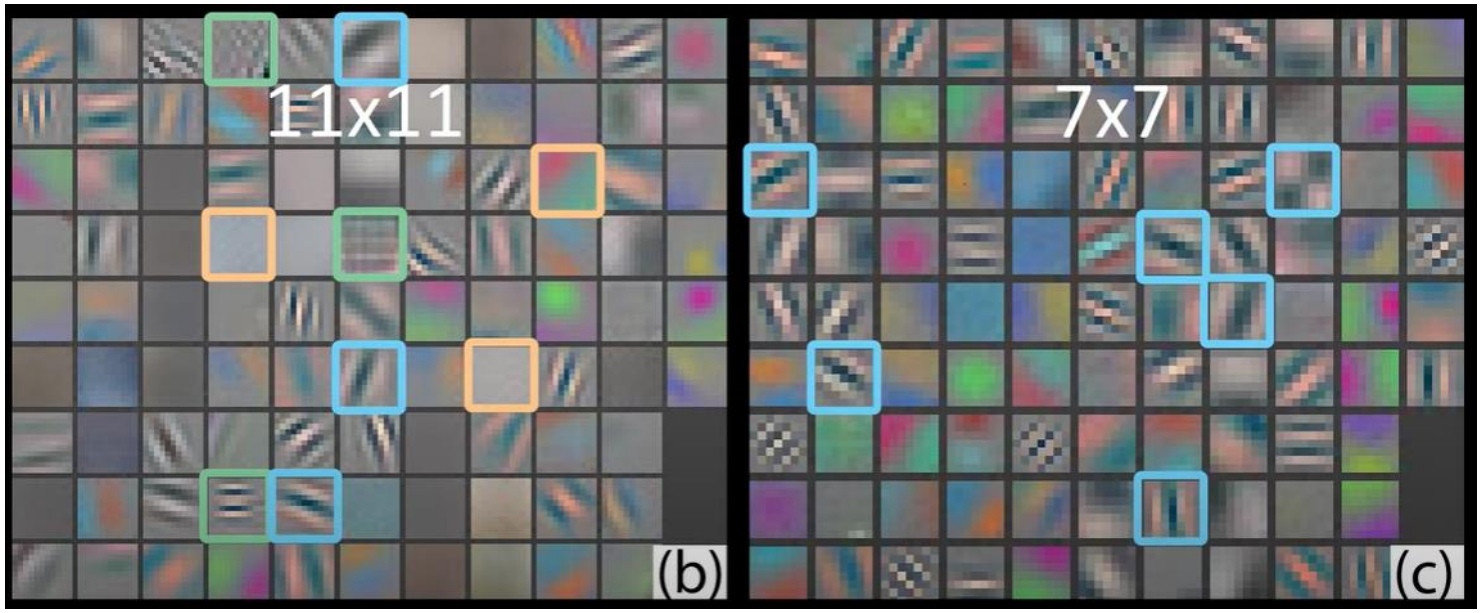
Concept of „Convolutional Filter Vision”



Two 3x3 filters use less weights (72%) but see the same features!

Proof?

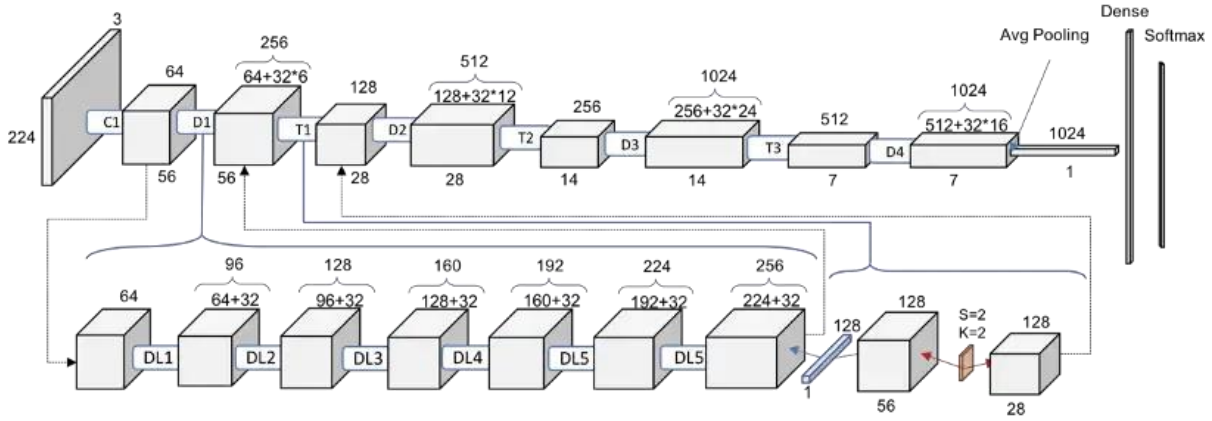
- AlexNet (2012): first Convolution: 11x11, second: 5x5, then 3x3
- ZFNet, aka Optimized AlexNet (2013): 7x7, then 5x5, 3x3



