بسم الله الرحمن الرحیم

How to Design Efficient Deep Convolutional Architectures

SEYYED HOSSEIN HASANPOUR

FALL 1396

HAMIM©

Contents

- Design Choices
  - Major Architecture trends
    - LeNet
    - AlexNet
    - NIN
    - VGG
    - GoogleNet-INCEPTION/Batch-Normalization
    - ResNet(ResBlock)
    - Wide Residual Net
    - DenseNet
    - SqueezeNet
    - ALLCNN
    - Detailed discussion
      - Strided convolution vs pooling
      - Over-lapped pooling vs non-overlapped pooling
      - Enhancing pooling
      - Dropout utilization tips
      - etc
- Knowledge distillation

LeNet5

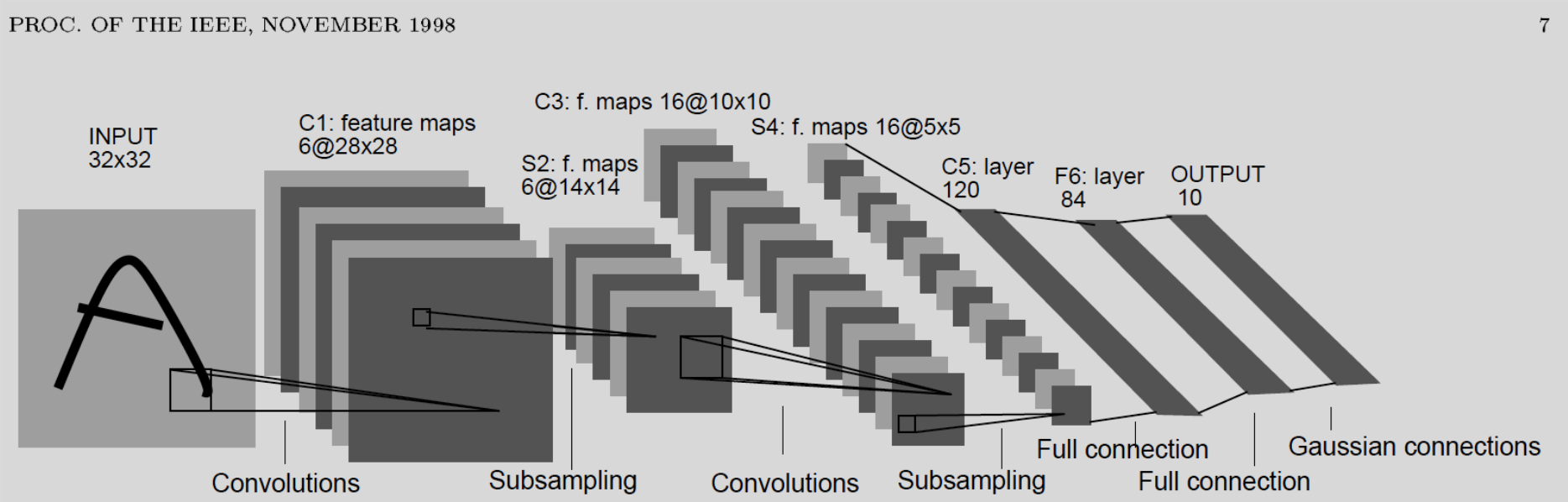
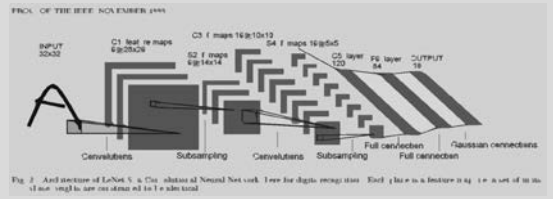


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeNet5

- The First introduction to CNN . A 7 Layered network. (3 Convs, 2 subsampling, and then FC!)
- The idea was a pipeline with two parts, one a feature leaner/extractor – the other an MLP
  - What made him use CNN?
    - Input structure matters , in MLP it is completely ignored!
    - Correlation matters and can result in extraction of good local features
    - Invariance is important especially for images , and MLP although can learn invariant features, they need more weights and thus a lot more data. In order to cope with invariance, weights at different locations need to exist and look for a specific feature! CNN solves all of these by weight sharing!
- 2x2 non-overlapping Average pooling was used /the notion of down-sampling
- Tanh was used
- 5x5 kernels were used
- Normalizing the input! (zero mean, variance 1 !)
  - Why? Because it accelerates learning!

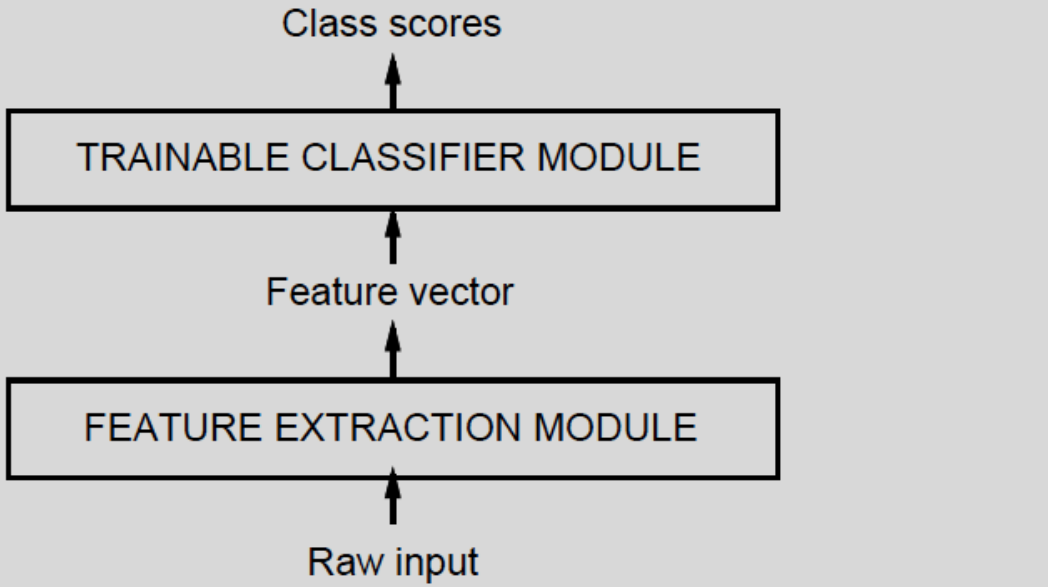
  -And also avoid saturating the sigmoid or Tanh activation functions.



- Nonlinearity was applied after pooling
- The notion of pyramid shaped architecture, increase number of feature-maps and shrink spatial dimensions.
  - why? In order to achieve invariance
- When a feature is detected , the exact spatial location doesn't matter
- Bigger (and deeper) networks work better (on larger dataset). LeNet1 to LeNet4 to LeNet5

  Artificially increase the dataset by using distortions!
- By increasing the dataset, NNs advantage will become more striking than other methods.

LeNet5

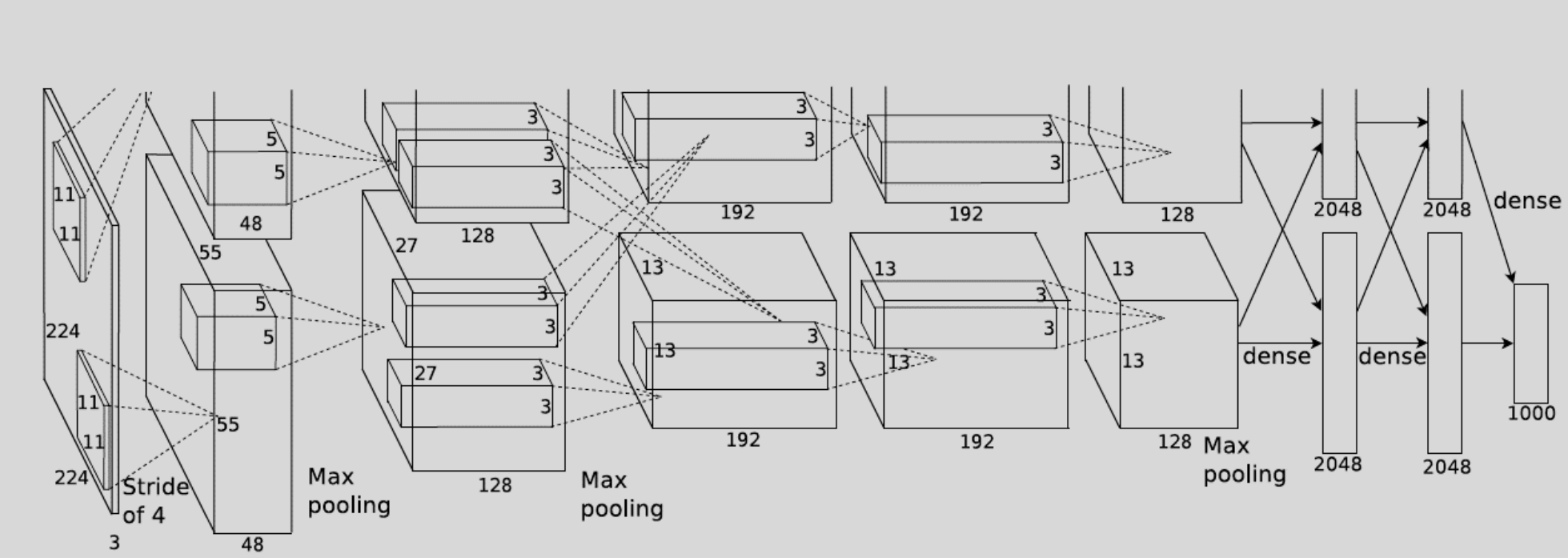


Classic pipeline which is also used in early architecture designs such as LeNet

LeNet5 Highlights/takeaway

- Use Tanh, its better than sigmoid
- Use normalization, it speeds up training and avoids saturation to some extend
- Use non-overlapping 2x2 average pooling! Just down-sample, pooling itself is not important!
- Larger network works better but also wants larger dataset
- Create larger dataset with distortions(synthesized data!)
- Use 5x5 filters
- Create network to have a pyramid form, shrink spatial dims while increasing number of feature-maps
- Keeping and utilizing local correlation matters !
- Have Conv plus some FC at the end!

AlexNet

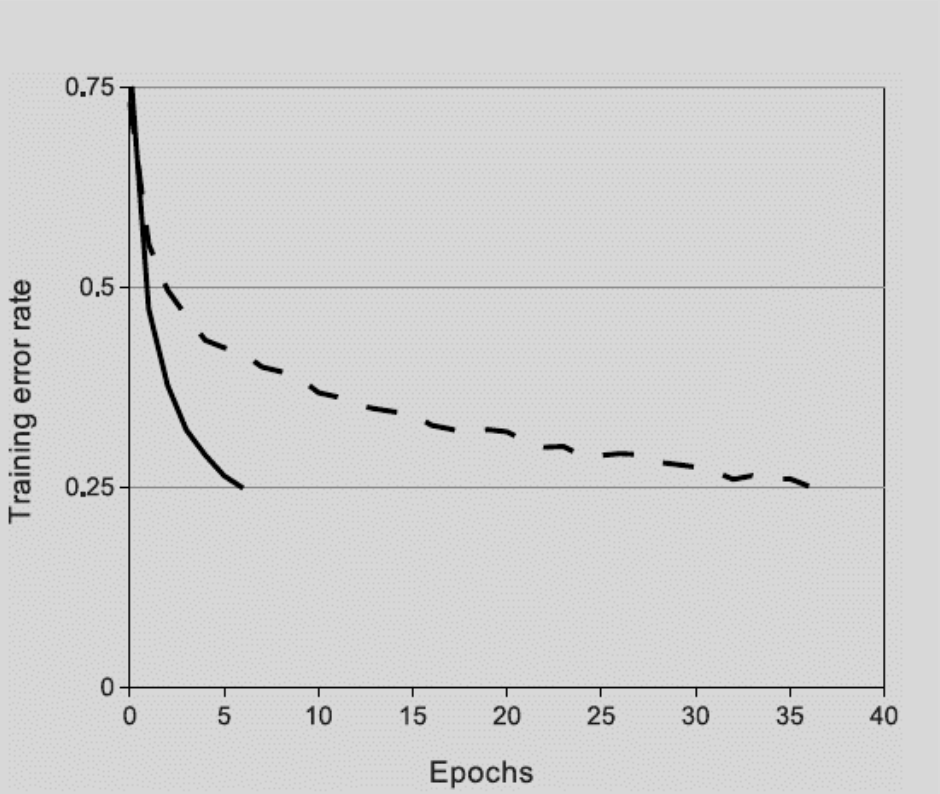


AlexNet Architecture

AlexNet

- Based on Lenet5
- 8 layer, follows the same intuition, a pipeline with two parts
- Uses ReLU for the first time (in a CNN)
  - Why? The train several times faster!
  - ReLUs have the desirable property that they do not require input normalization to prevent them from saturating
- Uses Maxpooling – over lapped pooling! :
  - Better performance (maybe better initial state only)
    noticed its harder to overfit! (observed during training ,models with overlapping pooling find it slightly more difficult to overfit.)
- 60M parameter, overfits so suggests:
  - Uses dropout(doubles the training time for convergence)
    weight decay and Augmentation
- Arbitrary kernel sizes , 11x11 and 5x5 kernels were used
  - Depth is really important for achieving their results! removing any conv layer, results in a loss of 2% in top-1 performance!
- Improved results by using larger(wider) network! And training longer!
  - Why larger and not deeper ?
- Weird architecture, groups are divided
- Introduces a new normalization layer called Local contrast normalization
  - Improves generalization (they claim!)
- Suggests some preprocessing that can help (talk about these later)

AlexNet

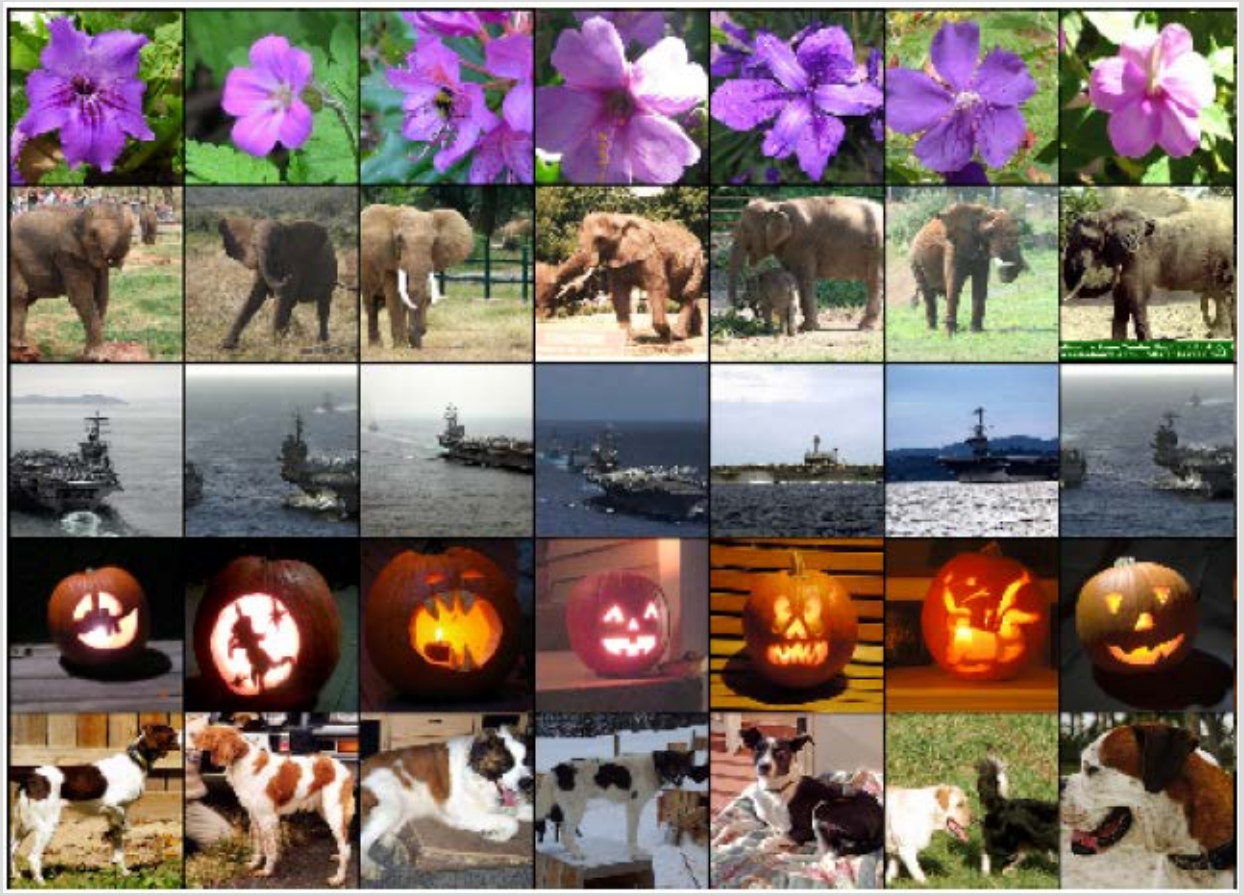


ReLU decreases training time several times!
Picture : A four-layer convolutional neural network with **ReLUs** (solid line) reaches a 25% training error rate on CIFAR-10 **SIX** times faster than an equivalent network with **Tanh** neurons(dashed line).