- VGG (2014): Only 3x3 Convolutions

#7

CNN Input Size



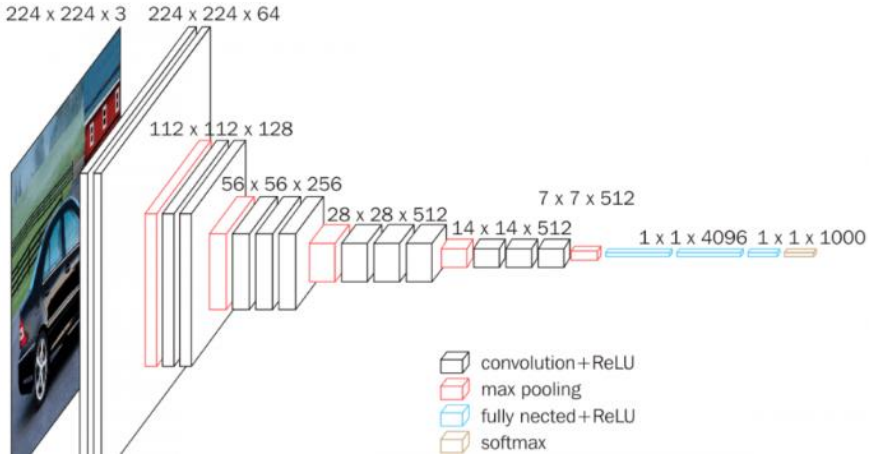
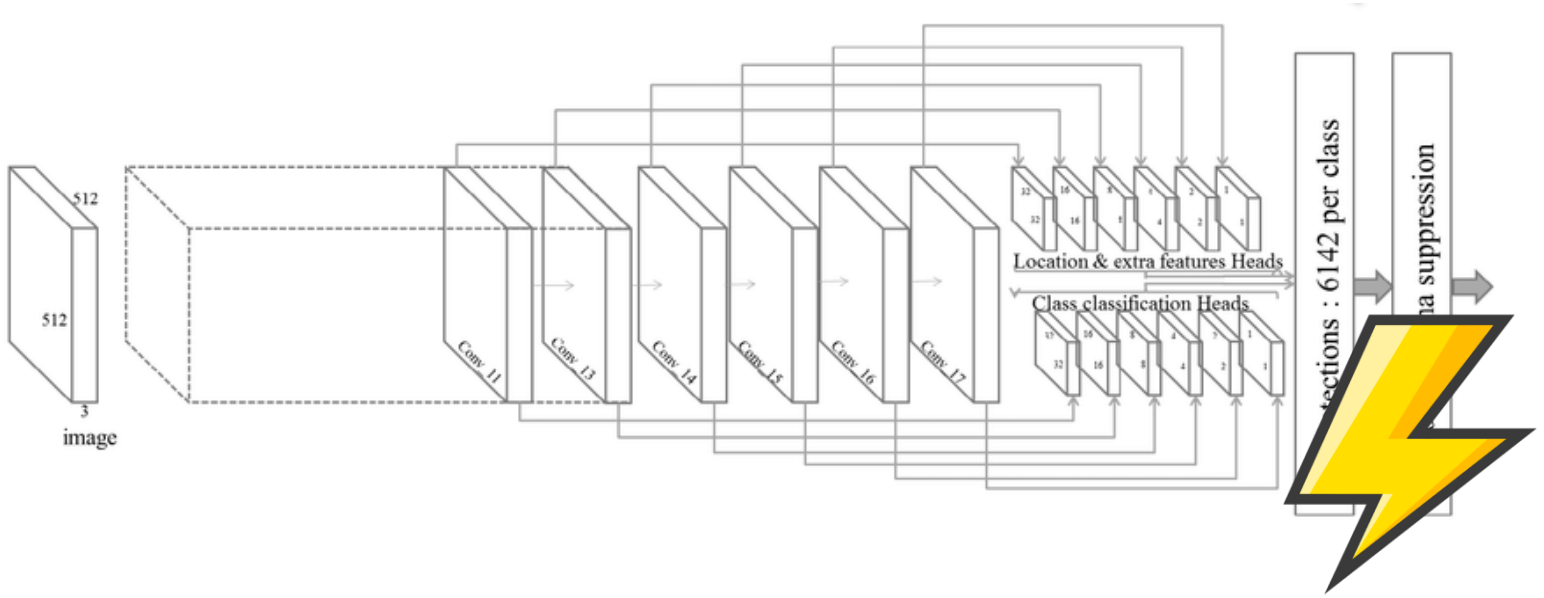


Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	3 × 3 × 3 × 32	224 × 224 × 3
Conv dw / s1	3 × 3 × 32 dw	112 × 112 × 32
Conv / s1	1 × 1 × 32 × 64	112 × 112 × 32
Conv dw / s2	3 × 3 × 64 dw	112 × 112 × 64
Conv / s1	1 × 1 × 64 × 128	56 × 56 × 64
Conv dw / s1	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 128	56 × 56 × 128
Conv dw / s2	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 256	28 × 28 × 128
Conv dw / s1	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 256	28 × 28 × 256
Conv dw / s2	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 512	14 × 14 × 256
5× Conv dw / s1	3 × 3 × 512 dw	14 × 14 × 512
5× Conv / s1	1 × 1 × 512 × 512	14 × 14 × 512
Conv dw / s2	3 × 3 × 512 dw	14 × 14 × 512
Conv / s1	1 × 1 × 512 × 1024	7 × 7 × 512
Conv dw / s2	3 × 3 × 1024 dw	7 × 7 × 1024
Conv / s1	1 × 1 × 1024 × 1024	7 × 7 × 1024
Avg Pool / s1	Pool 7 × 7	7 × 7 × 1024
FC / s1	1024 × 1000	1 × 1 × 1024
Softmax / s1	Classifier	1 × 1 × 1000



Why 224x224? Why not 256?

These networks use 224x224 input size:

- VGG models (VGG16, VGG19)
- Resnet and it’s variants (Resnet18, Resnet34, Resnet50,..)
- Mobilenet architecture (mobilenetv1, mobilenetv2 and mobilenetv3)

Should you?

Recall power of 

		Closest power of 2:		
For input 224 x 224 x 3, total values count:	150,528	2^17 = 131,072	Delta: 19 456	224 can be divided by 2 5 times, then it reaches value of 7 (odd & <10, perfect for final processing)
For input 256 x 256 x 3, total values count:	196 608	2^17 = 131,072 2^18 = 262,144	Delta: 65,536 (both)	224 can be divided by 2 5 times, then it reaches value of 8 (even)