AlexNet

- Probing Networks visual knowledge :
  Nice visualizations which give intuitive understanding of what is happening
  - Features at last layers contain abstract high level information
    by comparing them, we notice similar identities, have small Euclidean distance!
  - Comparing all of those features is inefficient, instead use an autoencoder and compress the representation, use that instead

AlexNet

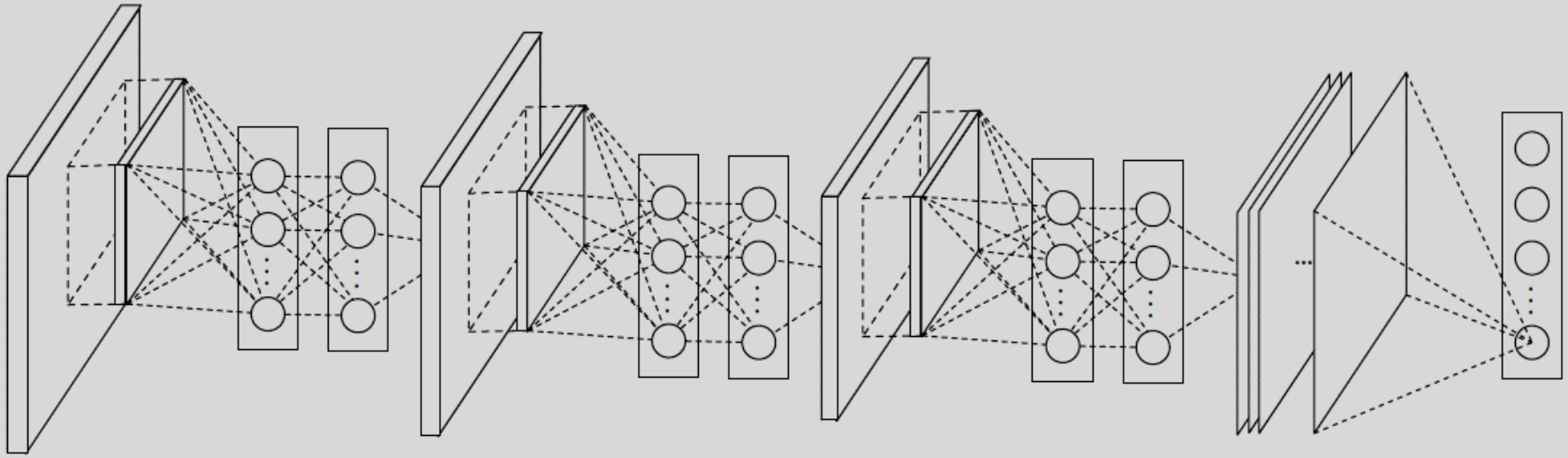


Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image

AlexNet-Highlights/takeaway

- Larger network
- Deeper network, depth matters!
- ReLU nonlinearity
- Overlapping Maxpooling (as apposed to non-overlapping average pooling in LeNet)
- Dropout and weight decay
- Several CNN and several FC layers to achieve highlevel abstract features
- Use Local Contrast Normalization
- Use 5x5 kernels , also use bigger ones if needs be !
- Uses Gaussian for initialization with std: 0.01 for Conv and std: 0.005 for FC layers
- If image is large, use larger kernels

Network in Network

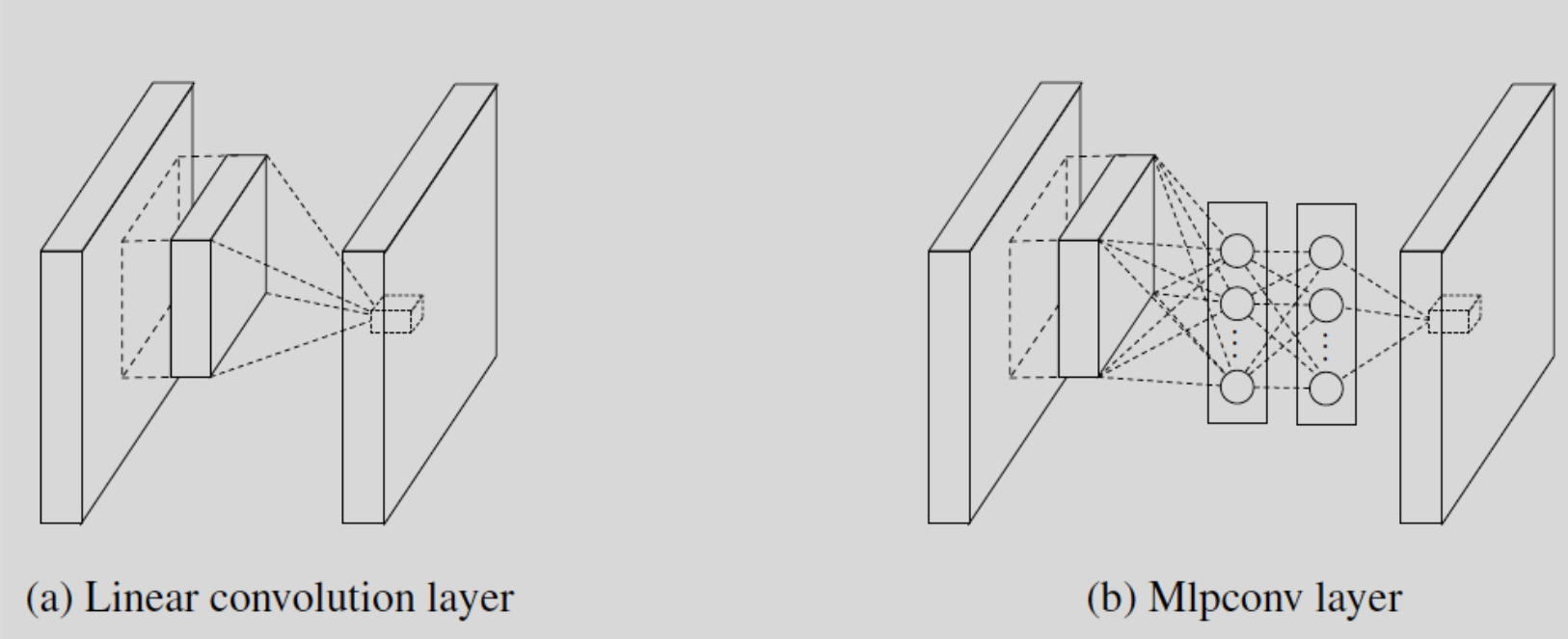


The overall structure of Network In Network.

Network in Network

- Several unique intuitions and proposals are introduced
- GLM
  - Conv as a Generalized Linear Model (GLM) for underlying local patches
  - Problem? Linear nature, needs lots of feature-maps
  - Implicitly considers the latent concepts linearly separable
- MLPConv
  - MLP imposes a lot of parameters, not good
  - Network In Network: using 1x1 convolution inside a network
  - MLP – FC – Conv 1x1 (FC approximates MLP, and 1x1 Conv approximates FC)
  - Nonlinear nature, more efficient
- Proposes to use Global pooling!
- Proposes the FCN or Fully Convolutional Network
  - May have different use cases, such as a handy tool in providing meta informations (semi)automatically
- Does ZCA preprocessing , uses dropout!
- Starts with large learning rate, decreases manually when loss plateaus

Network in Network



Comparison of linear convolution layer and mlpconv layer. The linear convolution layer includes a linear filter while the mlpconv layer includes a micro network (we choose the multilayer perceptron in this paper). Both layers map the local receptive field to a confidence value of the latent concept.

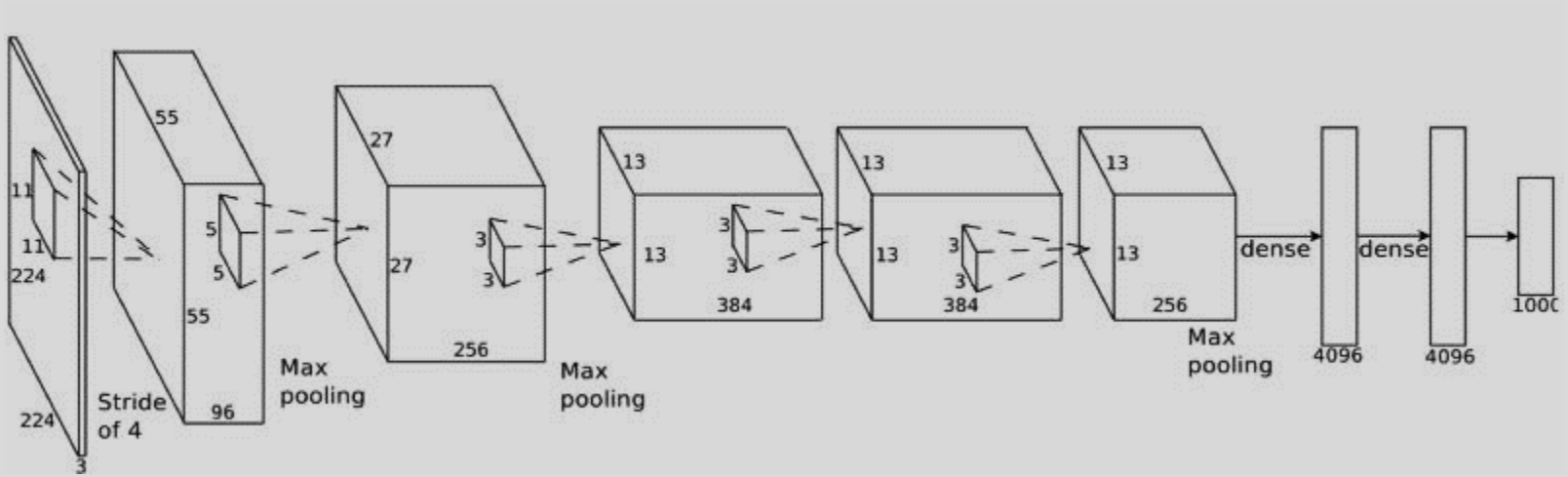
Network in Network Highlights/takeaway

- Ignores the old established extractor-classifier pipeline, Introduces NIN
- Says :use Global Average pooling instead of FC layers, less overfit, better generalization
- Uses 5x5, 3x3 and 1x1 filters
- ReLU nonlinearity
- Overlapping 3x2 Maxpooling and average pooling (kernel 3x3, stride of 2x2)
- Dropout and weight decay
- No usage of Local Contrast Normalization
- Uses Normalization of Input and ZCA whitening
- Uses Gaussian for initialization with std: 0.05
- Doesn’t follow the pyramid form unlike its predecessors .
- Replaced the MLP from the end of the network and used it inside the network!
- Note!
  - Did not achieve good results on ImageNet, why? We investigate this later in the course
  - TLDR: its because of not adhering to correlation preservation)

VGGNET

- One of the most beautifully devised, well performing yet inefficient architectures
- Widely used
- The intuition is basically the same architecture wise, but introduces some new concepts
- Improvements ?
  - smaller kernels and stride at first layer caused the improvements(by overfeat)
  - Multi-scale training and dense evaluation introduced improvements
- Their main contribution ?
  - 3x3 kernels , the notion dynamic receptive field, 2 3x3 provide the effective receptive field of a 5x5 , 3 of such provide that of a 7x7!
- Non overlapping Maxpooling 2x2, stride of 2.
- Does not use local contrast normalization
- Several architectures ranging from 11 to 19 layers, (8 conv to 16 with 3FC layers) with (almost) the same number of parameters! 133M to 144M
- Uses ReLU, uses 1x1 (as a linear transformation in channels) but reports unsatisfactory results.
- Deeper is better! Network depth!
- Larger is better ! Inputs!
- Used pre-initialization of some layers!
- Used multi scale training
- Convert to Fully convolutional network for testing!
- Using Xavier later which let them not to bother pre-initalization !
- Single-scale training/Multi-scale training

What’s different?



What’s different?

Improvements through the years:

- Smaller receptive field (3x3)
- Training densely over the whole image and over different scales

VGG Configuration Overview

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Number of parameters: 3x3 vs larger kennels

Assuming that both the input and the output of a three-layer 3×3 convolution stack has C channels, the stack is parameterized by: $3(3^2C^2) = 29C^2$ weights.

A single 7 × 7 conv. layer would require $7^2C^2 = 49C^2$ parameters.(81% more)

More convolutions stacked

- We can get the same effective receptive field of 7x7 with 3 stacked 3x3 conv. Layers (with ReLU between each) with less parameters.
- We gain more non-linearity of the decision function, while not affecting the receptive field of the convolution layers.

How to train such deep network?

The initialization of the network weights is a big factor, since we might get stuck in a local minima.

How to train such deep network?

The initialization of the network weights is a big factor, since we might get stuck in a local minima.

Solution: beginning with a “shallow” ConvNet, train with random initialization. When training deeper networks, initialize first four conv. layers and last 3 fully-connected layers with the already trained parameters from the shallow network.

How to train such deep network?

“Training image size. Let S be the smallest side of an isotopically-rescaled training image, from which the ConvNet input is cropped. While the crop size is fixed to 224 × 224, in principle S can take on any value not less than 224: for S = 224 the crop will capture whole-image statistics, completely spanning the smallest side of a training image; for S≫ 224 the crop will correspond to a small part of the image, containing a small object or an object part.”

Setting the training scale S

There are two different approaches:

1. Single-scale training (S is fixed)
2. Multi-scale training (S is randomly chosen from a range)

Single Scale Evaluation

Run tests with image rescaled to size Q where:

- $Q = S$ for ConvNets trained for Single-Scale.
- $Q = 0.5(S_{min} + S_{max})$ for Multi-Scale.

Single Scale Evaluation

- Using LRN did not improve the results compared with the same configuration without LRN.
- The deeper the ConvNet the smaller the error was.
- Multi-scale training gave much better results.

Single Scale Evaluation

Table 3: **ConvNet performance at a single test scale.**

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	**25.5**	**8.0**

Multi-Scale Evaluation

We will now look at results when you test over different Q sizes of the same network:

- The test is run over several rescaled versions of the test image.
- The results are then averaged to get the classification.

Note: Because a large discrepancy between training and testing scales leads to a drop in performance, the test scales were close to the training one.

Multi-Scale Evaluation

Table 4: **ConvNet performance at multiple test scales.**

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
B	256	224,256,288	28.2	9.6
C	256	224,256,288	27.7	9.2
	384	352,384,416	27.8	9.2
	[256; 512]	256,384,512	26.3	8.2
D	256	224,256,288	26.6	8.6
	384	352,384,416	26.5	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**
E	256	224,256,288	26.9	8.7
	384	352,384,416	26.7	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**

Dense vs multi-crop evaluation

- Dense evaluation: In dense evaluation, the fully connected layers are converted to convolutional layers at test time, and the uncropped image is passed through the fully convolutional net to get dense class scores. Scores are averaged for the uncropped image and its flip to obtain the final fixed-width class posteriors.

- Multi-Crop: taking multiple crops of the test image and averaging scores obtained by

  passing each of these through the ConvNet.

Dense vs multi-crop evaluation

ConvNet config. (Table 1)	Evaluation method	top-1 val. error (%)	top-5 val. error (%)
D	dense	24.8	7.5
	multi-crop	24.6	7.5
	multi-crop & dense	**24.4**	**7.2**
E	dense	24.8	7.5
	multi-crop	24.6	7.4
	multi-crop & dense	**24.4**	**7.1**

VGGNET highlights/takeaways

- 3x3 every where, don’t use 5x5 or 7x7 or larger! (e.g. 5x5 results in inferior performance!)
- Non-overlapping pooling 2x2!
- ReLU
- Use pre-initialization or yet even better go with Xavier!
- Multi Scale, Multi Crop, Dense evaluations
- Deeper is better
- Smaller kernels better than larger ones, but spatial correlation matters as well (don’t use 1x1)!
- Convert Fully connected layers to convolutional one at test time!

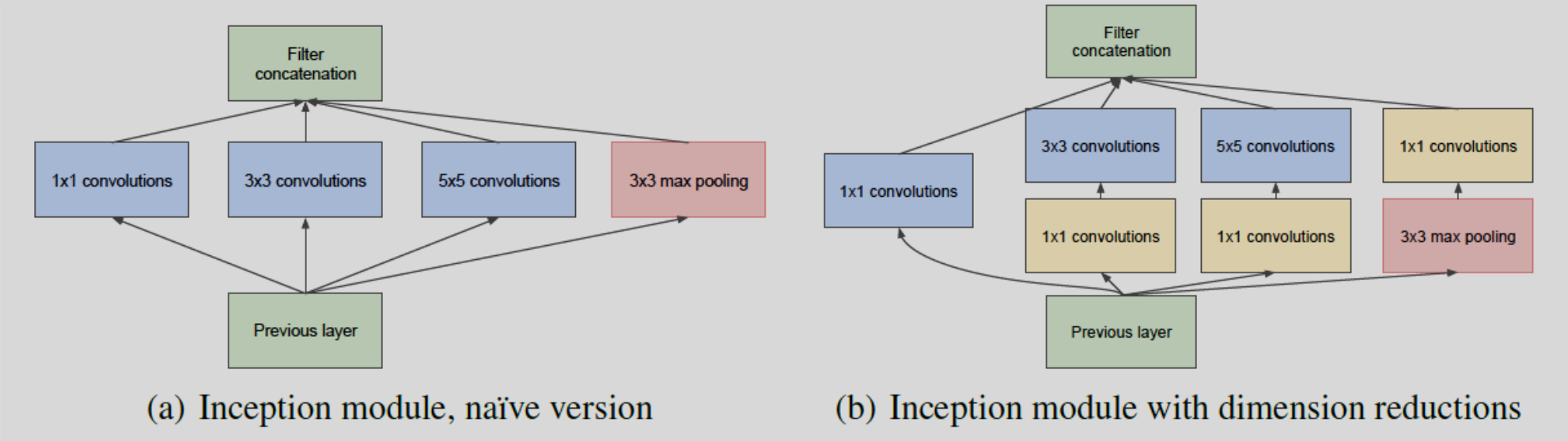
GoogleNet_V1

- Inspired by NIN and other former works, culmination of NIN
- Classic uniform designs are inefficient
  - Uniformly increasing, leads to over-fitting as the net is further enlarged
  - Dramatically increases the computational costs. (For example, in a deep network, if two conv layers are chained, any uniform increase in the number of their filters results in a quadratic increase of computation. If the added capacity is used inefficiently (for example, if most weights end up to be close to zero), then a lot of computation is wasted.
- Solution to solve the uniform issue? Moving from fully connected to sparsely connected even in CNN!
  - There are issues in that regard
    - 1.the current infrastructures are not efficient for non uniform sparse data structures , the overhead is huge!
    - 2.the infrastructure and libraries for dense matrix calculations are very efficient and this widens the gap further! The uniformity of the structure and large number of filters and greater batch-sizes, allow for efficient dense computation, that's why this trend is string. (formerly in LeNet they used sparse connections by AlexNet changed the trend again because of the former reason)
  - Hard for fine-tuning

GoogleNet_V1

- Introduces Inception v1
  - Global Average pooling but with a linear fc layer at the end for easier fine-tuning on other datasets.
    - Improvements achieved by using global avg (0.6%)
  - Uses mean subtraction
  - Training is hard, Uses axillary classifiers!
    - to enhance gradient flow (used a weight of 0.3)
- Drastically reduces parameters but increases FLOPS (limited themselves to 1.5B FLOPS for practical scenarios)
- Still well performing, but
  - not uniform
  - And?

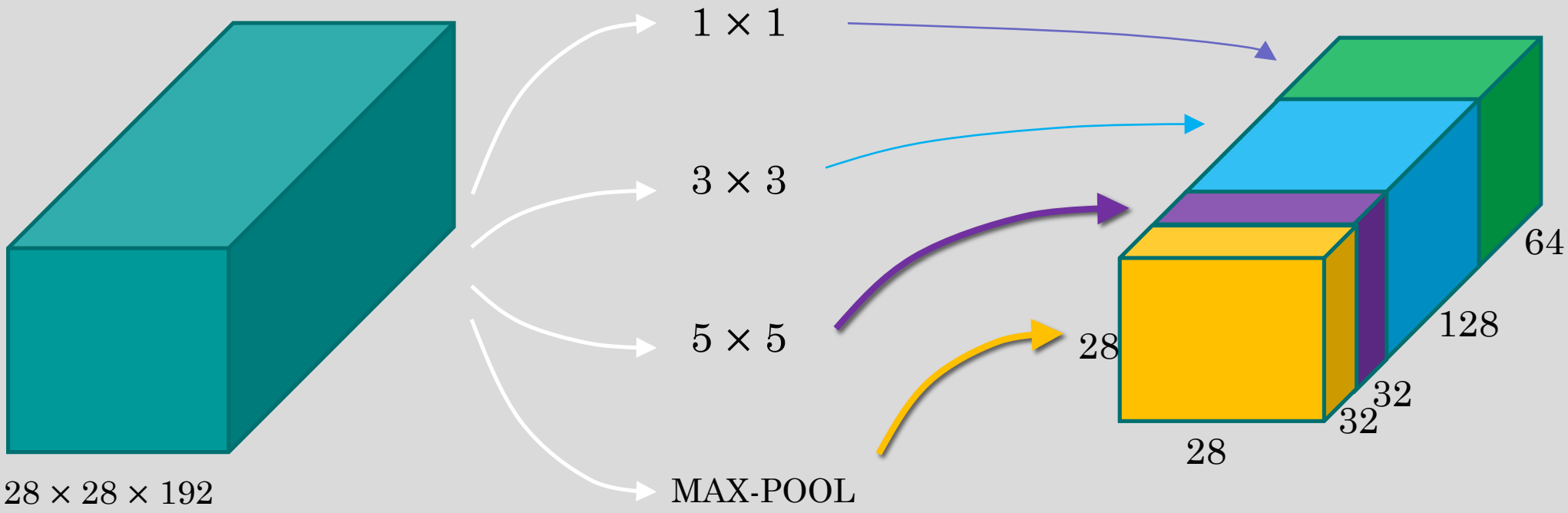
GoogleNet_V1



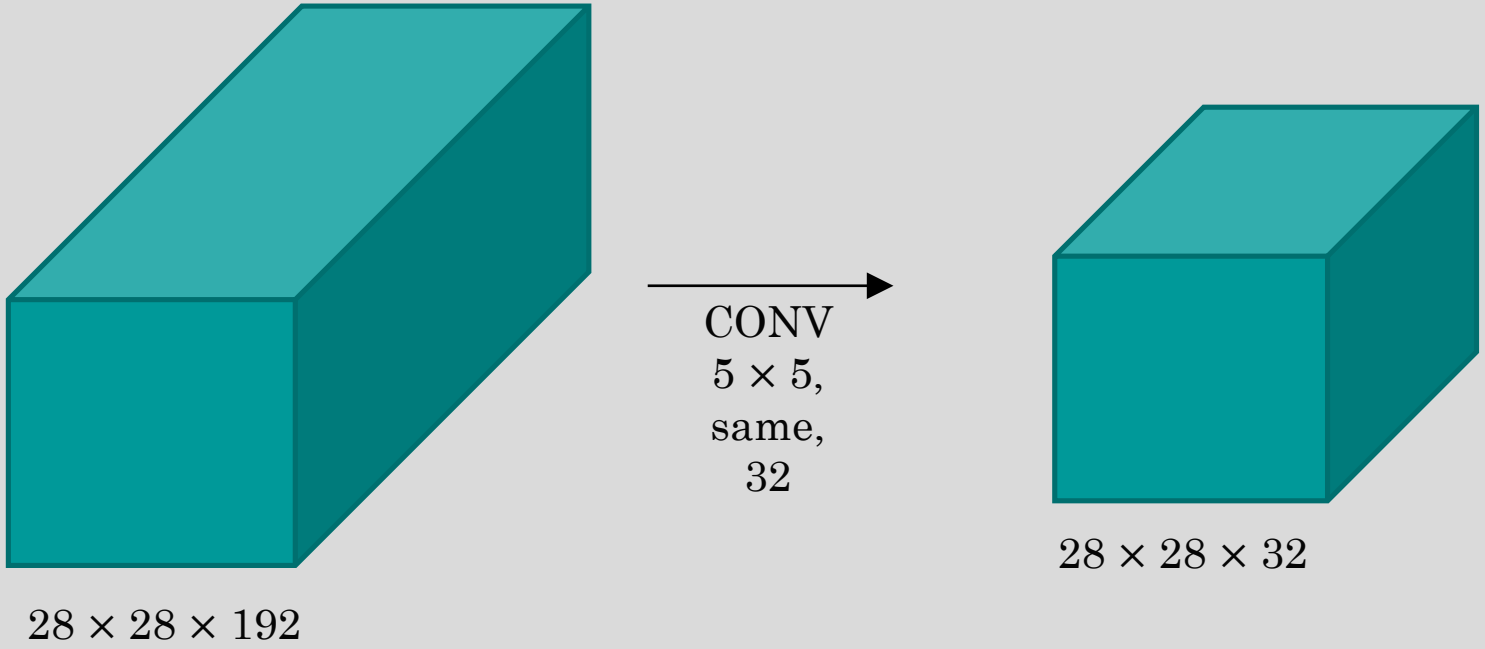
(a) Inception module, naïve version

(b) Inception module with dimension reductions

GoogleNet_V1

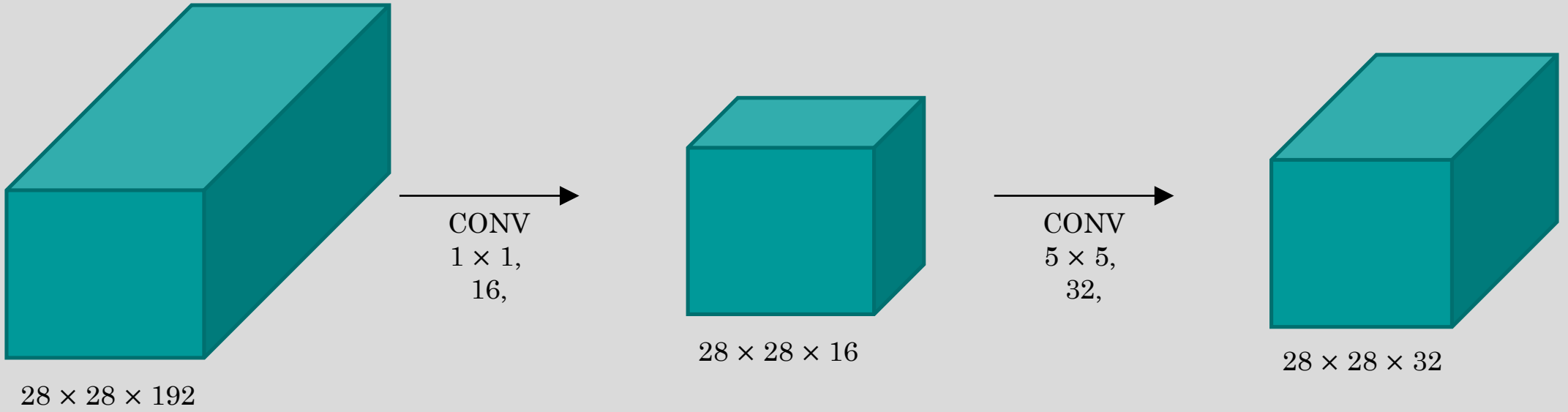


GoogleNet_V1



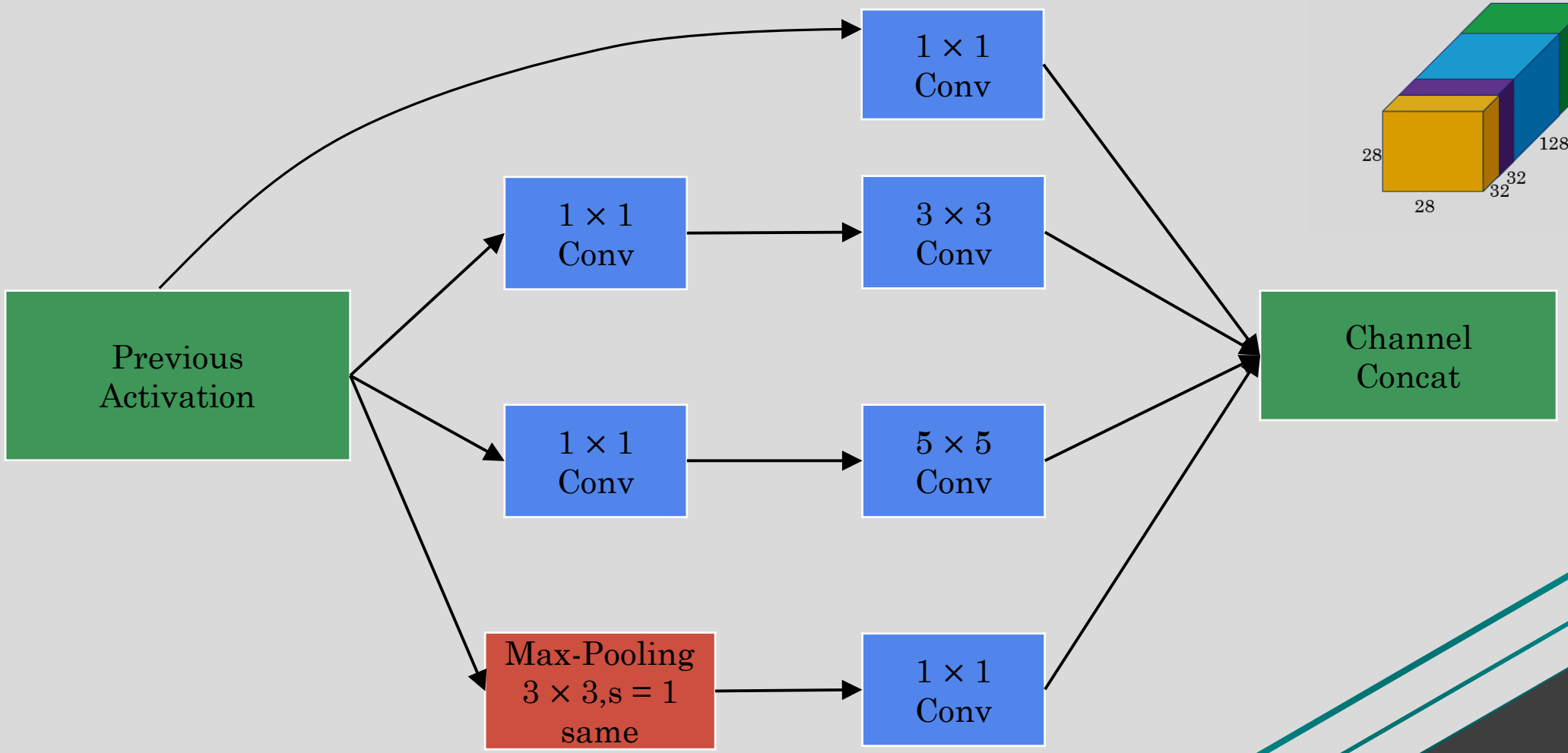
- (5x5x192)x32 = **153,600** parameters
- (5x5x192)x28x28x32 = **120**,422,400 = **120M** FLOPS

GoogleNet_V1

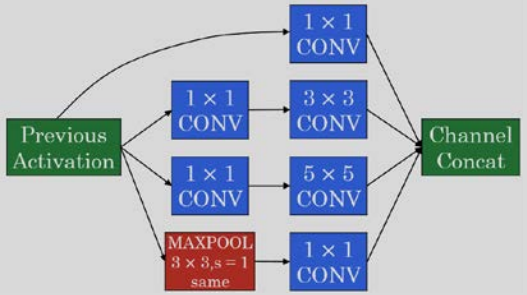


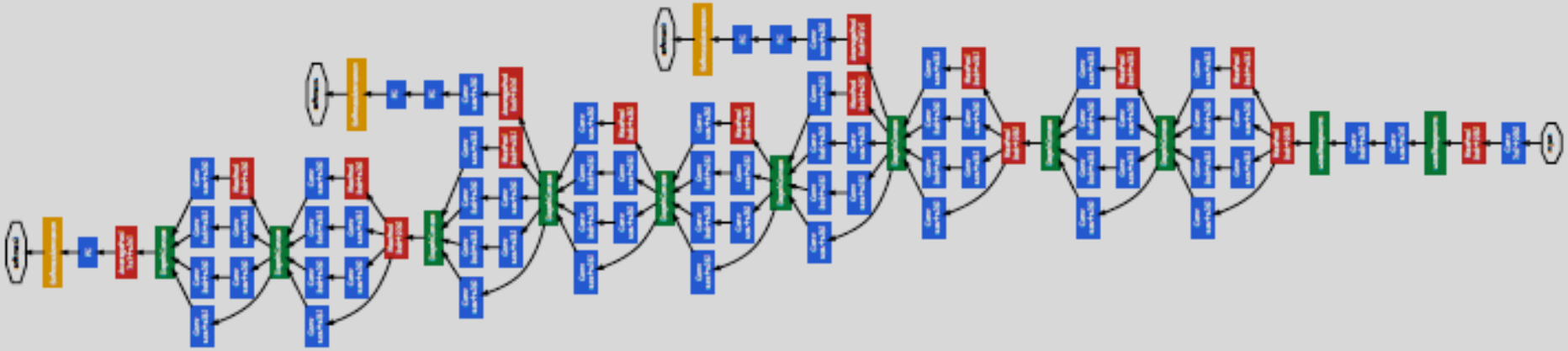
- (1x1x192)x16 = 3,072
  (5x5x16)x32 = 12,800
- 12800+3072 = **15,872** parameters
- (1x1x192)x28x28x16 = 2,408,448
- (5x5x16)x28x28x32 = 10,035,200
- 10035200+2408448 = **12**,443,648 = **12M** FLOPS

GoogleNet_V1



GoogleNet_V1





GoogleNet_V1

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Table 1: GoogLeNet incarnation of the Inception architecture

GoogleNet_V1 highlights/takeaway

- Sparsity matters, lets approximate that !
  - Use Inception module!
- Uniform design is inefficient ! It over-fits when enlarged and has computation overhead!
  - Use 1x1 to decrease computation
  - Bottleneck concept (shrink representation down to a small quantity and then increase it)
- Uses mean subtraction
- Uses multi crop
- Uses overlapping pooling
- Uses Average Global Pooling

GoogleNet_V2/BatchNorm

- Introduces BatchNorm, claims no need for dropout! Also the idea is something else, but it has a regularization effect!
- 30% overhead
- Allows for higher learning rate
- Enhances generalization because it doesn’t consider one sample but a batch of samples.
- What is BatchNormalization ?
  - Explain
- What is covariate shift ?
  - Internal covariate shift?
- What's special about normalizing normally?
  - We need identity so that if the network thinks that the un-normalized distribution is better , then so be it!
  - Gama and Beta are for achieving identity function!