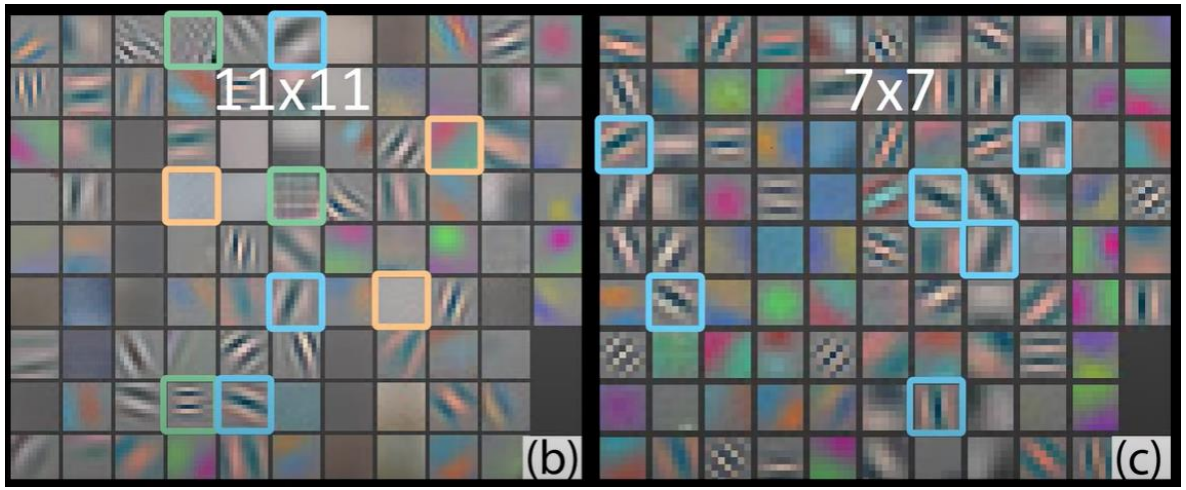
In summary, saves computation & provides an odd dimensions for final Convolutional filter – therefore, use image sizes that have an odd (prime?) divisor, and a bunch of 2s

#8

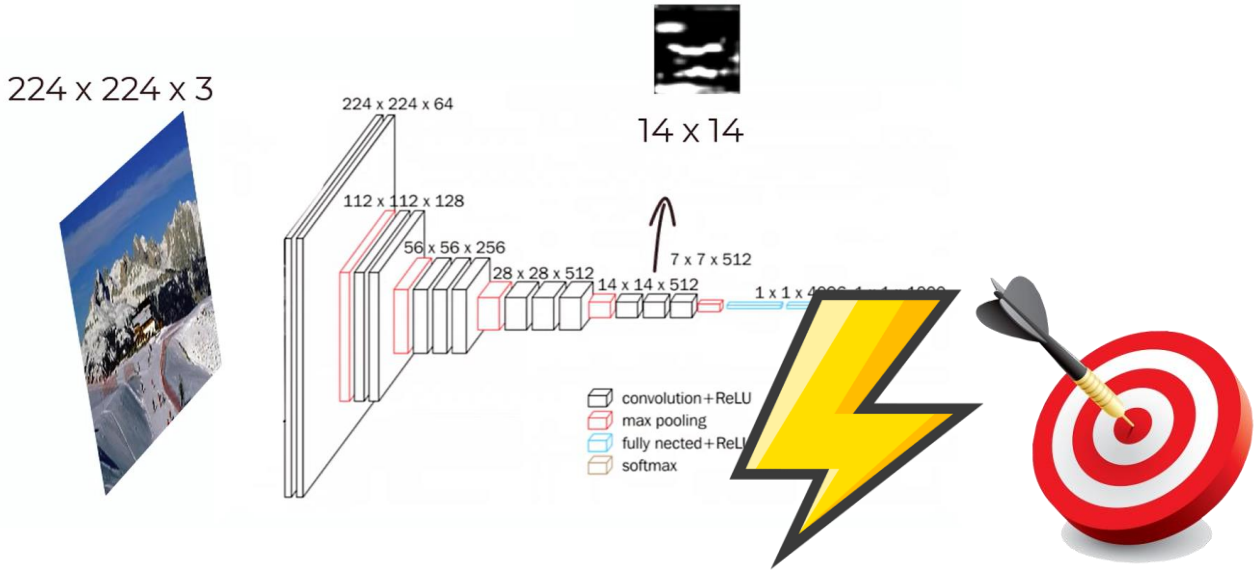
Analyze Conv2D layers

First & second layer filters:

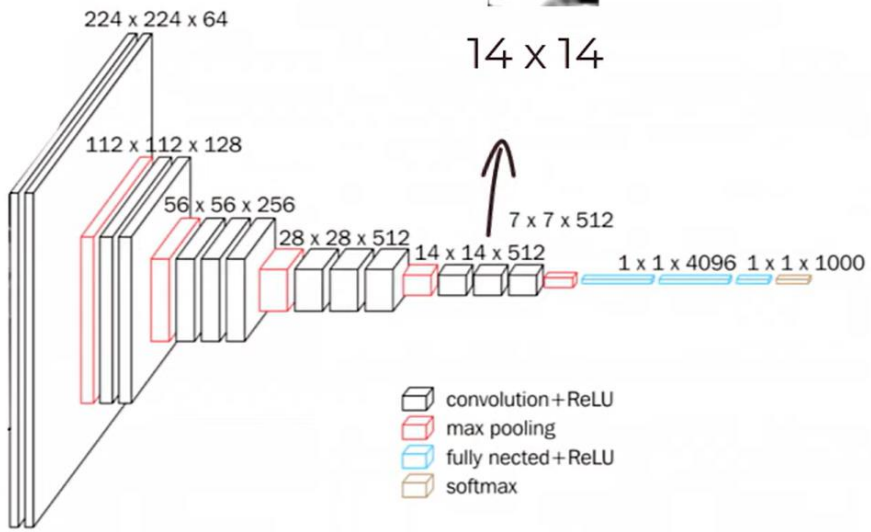
-> If high-frequency features appear, try „Yin-Yang principle”



Final layers:
Get any filter <u>output</u>, scale it up (Bilinear Interpolation), apply to image