GoogleNet_V2_BatchNorm

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

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

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

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

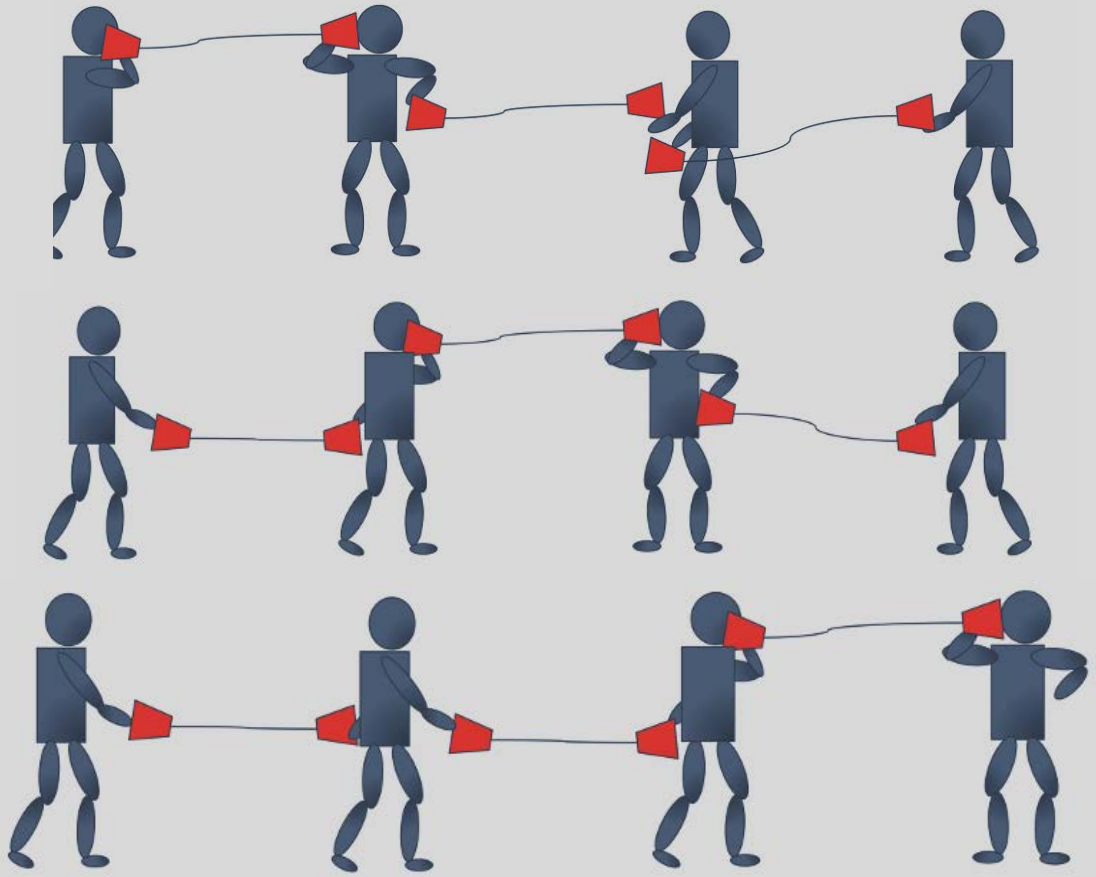
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$
$$\beta^{(k)} = \mathrm{E}[x^{(k)}].$$

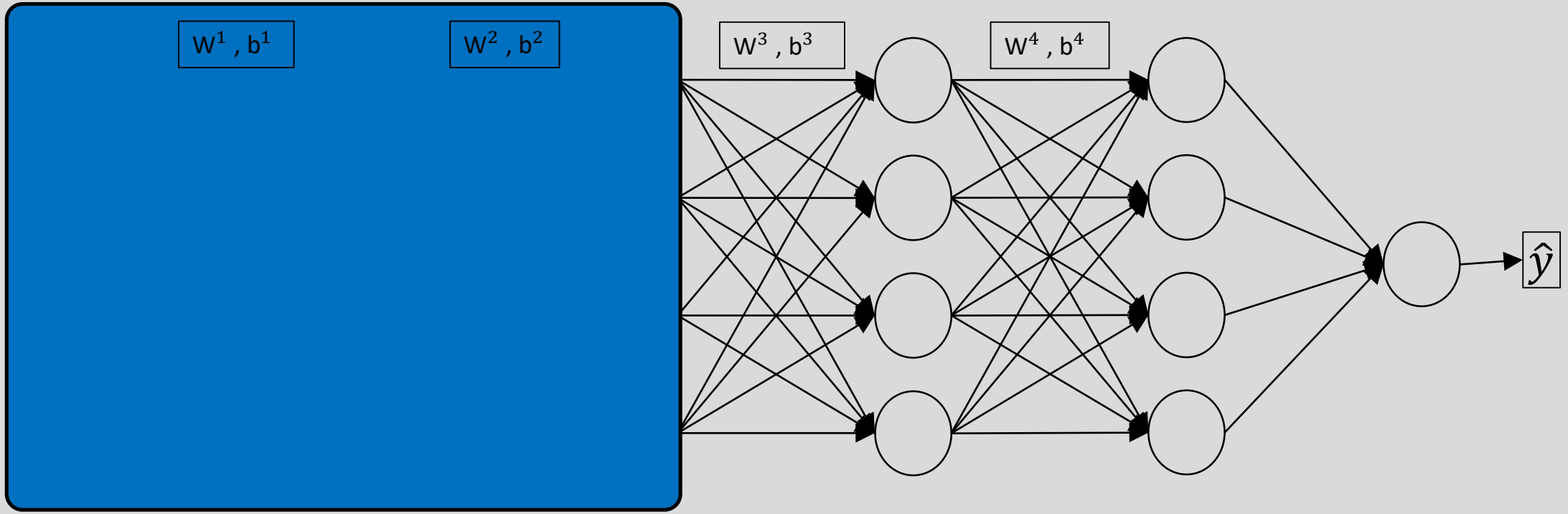
"What if the activations at some upper layers shouldn't be normalized?"

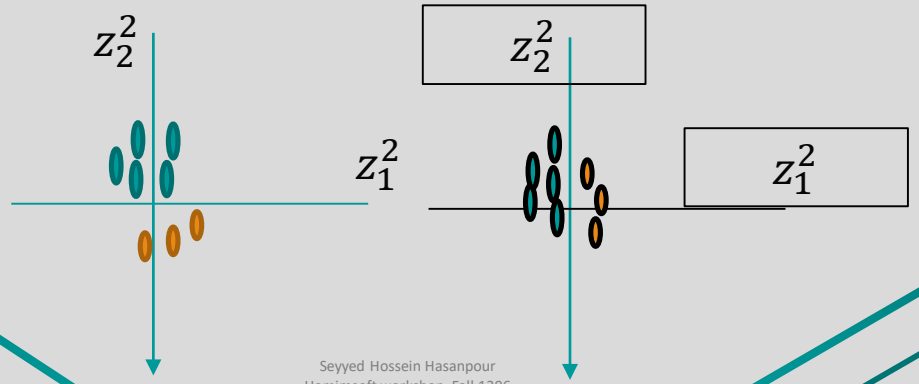
If gamma = $sqrt(var(x))$ and beta = $mean(x)$, the original activation is restored.

GoogleNet_V2_BatchNorm



Why this is a problem with neural networks?





GoogleNet_V2/BatchNorm

- The inception V2 is introduced
  - Changes : 5x5 replaced with 3x3 -> this resulted in 9 more layers in the network
  - Used separable convolution with depth 8 multiplier on the first layer, reduces computation overhead but increased memory consumption

BatchNormGoogleNet_V2 highlights/takeaway

- Use BatchNorm!
- Use less dropout and weight decay when BN is used
- Shuffle your data thoroughly , it affects BN performance and thus your models!
- Use 3x3 instead of 5x5!
- If an architecture did not use BatchNorm in first place, don’t use one in fine-tuning process.
- It enhances generalization because it takes into account several samples rather than one.
- It acts as a regularizer because it injects noise into the network parameters
  - Because of normalization the signal is changed
- The more diverse the samples in each batch, the better (shuffle more)
- Bigger batch sizes enhance the result of BatchNormalization because its a better approximation of the dataset
- It allows for higher learning rates
- Much less dependent on initialization values
- Saturating functions such as sigmoid and Tanh can now be easily used in deep arch without vanishing gradients or exploding gradients issues.

GoogleNet_V3

- What's the idea ?
  - How inception can be further tuned
  - Introducing some principles in designing better inception based architecture
  - The idea is still based on viewing the network design as a NIN paradigm
    - Everything is basically framed into a network and talked about

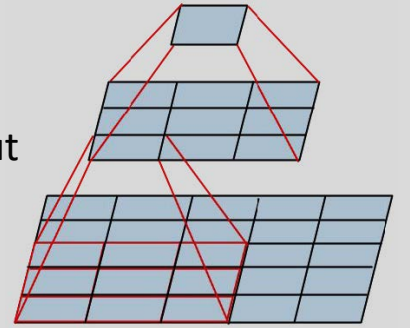
Principles

- Avoid representational bottleneck early in the network
- Avoid bottlenecks with extreme compression
  (because it destroys important correlation information in the input)
- Balanced depth and width

Factorizing Convolutions with Large Filter Size

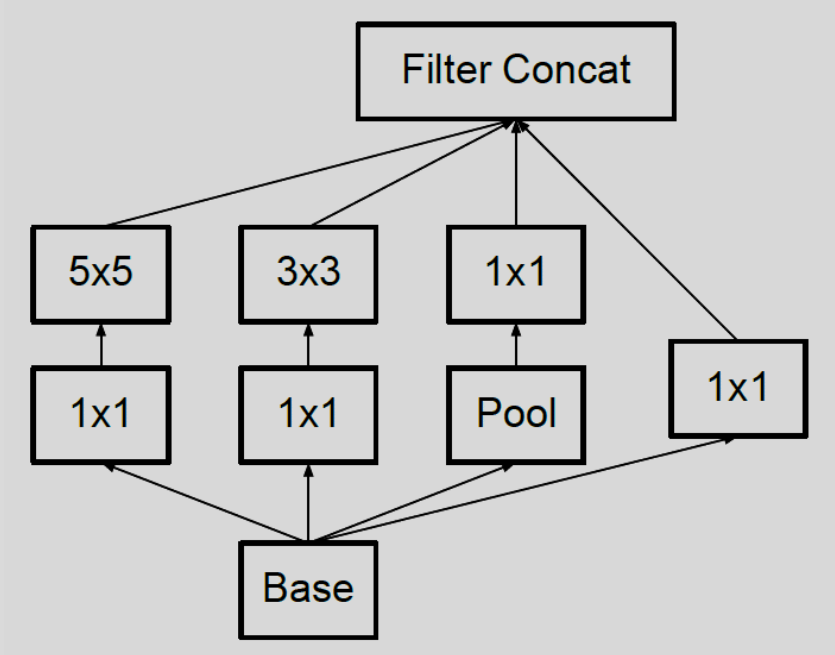
- Factorizing larger kernels into smaller ones
- Why factorization matters?
  - Inception is fully convolutional, each weight equals one mult, decreasing computational overhead, directly reduces #params
- 7x7 ,5x5 impose much more over head than 3x3
  - Ex: each 5x5 has 25/9 times more overhead

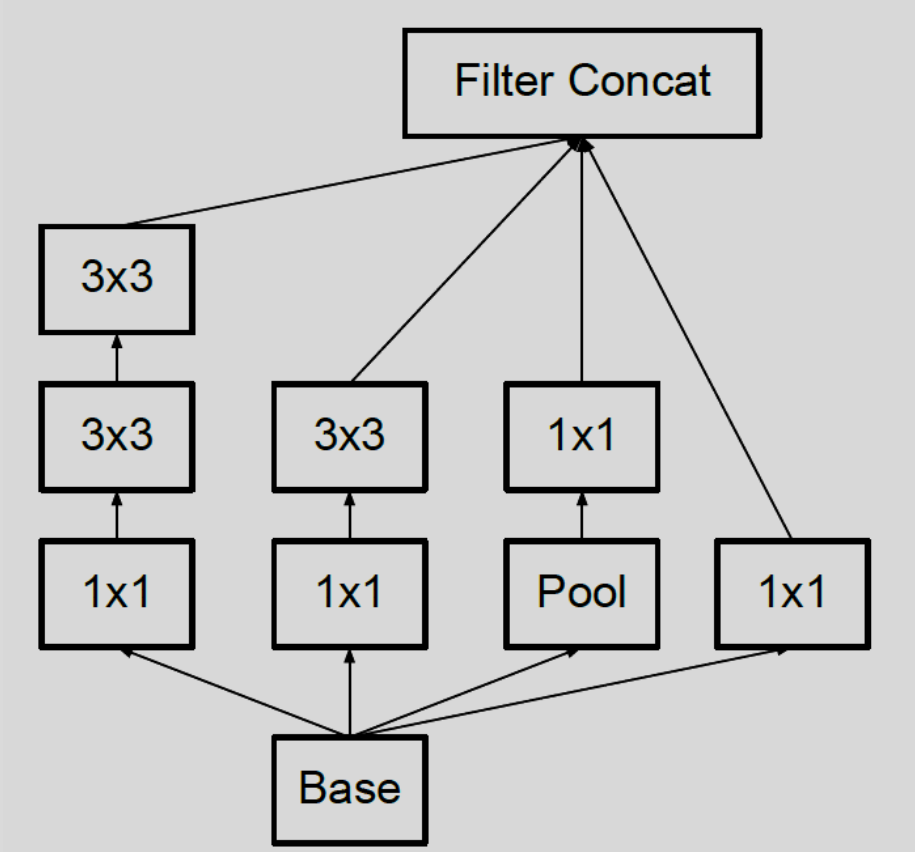
- Can we convert a 5x5 kernel into a mini network with the same input and output depth but less overhead?
  - Yes, each output looks as if it's a mini fc net sliding on a 5x5 area!
- Since invariance counts, lets use weight sharing!(Conv layer instead of FC)
  - Create a two layer network, the first layer 3x3, the 2nd one FC
  - Lets convert that, into two 3x3 kernel to replace 5x5



Mini-network replacing the 5×5 convolutions

Factorization

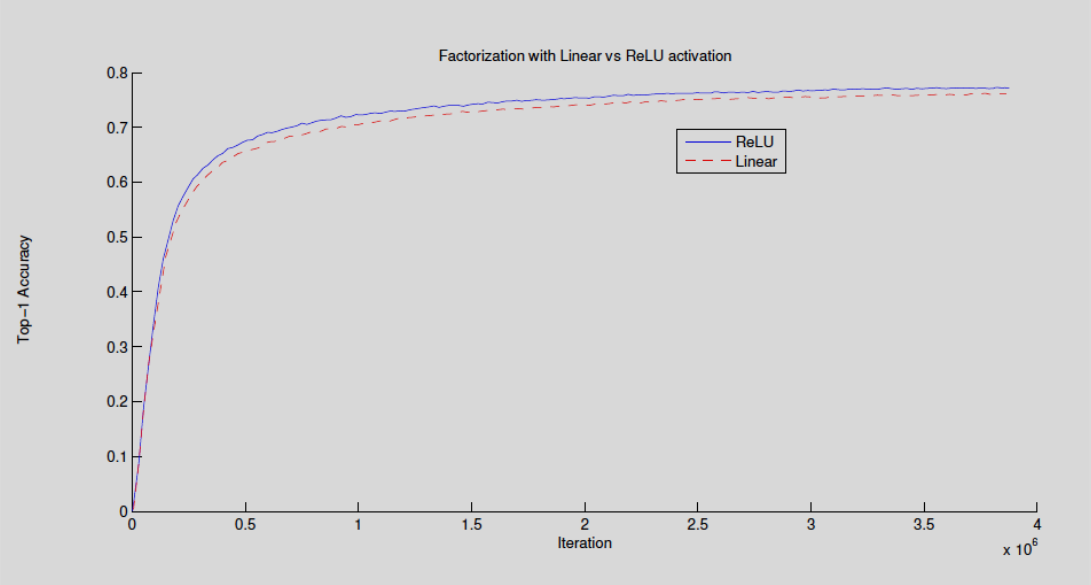




A 5x5 filter replaced by two 3x3 filters

Factorizing Convolutions with Large Filter Size

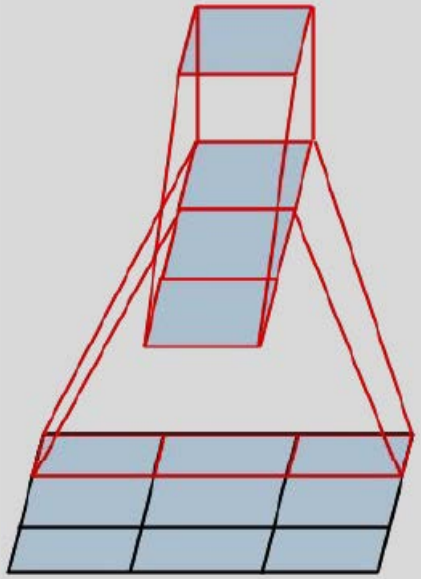
- Two Questions arise:
  - Does this result in any loss of expressiveness ?
    - Not sure, needs more experiments. as long as we use aggregation we are fine.
  - If our main idea was to factorize the linear part, shouldn't we keep linear activations?



Using linear activation was always inferior to using rectified linear units in all stages of the factorization.

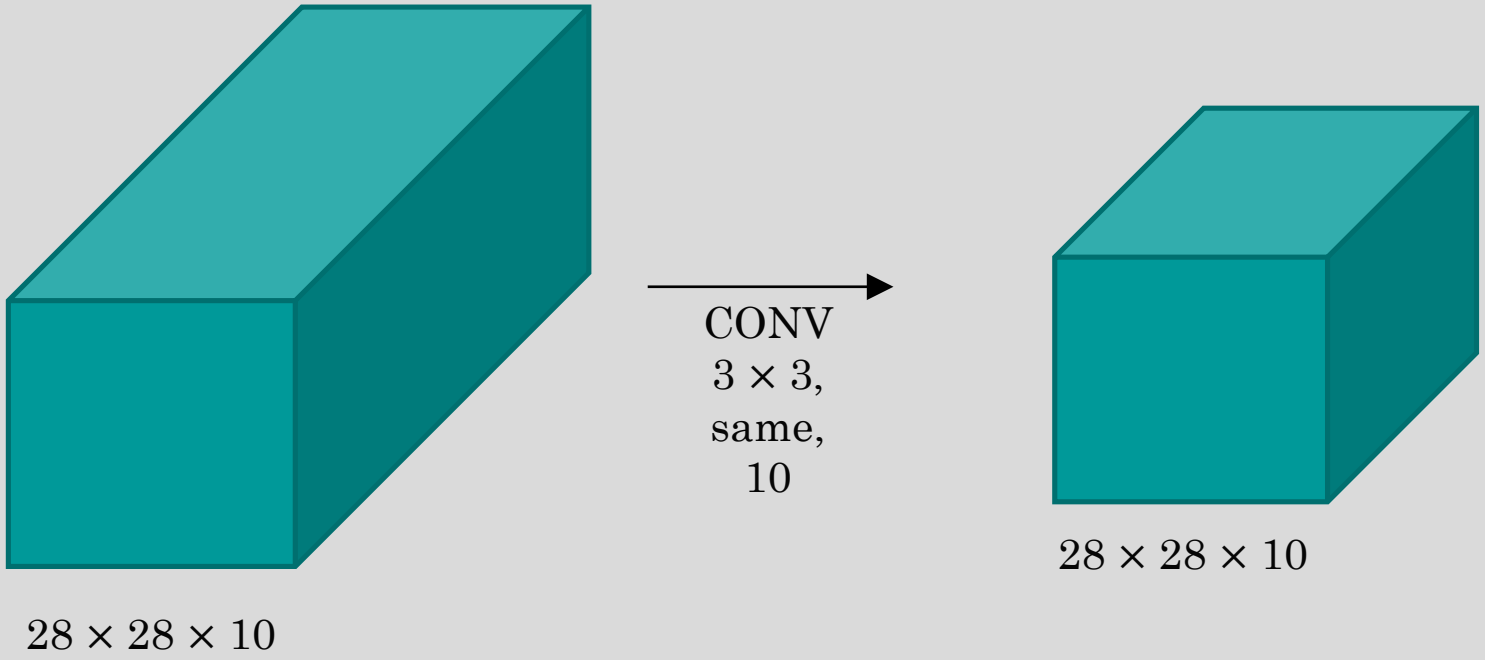
Spatial Factorization into Asymmetric Convolutions