224 x 224 x 3



14 x 14

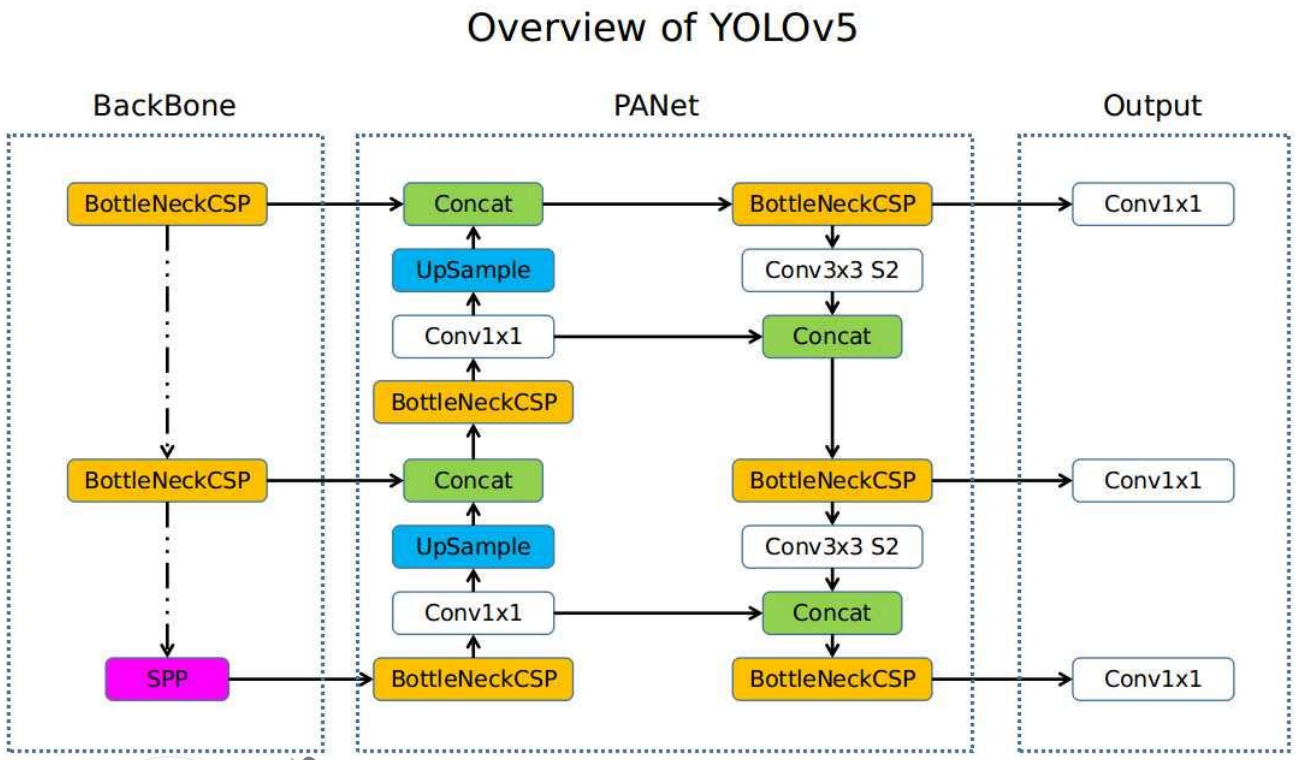
224 x 224

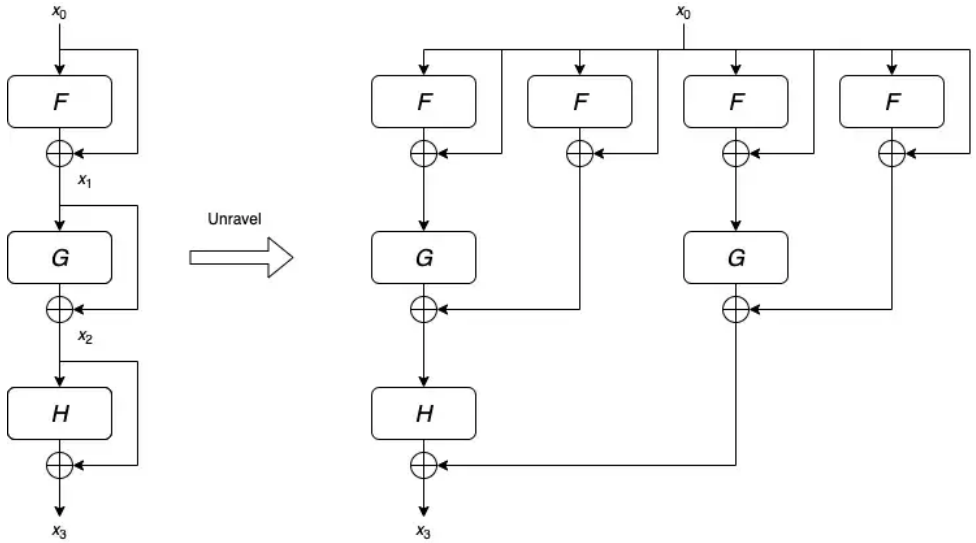
224 x 224 x 3

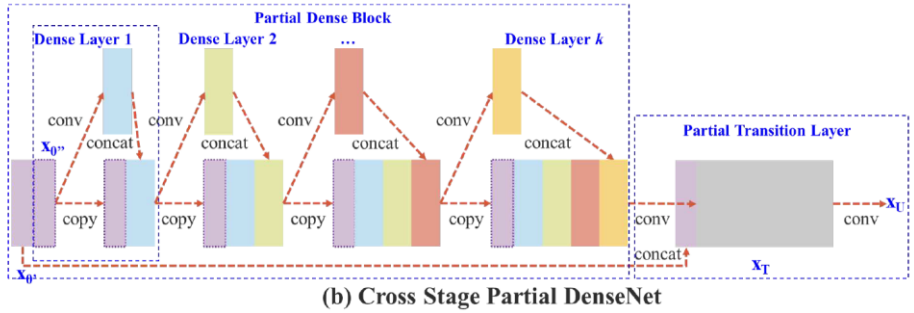