- larger kernels might not be generally useful since they can be replaced with 3x3
  - Can we factorize 3x3 into smaller kernels?
    - Yes , we can factorize it into 2x2!
    - Any more?
- Lets do asymmetric factorization :
  - Nx1 and 1xN
  - Which one to choose 2x2 or two 3x1 and 1x3 ?
    - Use asymmetric factorization its 33% cheaper



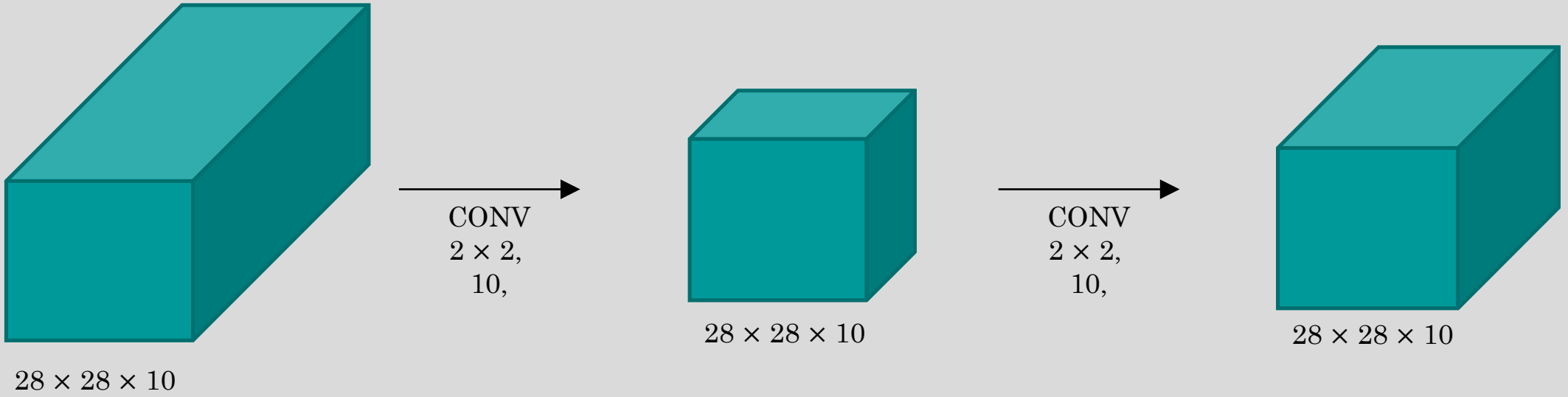
Mini-network replacing the 3x3 convolutions. The lower layer of this network consists of a 3x1 convolution with 3 output units.

default



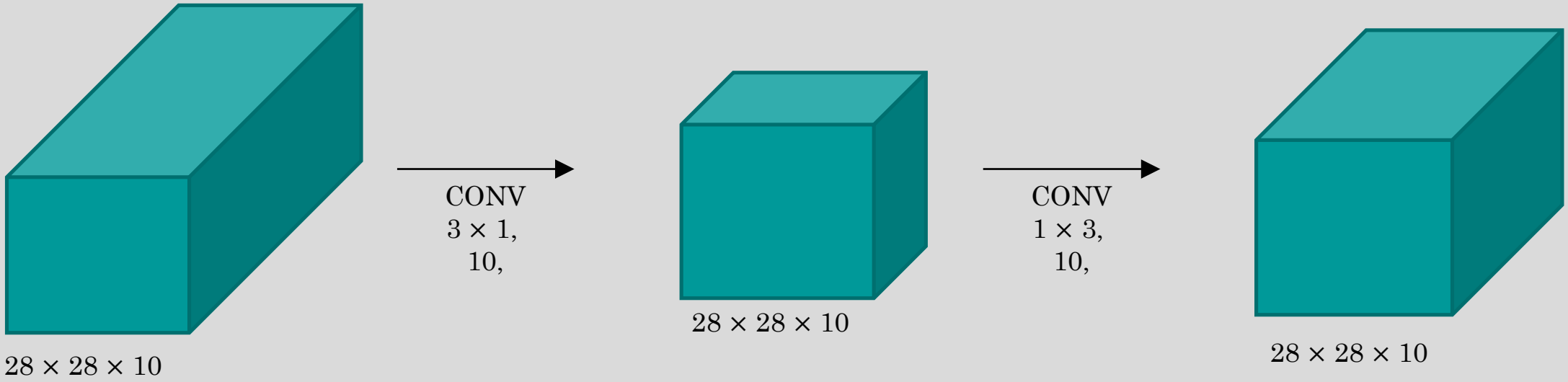
- (3x3x10)x10 = **900** parameters

Two 2x2 kernels



- (2x2x10)x10 = 400
  (2x2x10)x10 = 400
- 300+300 = 800 = **11% decrease**

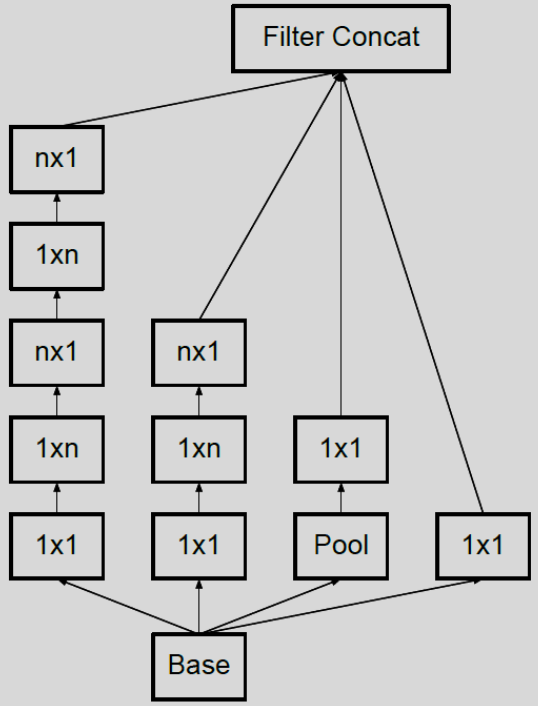
3x1 and 1x3



- (3x1x10)x10 = 300
  (1x3x10)x10 = 300
- 300+300 = 600 = **33% decrease**

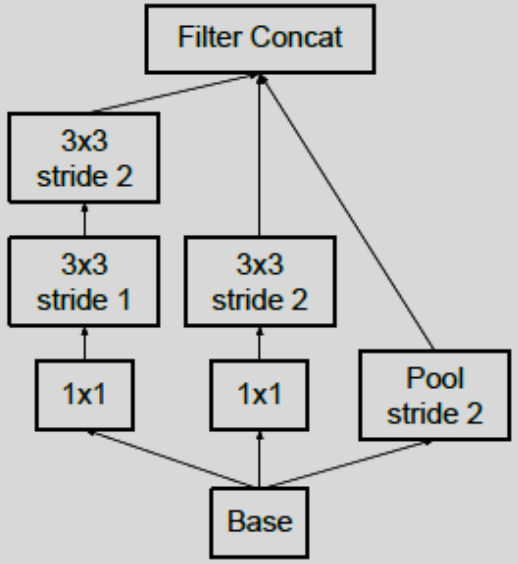
Spatial Factorization into Asymmetric Convolutions

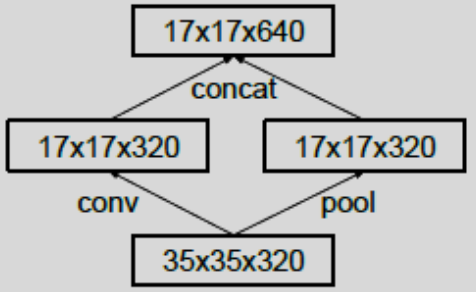
- Theoretically you can use any number, but the effect is not known, experiment it yourself.
- Always remember that this works exclusively well in inception like modules
- Don’t use factorization early in the network it will destroy valuable information



Grid size reduction

- Input volume $d * d * k$
- We need $\left(\frac{d}{2}\right) * \left(\frac{d}{2}\right) * 2k$
- If we use conv and then stride
  - $2d^2k^2$
- If we use one conv with stride 2 and
- One pooling and concat that we get
- $2\left(\frac{d}{2}\right)^2 k^2$





ResNet

- Introduces Residual connection , also known as skip connections
- It enhances the gradient flow, prevents degradation issue which hunts plain nets!
- What is a plain net? What is the degradation issue ?
  - Explain
- How does it prevent that ? Explain ResBlock
- Preactivation variant?
  - Resnet v1.1!?
    - See the post activation as pre activation , thus use BN ReLU again on them!

Depth Saturation

As we saw in previous works:

- The deeper is better, using BN we should no more face vanishing gradient but…
- After a certain depth, going deeper doesn't improve the accuracy and
- Doing so *lowers* the accuracy.

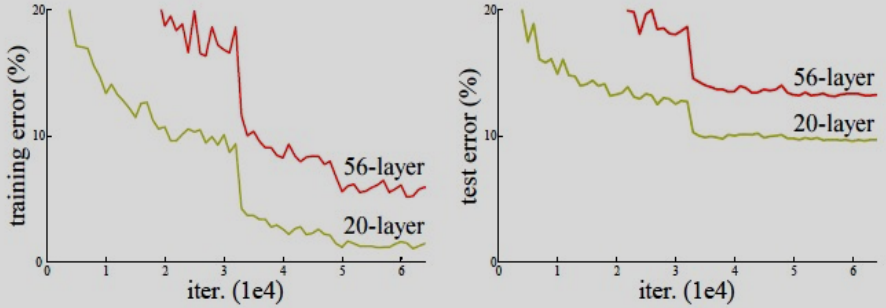
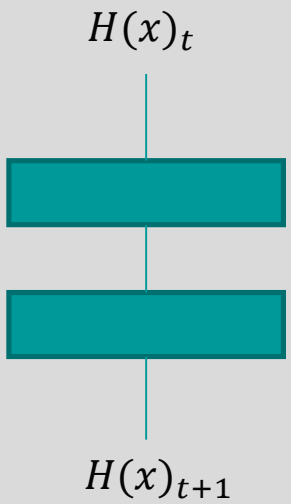


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Deep Residual Learning

- What is residual learning?
- Let $H(x)$ to be the underlying mapping to be fit by a few stacked layers. One can hypothesize that multiple nonlinear layers can asymptotically approximate the residual function $H(x)-x$
- Rather than expect the stacked layers to approximate $H(x)$, we explicitly let these layers approximate $F(x):= H(x)-x$

Deep Residual Learning



What we want is identity it means : we want

$$H(x)_{t+1} - H(x)_t = 0$$

$$H(x)_t = x$$

so we can write

$$H(x)_{t+1} - x$$

In fact we want to see how much difference there is between the two mappings (the two outputs)
Ideally this should be 0 . So we can write it as

$$F(x) = H(x)_{t+1} - x$$

And each time perturb it in a way so that the error f(x) is minimized . This means the solver needs to learn x
So that $x - x = 0$. And we know this wont happen , since if it were to happen, we wouldn't be talking about this in first place!
Therefore, Instead what we do is to change the formulation like this:

$$H(x)_{t+1} = F(x) + x$$

And by this, the solver should be able to easily set f(x) to zero to achieve x and therefore become identity . And this is what happens in reality ! Hence the name residual learning .
Moreover, $F(x) + x$ can be easily implemented in neural networks.

Deep Residual Learning

- How does that help?

If the added layers can be constructed as identity mappings, a deeper model should have training error no greater than its shallower counterpart.

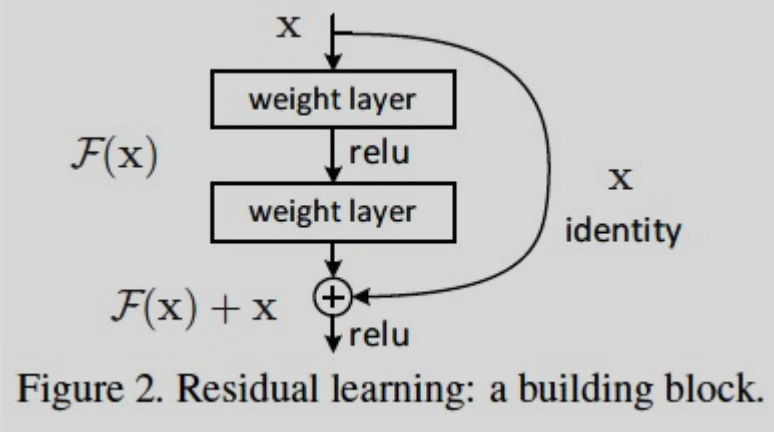


Figure 2. Residual learning: a building block.

Deep Residual Learning

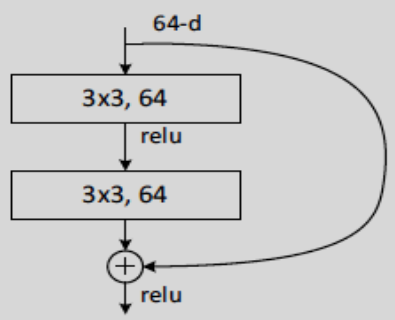
- What we gain?

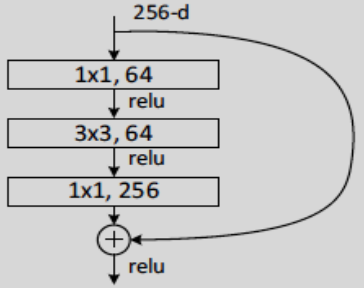
1. The solver has easier time converging by giving identity mapping as preconditioning
2. The shortcuts introduce no extra parameter nor computation complexity (compared with a “plain” counterpart)

ResNets

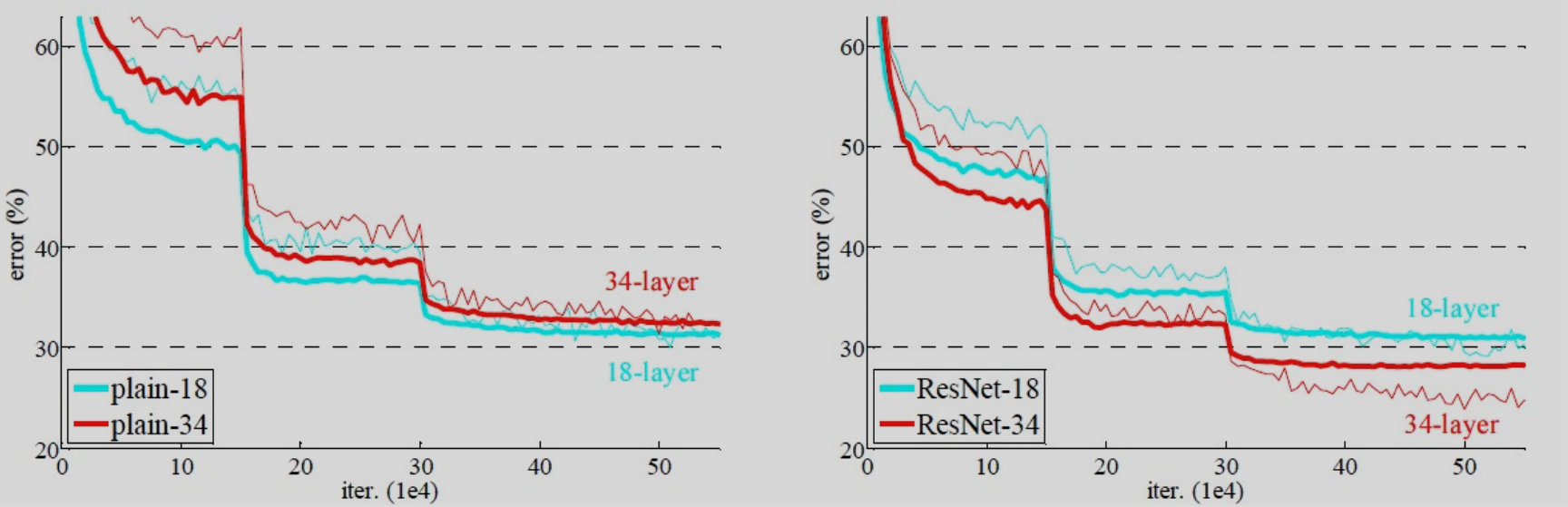
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
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

Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of blocks stacked. Down-sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.





ResNets VS Plain



Plain-18 beats Resnett18, indicating up to 18 layers, the solver can find the proper solution in parameter space. Whereas this changes when the architectures reach the depth of 34! ResNet wins (yet still PlainNet is comparable) . As the depth increases this becomes more accentuated.

ResNets VS Plain

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Exploring Over 1000 layers

- ResNet with depth of 1202 layers was tested and shows *no optimization difficulties* and achieved <0.1% training error.
- There are however problems using such a deep network.