„Snow detector”

224 x 224 x 3

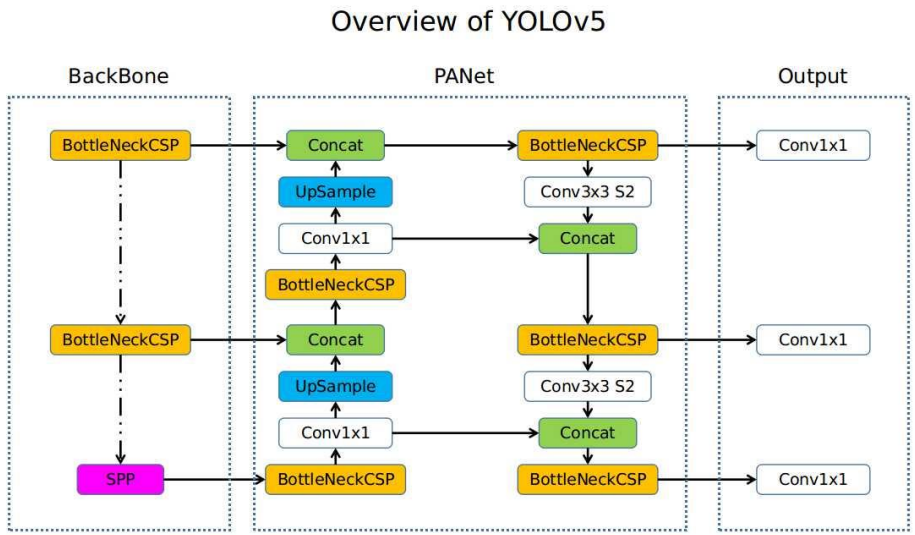
#9 The power of skip – connections, residual blocks and multiple outputs – on YOLO v5