	# layers	# params	
FitNet [35]	19	2.5M	8.39
Highway [42, 43]	19	2.3M	7.54 (7.72±0.16)
Highway [42, 43]	32	1.25M	8.80
ResNet	20	0.27M	8.75
ResNet	32	0.46M	7.51
ResNet	44	0.66M	7.17
ResNet	56	0.85M	6.97
ResNet	110	1.7M	**6.43** (6.61±0.16)
ResNet	1202	19.4M	7.93

Exploring Over 1000 layers

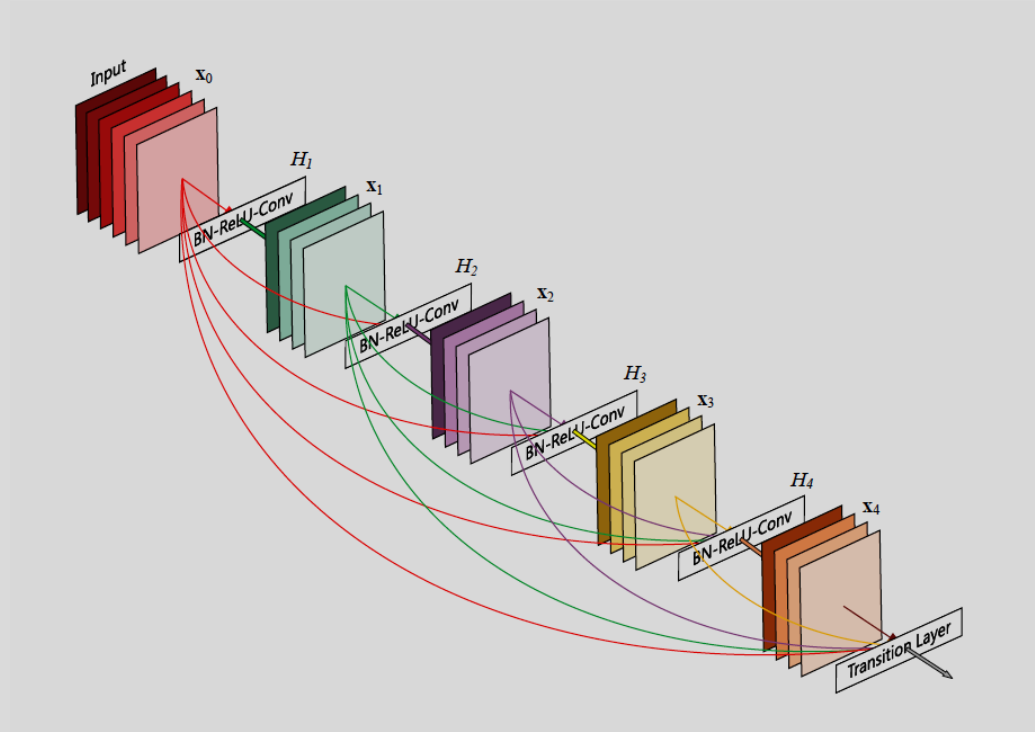
- The 1202-layer network may be too big for the CIFAR-10 dataset.
- Other best results applied maxout/dropout on the dataset, but ResNet only imposes regularization.

combining with stronger regularization may improve results.

Wide residual network

- ResNet ! But wider!
- What is the intuition
  - Wider is better
  - Wider makes parallelization easier!
  - In ResNet many layers don’t do much so why use them in first place?

DenseNet



DenseNet

DenseNet

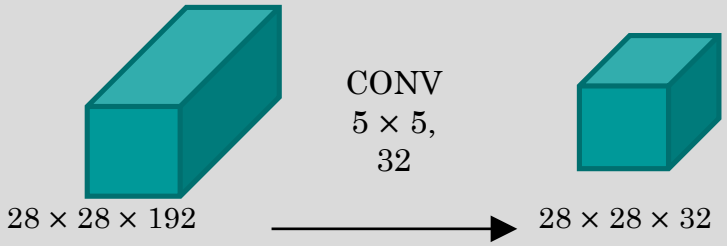
- DenseNet !
- What is the intuition (Ensure maximum information flow!)
  - Create better information pool, how? Just add all layers to all next layers. Why? This makes heavy use of feature reuse.
  - Just concat the feature-maps, never sum them!
  - What's bad about it? Its extremely slow,
  - There are efficient implementations that made it several times more usable!
  - It was impossible before to use higher number of parameters with this network! For CIFAR10, it took 4 Titanx to train the network with 26 Mb parameters !
  - to give you and idea, a model (DenseNet-BC, L = 100, k = 12, with 0.8M parameters) takes more than 8 gigabytes of video memory (or VRAM).

MobileNet

- Introduced in 2017! Provides VGGNet level accuracy with 30x less number of parameters
- This paper proposes a class of network architectures that allows a model developer to specifically choose a small network that matches the resource restrictions (latency, size) for their application
- Inspired by Flattened Convolutional architecture, used group separable convolutions!
- How?

Depthwise separable convolution

- A standard convolution, filters and combines outputs in one step
- A Depthwise convolution does this in two step:
  - the depthwise convolution applies a single filter to each input channel
  - The pointwise convolution then applies a 1x1 convolution to combine the outputs the depthwise convolution
- This reduces the computation and number of parameters drastically!
- If K is the kernel size, M the input channel, N the output channel and D_F the output dimension then :
- $D_K * D_K * M * N * D_F * D_F$



- (D_Kx D_KxM)xNx D_F× D_F
- (5x5x192)x32x28x28 = **120**,422,400 = **120M** FLOPS
- 5x5x192x32 = 153,600 parameters

MobileNet

- How it reduces the computation
- Depthwise convolution :
  - $D_K . D_K . M . D_F . D_F$
- Pointwise convolution :
  - $M . N . D_F . D_F$
- Pointwise separable convolution:
  - $D_K . D_K . M . D_F . D_F + M . N . D_F . D_F$
- And the reduction rate achieved by separable convolution :
  - $$\frac{D_K . D_K . M . D_F . D_F + M . N . D_F . D_F}{D_K . D_K . M . N . D_F . D_F} = \frac{1}{N} + \frac{1}{D_k^2}$$

(D_Kx D_KxM) x $D_F \times D_F$
(5x5x192)x28x28 = **3,763,200**

(D_Kx D_KxM)xN x $D_F \times D_F$
(1x1x192)x32x28x28 = **4,816,896**

$\mathbf{3,763,200 + 4,816,896}$ = **8,580,096 FLOPS**

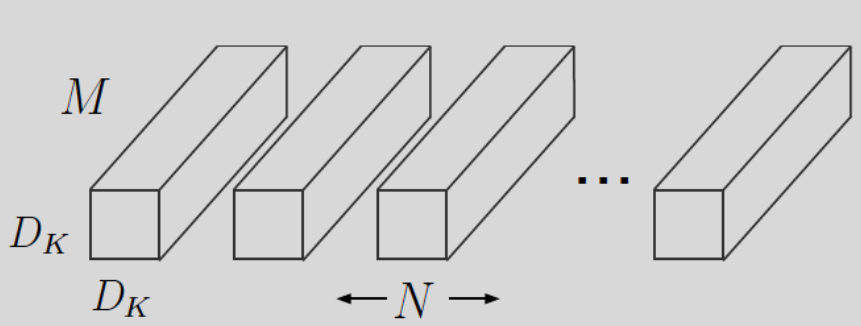
$\mathbf{5 * 5 * 192 = 4,800}$ **Params**
$\mathbf{192 * 32 = 6,144}$ **Params**

$\mathbf{4,800 + 6,144 = 10,944}$ **Parameters**

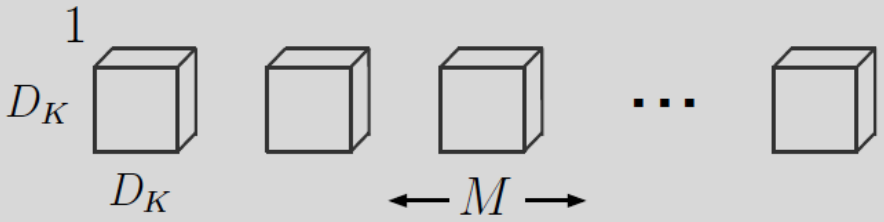
$\frac{10944}{153600}$ x100 = 7.12%

$\frac{1}{32} + \frac{1}{25}$ x100 = 7.12%

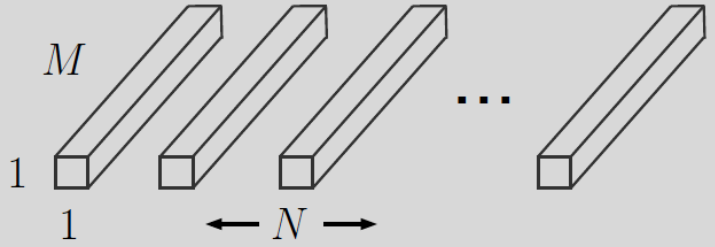
Depthwise separable convolution



(a) Standard Convolution Filters



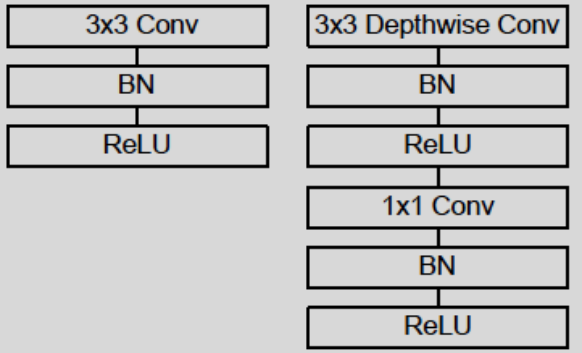
(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

MobileNet

- Used 3x3 kernels
- Decreased computations up to 8~9 times!
- Didn’t need any factorization schemes introduced by Inception v3
- Slight influence on the overall accuracy
- They used BN and ReLU for both stages (while in normal conv this is a linear operation, here it becomes nonlinear)
- Use strided convolution for “down-sampling”
- Use global average pooling followed by a fc layer
- Has 28 layers

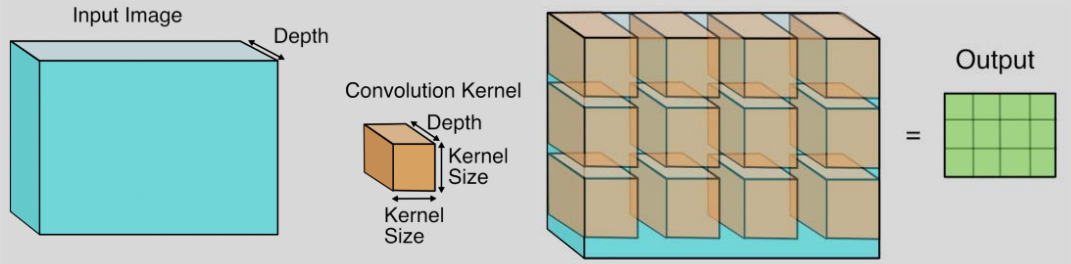


MobileNet architecture

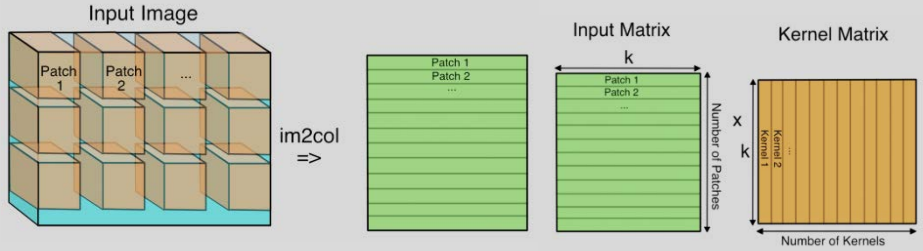
Table 1. MobileNet Body Architecture

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

MobileNet



- 95% of the computation happens in 1x1 convolutions
- 75% of all parameters reside in 1x1 conv layers
- It can be efficiently executed using GEMM(general matrix multiply
  - No need for reordering, or tricks such as im2col
  - Its fast!



- No weight decay on depth-wise convolutions
  - Since there are so few parameters in them
- Width Multiplier: Suggests a multiplier α, to achieve the best compromise in width(number of neurons per layer)
  - $D_K . D_K . \alpha M . D_F . D_F + \alpha M . \alpha N . D_F . D_F$
- Resolution multiplier : Suggests a multiplier ρ(Rho), to achieve the best compromise in feature-map size
  - $D_K . D_K . \alpha M . \rho D_F . \rho D_F + \alpha M . \alpha N . \rho D_F . \rho D_F$

HAMIM©

MobileNet

Table 4. Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

Table 5. Narrow vs Shallow MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.75 MobileNet	68.4%	325	2.6
Shallow MobileNet	65.3%	307	2.9

Table 6. MobileNet Width Multiplier

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Table 7. MobileNet Resolution

Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

MobileNet

Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

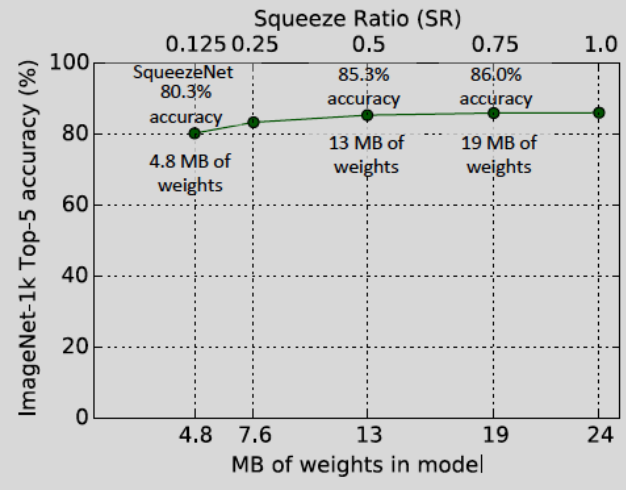
Table 9. Smaller MobileNet Comparison to Popular Models

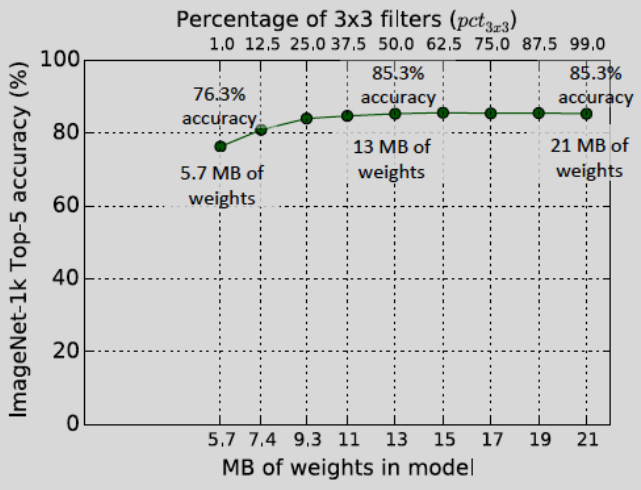
Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.50 MobileNet-160	60.2%	76	1.32
Squeezenet	57.5%	1700	1.25
AlexNet	57.2%	720	60

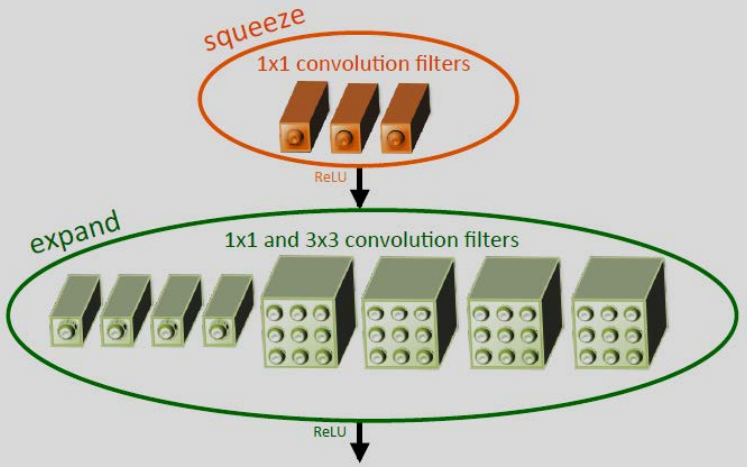
SqueezeNet

- One of the most interesting and lightweight architectures out there!, fully convolutional by the way! (inspired by NIN!)
- Extremely small yet performs at AlexNet level , with 50X less parameters
- Has 3 principles
  - Replace 3x3 by 1x1
  - Decrease number of input channels to 3z3
  - Down-sample late in the network so that layers have larger activation maps
- 20 layer architecture, different revisions, has several claims:
  - Local correlation is not that important ! Proof ?
    - Creating architecture with mostly 3x3 vs 1x1! This is wrong!
  - Fire-modules and bottlenecks !
  - Basically that’s it!
- Enhances architecture with skip connections or as they call it bypass connections
  - Why does it help ?
    - They reduce dimensionality severely and thus a lot of information is lost, bypass connections relieve such a problem to some extent
- A very good candidate for embedded systems or environments with minimum requirements
- The problem?
  - You can not scale it to achieve higher results after certain limits , or at least it wont be as efficient

SqueezeNet

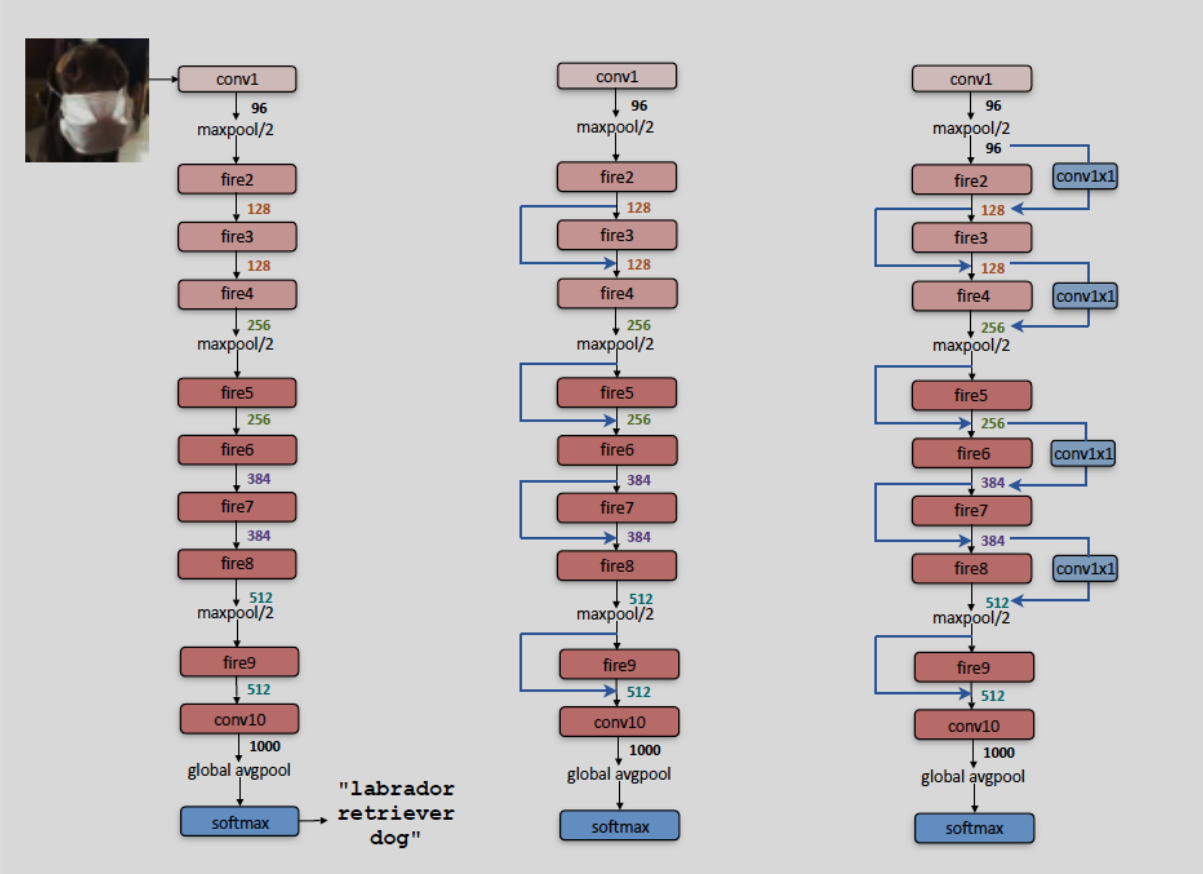






HAMIM©

SqueezeNet



Striving For simplicity

- Published in 2015, interesting points:
  - Instead of pooling (max) use strided convolutions
  - They claim it performs better , since Conv layers learn the same thing better
  - Is it correct ? Lets dig deeper
    - As we can see, proper use of pooling can enhance ones performance, therefore don't assume a conv layer with stride 2 performs just as a pooling operation, it does not. A conv by itself can not learn what pooling does, at the very least it needs an activation function to do that, to approximate or simulate pooling operation of a kind. Read the refs at the end for more information
  - Also introduces a new way for visualization

Lets Keep it simple V1/Architecture design

- State of the art result on CIFAR10, back in 2015, without data augmentation
- 2 to 25x less number of parameter than VGG, ResNet, ….
- Says keep everything simple first and then extend /upgrade your architecture with new inventions, algorithms,
- The idea is when everything works with all the limitation, removing such limitations, improves the performance, and also results in more efficiency and less overhead!
- V2 expands on this and almost outperforms all deeper and more complex architectures despite having several times fewer parameters!

Lets Keep it simple V1_Architecture design

- Depth , how to specify one ?
- Pooling Operation /Maxpooling /strided convolution /global average pooling
- Dropout /DeadReLU issue, feature adaptability
- Benchmarks
- Notice:
  - The experiments and results which we are to demonstrate belong to simpnetv2 paper, which extends the concept introduced in the first version.

Lets Keep it simple - Principles

- Gradual Expansion and minimum allocation
  - PLS/PLD issues
  - Prevents excessive allocation of processing units
  - Prevents PLS/PLD issues
- Homogenous groups of layers
  - Helps to achieve granular fine-tuning
  - Better management of processing unit distribution
- Correlation Preservation
  - 1x1 vs 2x2 vs 3x3 vs 5x5 , which one to choose and where to use?
- Maximum information utilization
  - Pooling vs strided convolution (downsampling or new pooling)
- Maximum performance utilization
- Balanced distribution scheme
- Dropout utilization
  - Dead ReLU issue
  - Better generalization by adapting to new feature compositions

Lets Keep it simple - Principles

- Gradual Expansion and minimum allocation
  - Processing level saturation haunts shallower architectures
  - Processing level deprivation haunts deep architectures

TABLE II: Gradual expansion of the number of layers.

Network Properties	Parameters	Accuracy (%)
Arch1, 8 Layers	300K	90.21
Arch1, 9 Layers	300K	90.55
Arch1, 10 Layers	300K	90.61
Arch1, 13 Layers	300K	89.78

TABLE III: Shallow *vs*. Deep (related to *Minimum Allocations*) showing how a gradual increase can yield better performance with fewer number of parameters.

Network Properties	Parameters	Accuracy (%)
Arch1, 6 Layers	1.1M	92.18
Arch1, 10 Layers	570K	92.23
Arch1, 13 Layers	500K	92.42

Gradual expansion and minimum allocation

TABLE II: Gradual expansion of the number of layers.

Network Properties	Parameters	Accuracy (%)
Arch1, 8 Layers	300K	90.21
Arch1, 9 Layers	300K	90.55
Arch1, 10 Layers	300K	90.61
Arch1, 13 Layers	300K	89.78

TABLE III: Shallow *vs*. Deep (related to *Minimum Allocations*) showing how a gradual increase can yield better performance with fewer number of parameters.

Network Properties	Parameters	Accuracy (%)
Arch1, 6 Layers	1.1M	92.18
Arch1, 10 Layers	570K	92.23
Arch1, 13 Layers	500K	92.42

Homogeneous Groups of Layers

- Helps to achieve granular fine-tuning
- Better management of processing unit distribution
- Former architectures use it implicitly but have not taken the full advantage of such thing.

Correlation preservation

- Correlation the corner stone of CNN!
- Don’t use 1x1 or fc when locality of information matters (exclusively earlier layers)
- SqueezeNet had different viewpoint! Which one is correct?
- Why?
  - Ad hoc design, vague allocation scheme
  - Severe bottleneck which destroy a lot of useful information
  - It also can not scale properly
- Use 3x3, if not 2x2, if not 1x1
- Don’t use excessive amounts of processing units on 1x1 at the end

TABLE S3: Correlation Preservation: Squeezenet vs SimpNet on CIFAR10.

Network Properties	Parameters	Accuracy (%)
Squeezenet1.1_default	768K	88.60
Squeezenet1.1_optimized	768K	92.20
SimpNet_Slim	300K	93.25
SimpNet_Slim	600K	94.03

Correlation preservation

TABLE VII: Accuracy for different combinations of kernel sizes and number of network parameters, which demonstrates how correlation preservation can directly effect the overall accuracy.

Network Properties	Parameters	Accuracy (%)
Arch4, 3×3	300K	90.21
Arch4, 3×3	1.6M	92.14
Arch4, 5×5	1.6M	90.99
Arch4, 7×7	300K.v1	86.09
Arch4, 7×7	300K.v2	88.57
Arch4, 7×7	1.6M	89.22

TABLE VIII: Different kernel sizes applied on different parts of a network affect the overall performance, *i.e.*, the kernel sizes that preserve the correlation the most yield the best accuracy. Also, the correlation is more important in early layers than it is for the later ones.

Network Properties	Params	Accuracy (%)
Arch5, 13 Layers, 1×1 *vs.* 2×2 (early layers)	128K	87.71 *vs.* 88.50
Arch5, 13 Layers, 1×1 *vs.* 2×2 (middle layers)	128K	88.10 *vs.* 88.51
Arch5, 13 Layers, 1×1 *vs.* 3×3 (smaller *vs.* bigger end-avg)	128K	89.44 *vs.* 89.60
Arch5, 11 Layers, 2×2 *vs.* 3×3 (bigger learned feature-maps)	128K	89.30 *vs.* 89.44

Correlation preservation

TABLE VII: Accuracy for different combinations of kernel sizes and number of network parameters, which demonstrates how correlation preservation can directly effect the overall accuracy.

Network Properties	Parameters	Accuracy (%)
Arch4, 3×3	300K	90.21
Arch4, 3×3	1.6M	92.14
Arch4, 5×5	1.6M	90.99
Arch4, 7×7	300K.v1	86.09
Arch4, 7×7	300K.v2	88.57
Arch4, 7×7	1.6M	89.22

TABLE VIII: Different kernel sizes applied on different parts of a network affect the overall performance, *i.e.*, the kernel sizes that preserve the correlation the most yield the best accuracy. Also, the correlation is more important in early layers than it is for the later ones.

Network Properties	Params	Accuracy (%)
Arch5, 13 Layers, 1×1 *vs.* 2×2 (early layers)	128K	87.71 *vs.* 88.50
Arch5, 13 Layers, 1×1 *vs.* 2×2 (middle layers)	128K	88.10 *vs.* 88.51
Arch5, 13 Layers, 1×1 *vs.* 3×3 (smaller *vs.* bigger end-avg)	128K	89.44 *vs.* 89.60
Arch5, 11 Layers, 2×2 *vs.* 3×3 (bigger learned feature-maps)	128K	89.30 *vs.* 89.44

Maximum information utilization

- We need more information, what are the sources?
  - Larger dataset
  - Data-augmentation
  - Better information pools?
- Information pools
  - Resblock
  - Densblock
  - Pooling fusion
  - …
- Simple form of information pool
  - Larger input/feature-maps
- What if we don’t have access to more data?
  - Utilize what you have in the most efficient way
- How? Avoid rapid down-sampling /doing pooling
  - Larger feature-maps provide more information.
    Instead of adding more complexity and increasing depth or width, increase input dimension first
- Excessive use results in large amount of memory consumption (DenseNet 0.8param takes 8Gig vram)

Maximum information utilization

- Delayed pooling and how it affects performance

Network Properties	Parameters	Accuracy (%)
Arch3, L5 default	53K	79.09
Arch3, L3 early pooling	53K	77.34
Arch3, L7 delayed pooling	53K	79.44

Strided convolution vs Maxpooling

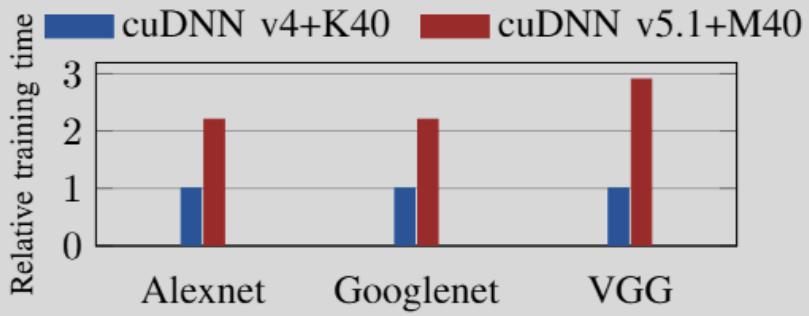
TABLE VI: Eeffect of using Strided convolution (†) *vs.* Max-pooling (∗). Max-pooling outperforms the Strided convolution regardless of specific architecture. First three rows are tested on CIFAR100 and two last on CIFAR10

Network Properties	Depth	Parameters	Accuracy (%)
SimpNet∗	13	360K	69.28
SimpNet∗	15	360K	68.89
SimpNet†	15	360K	68.10
ResNet∗	32	460K	93.75
ResNet†	32	460K	93.46

Maximum Performance utilization

- Use 3x3 and benefit from underlying optimizations

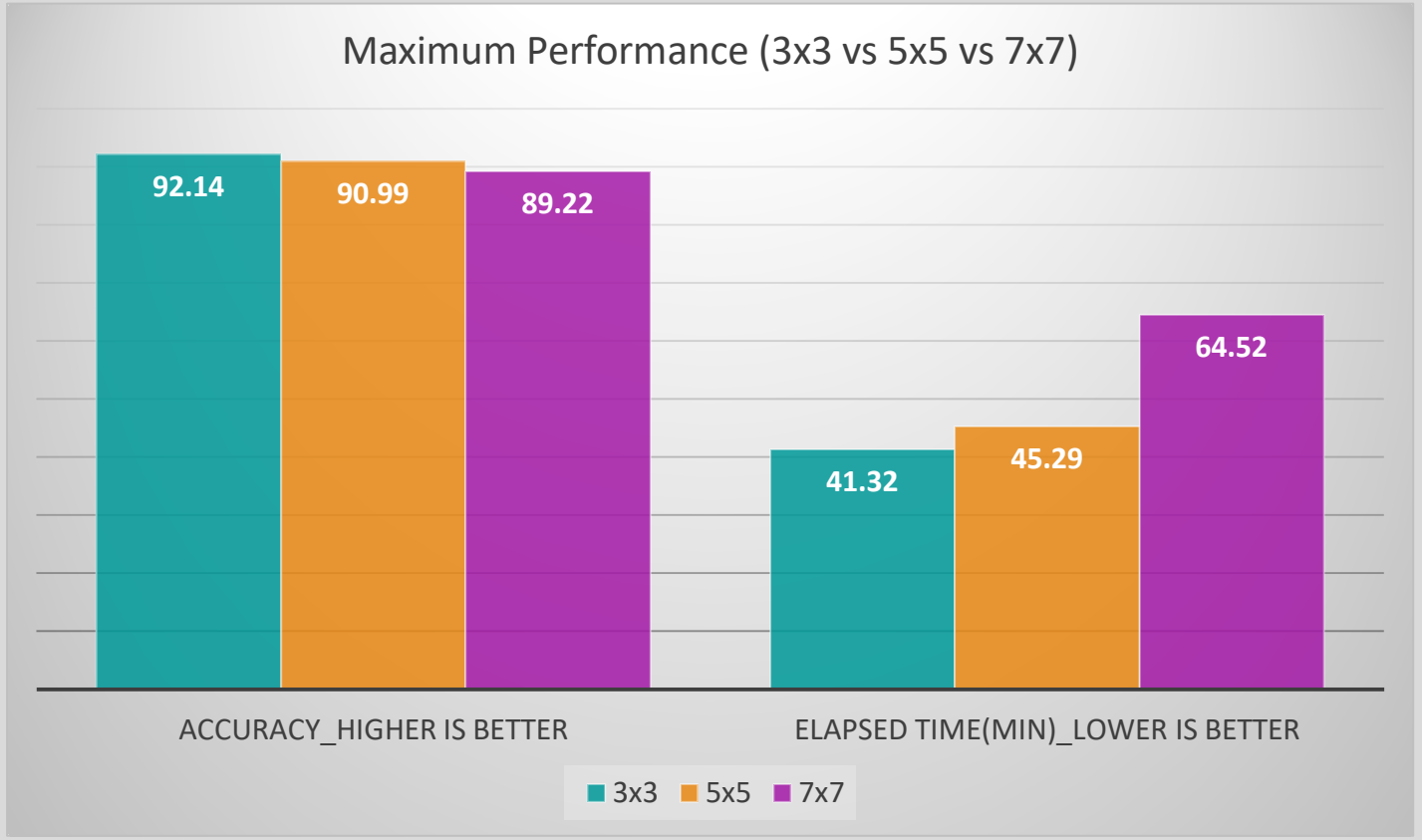
Fig. 1: Comparisons of the speedup of different architectures. This plot shows 2.7× faster training when using 3×3 kernels using cuDNN v5.x.



	k80+cuDNN 6	P100+ cuDNN 6	v100+cuDNN 7
2.5x + CNN	100	200	600
3x + LSTM	1x	2x	6x

TABLE I: Improved performance by utilizing cuDNN v7.x.

Maximum Performance utilization



The same architecture with 1.6Million parameters with different kernels

Balanced distribution

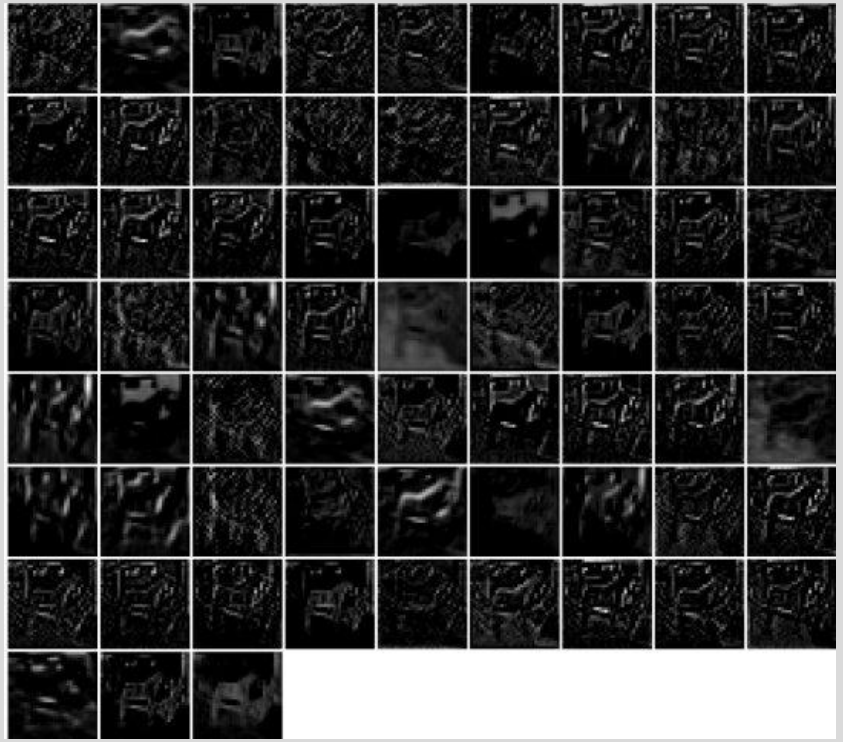
TABLE IV: Balanced distribution scheme is demonstrated by using two variants of simplnet architecture with 10 and 13 layers respectively, each showing how the difference in allocation results in varying performance and ultimately improvements for the one with balanced distribution of units.