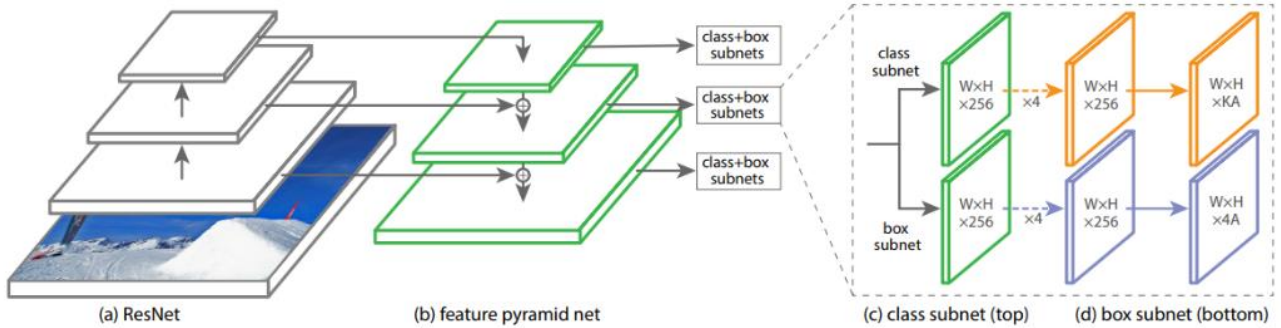


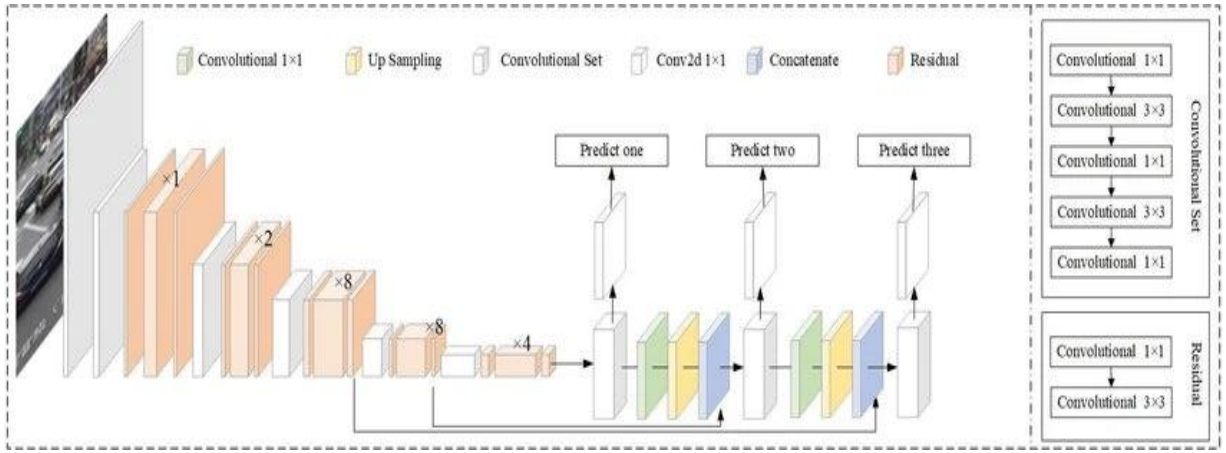


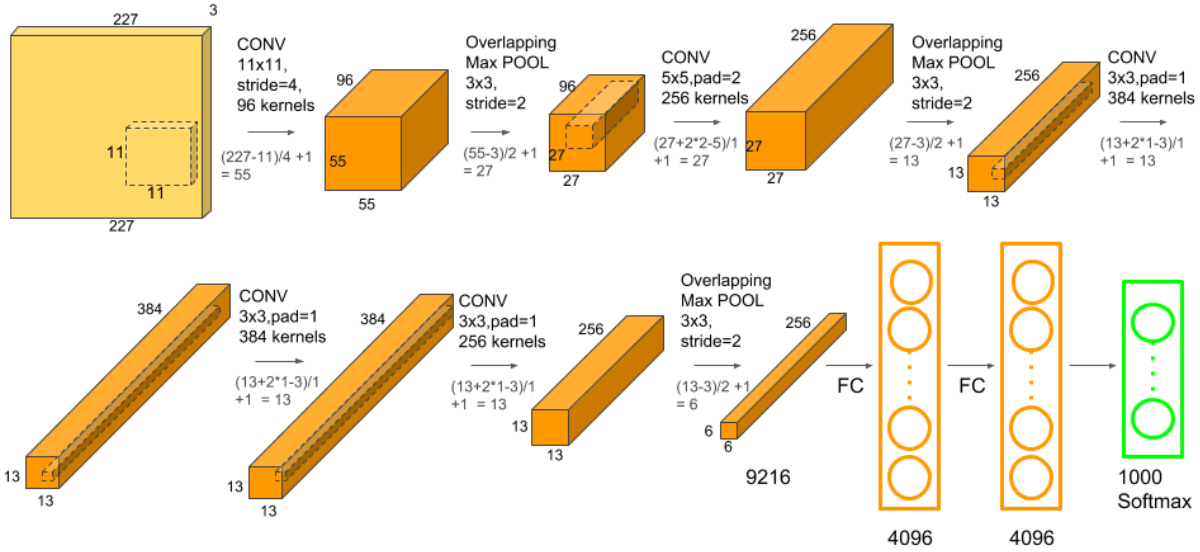


(b) Cross Stage Partial DenseNet


Overview of YOLOv5
BackBone
PANet
Output
BottleNeckCSP
BottleNeckCSP
SPP
Concat
UpSample
Conv1x1
BottleNeckCSP
Concat
UpSample
Conv1x1
BottleNeckCSP
BottleNeckCSP
Conv3x3 S2
Concat
BottleNeckCSP
Conv3x3 S2
Concat
BottleNeckCSP
Conv1x1
Conv1x1
Conv1x1


class+box subnets
class+box subnets
class+box subnets
class subnet
box subnet
W×H ×256
×4
W×H ×KA
W×H ×4A
(a) ResNet
(b) feature pyramid net
(c) class subnet (top)
(d) box subnet (bottom)


Convolutional 1×1
Up Sampling
Convolutional Set
Conv2d 1×1
Concatenate
Residual
×1
×2
×8
×8
×4
Predict one
Predict two
Predict three
Convolutional 1×1
Convolutional 3×3
Convolutional 1×1
Convolutional 3×3
Convolutional 1×1
Convolutional Set
Convolutional 1×1
Convolutional 3×3
Residual


227
227
3
11
11
CONV 11x11, stride=4, 96 kernels
(227-11)/4 +1 = 55
96
55
55
Overlapping Max POOL 3x3, stride=2
(55-3)/2 +1 = 27
96
27
27
CONV 5x5,pad=2 256 kernels
(27+2*2-5)/1 +1 = 27
256
27
27
Overlapping Max POOL 3x3, stride=2
(27-3)/2 +1 = 13
256
13
13
CONV 3x3,pad=1 384 kernels
(13+2*1-3)/1 +1 = 13
384
13
13
CONV 3x3,pad=1 384 kernels
(13+2*1-3)/1 +1 = 13
384
13
13
CONV 3x3,pad=1 256 kernels
(13+2*1-3)/1 +1 = 13
256
13
13
Overlapping Max POOL 3x3, stride=2
(13-3)/2 +1 = 6
256
6
6
9216
FC
4096
FC
4096
1000
Softmax