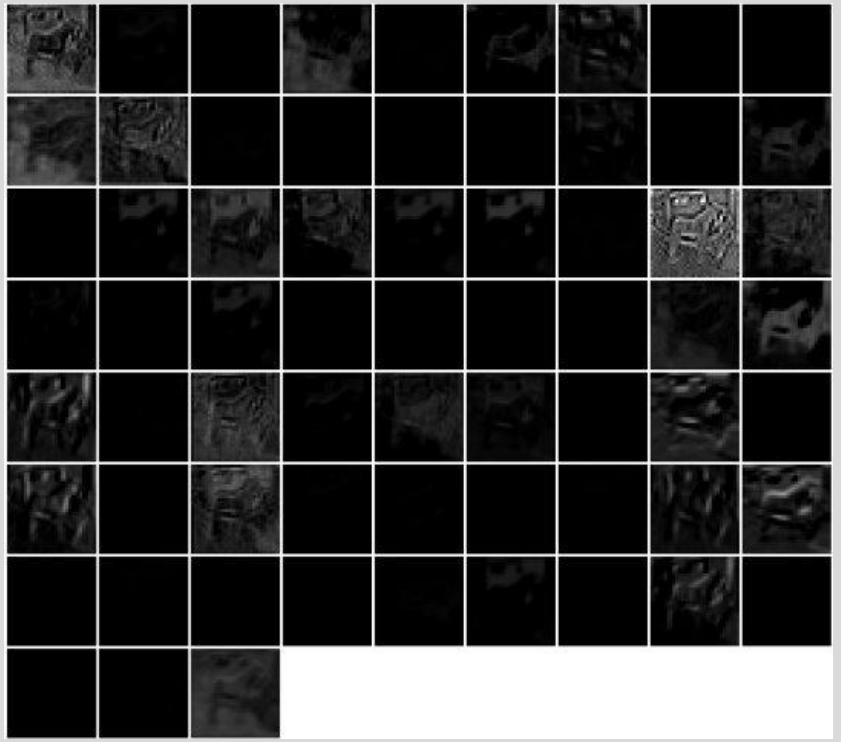
Network Properties	Parameters	Accuracy (%)
Arch2, 10 Layers (wide end)	8M	95.19
Arch2, 10 Layers (balanced width)	8M	95.51
Arch2, 13 Layers (wide end)	128K	87.20
Arch2, 13 Layers (balanced width)	128K	89.70

Dropout Utilization

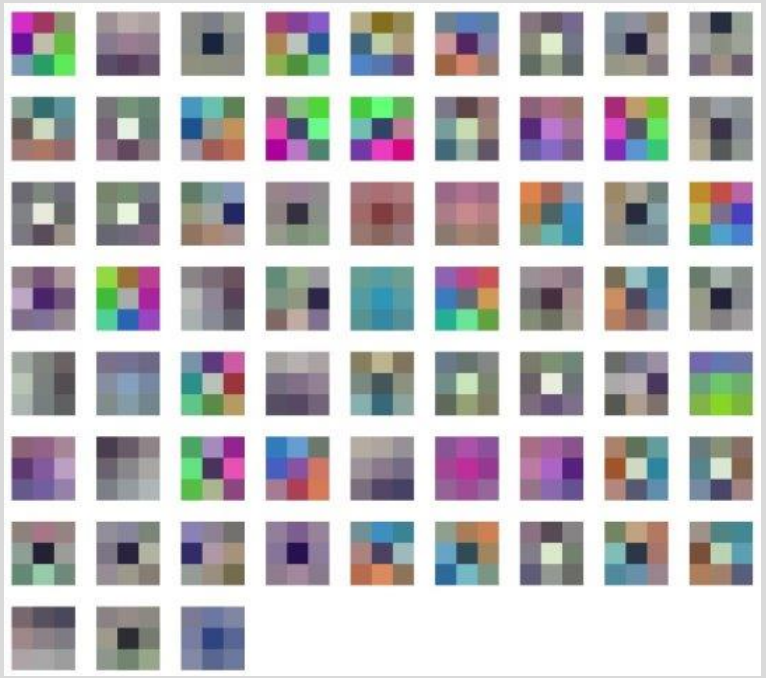
- Hinton says use 50% on fully connected layers
- There is a catch here!
  - Why only FC layers? Why not Conv layers as well?
  - The ratio may be high(most of the time is) and cause under-fitting/ at best it prolongs the convergence
- Use dropout on all layers
- Start with a small dropout probability e.g. 0.2 (especially if you are using BN)
- What's the effect of dropout applied on all layers?
  - Results in better generalization by creating diverse feature compositions
  - Creates more robust filters
  - Becomes more robust against noise and different changes in input
  - Prevents Dead ReLU issue
  - Provides more sparsity in the network which enhances generalization

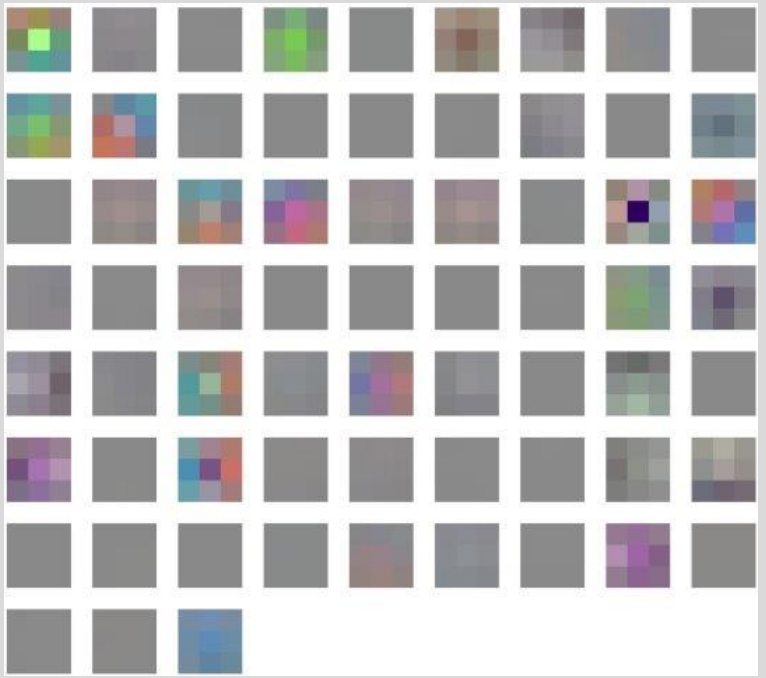
Lets Keep it simple V1/Architecture design



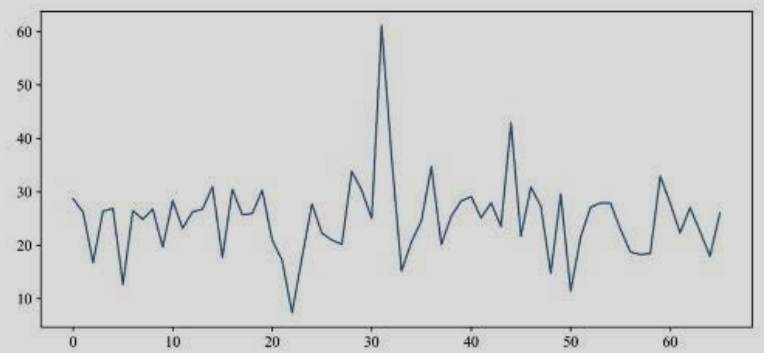


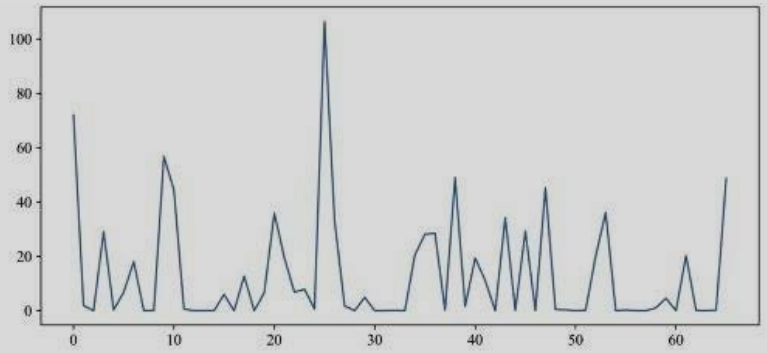
Lets Keep it simple V1_Architecture design

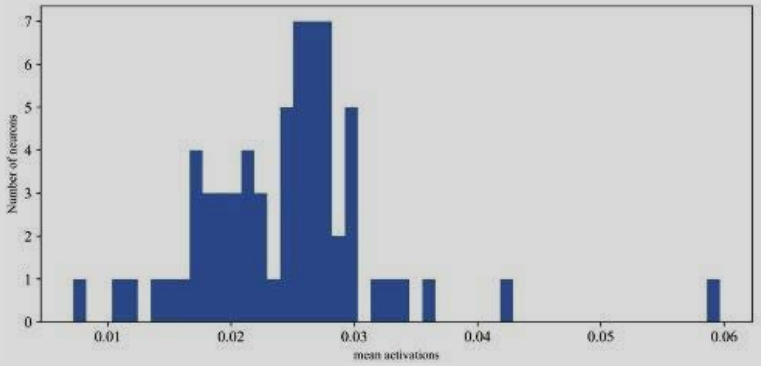


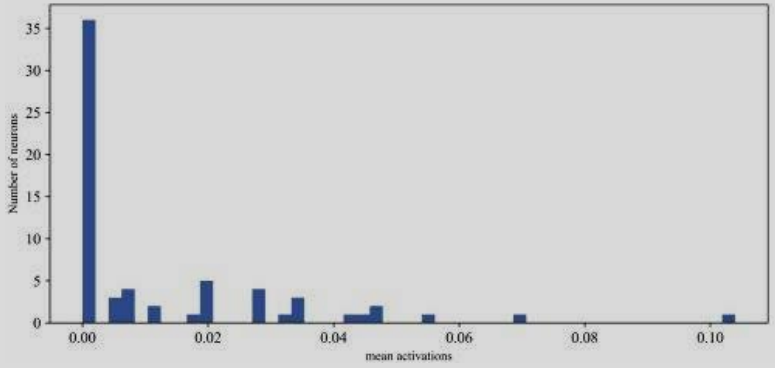


Lets Keep it simple V1_Architecture design

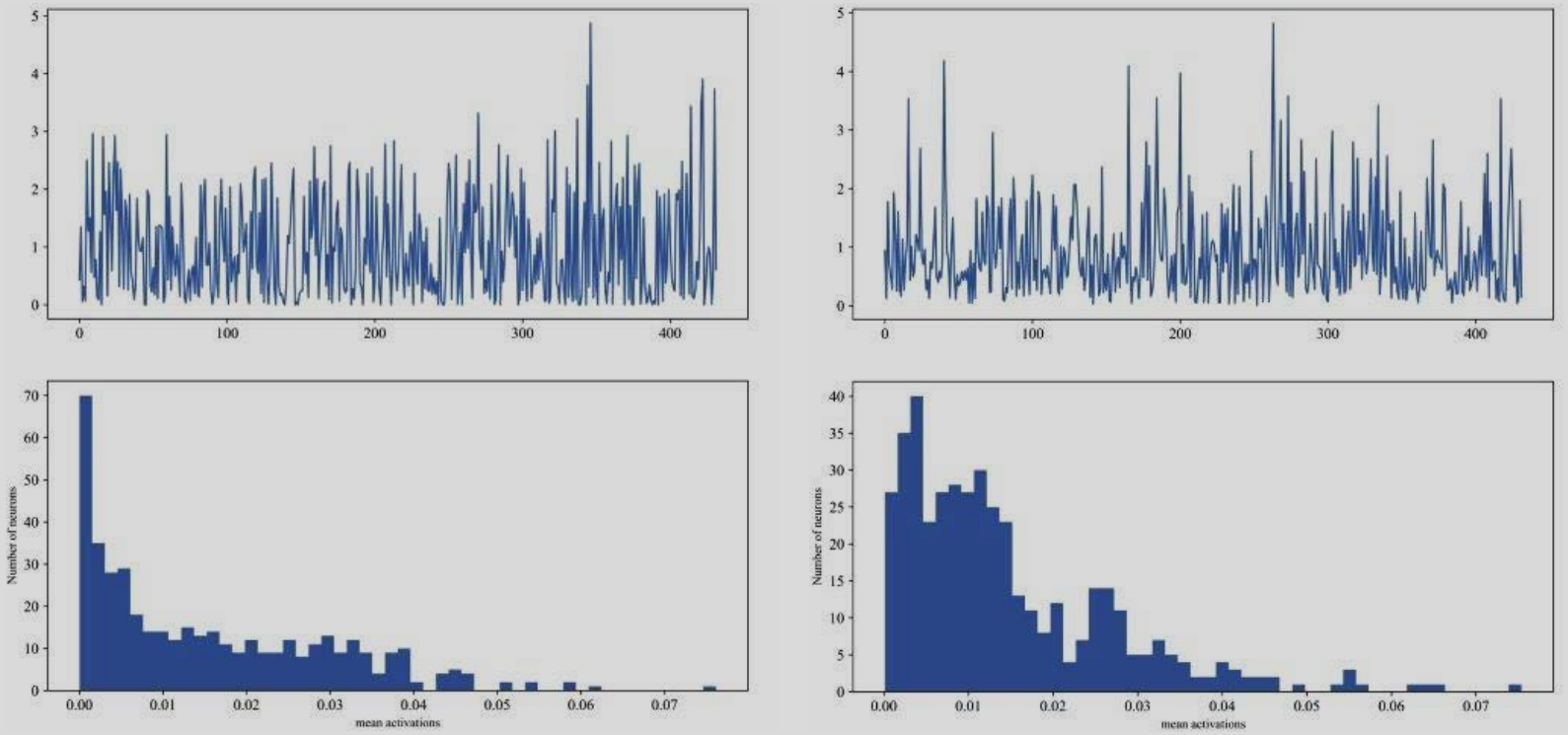








Lets Keep it simple V1_Architecture design



Architecture

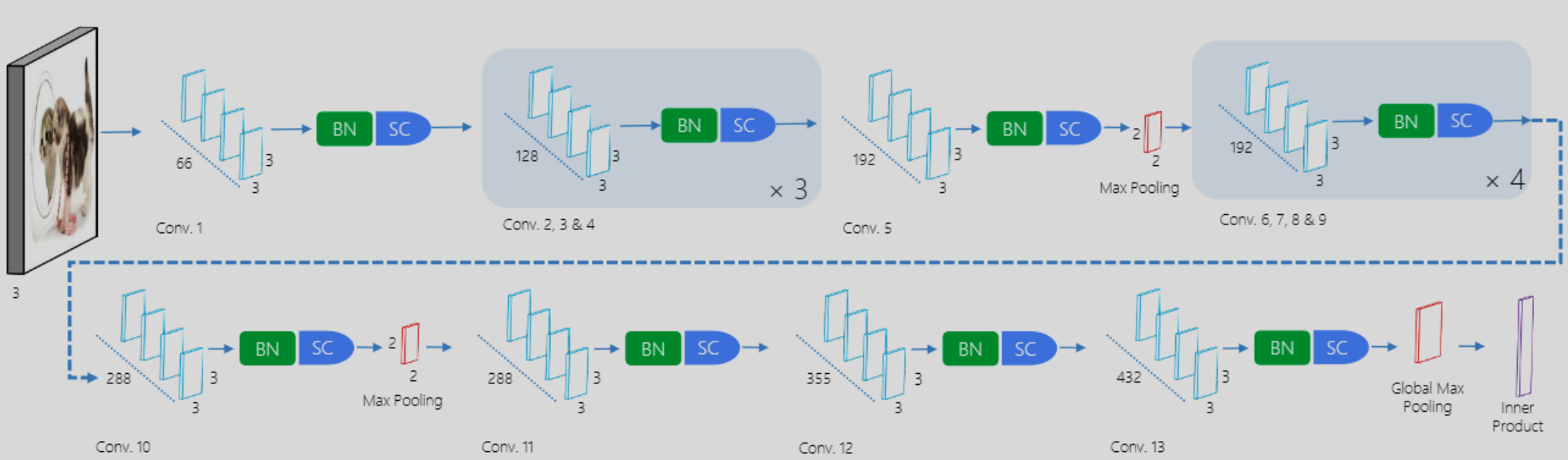


Fig. 6: SimpNet base architecture with no drop-out.

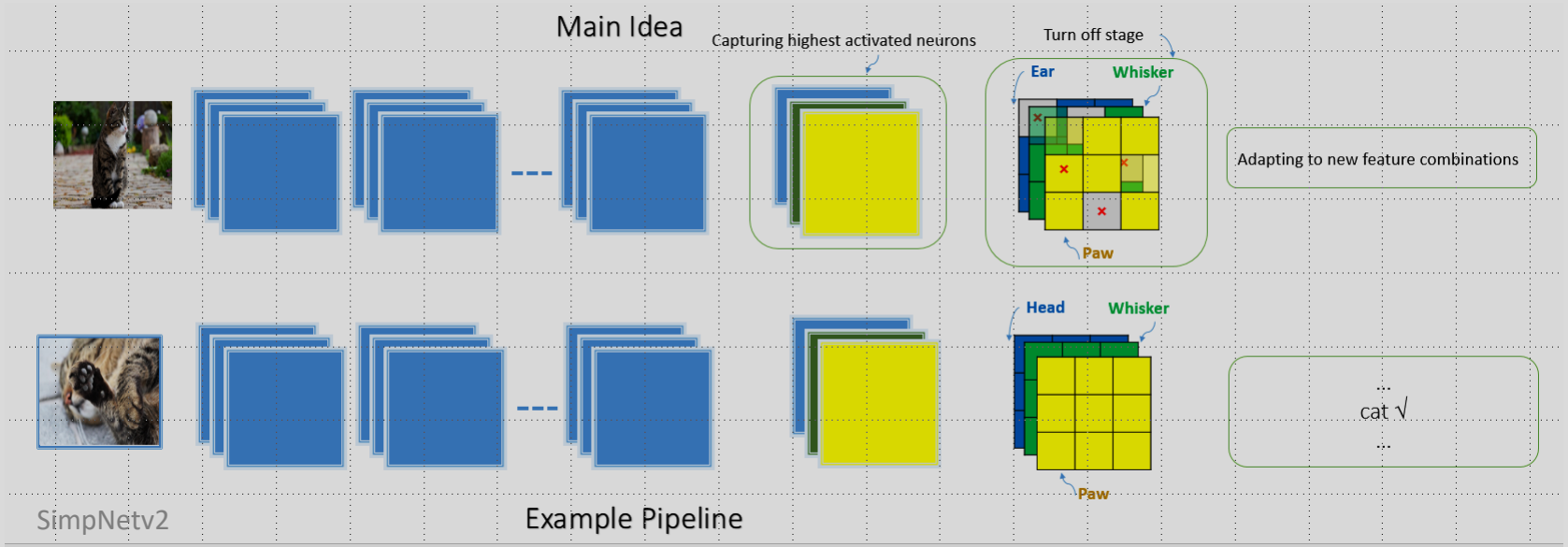
SimpNet Slimmed (SlimNet)

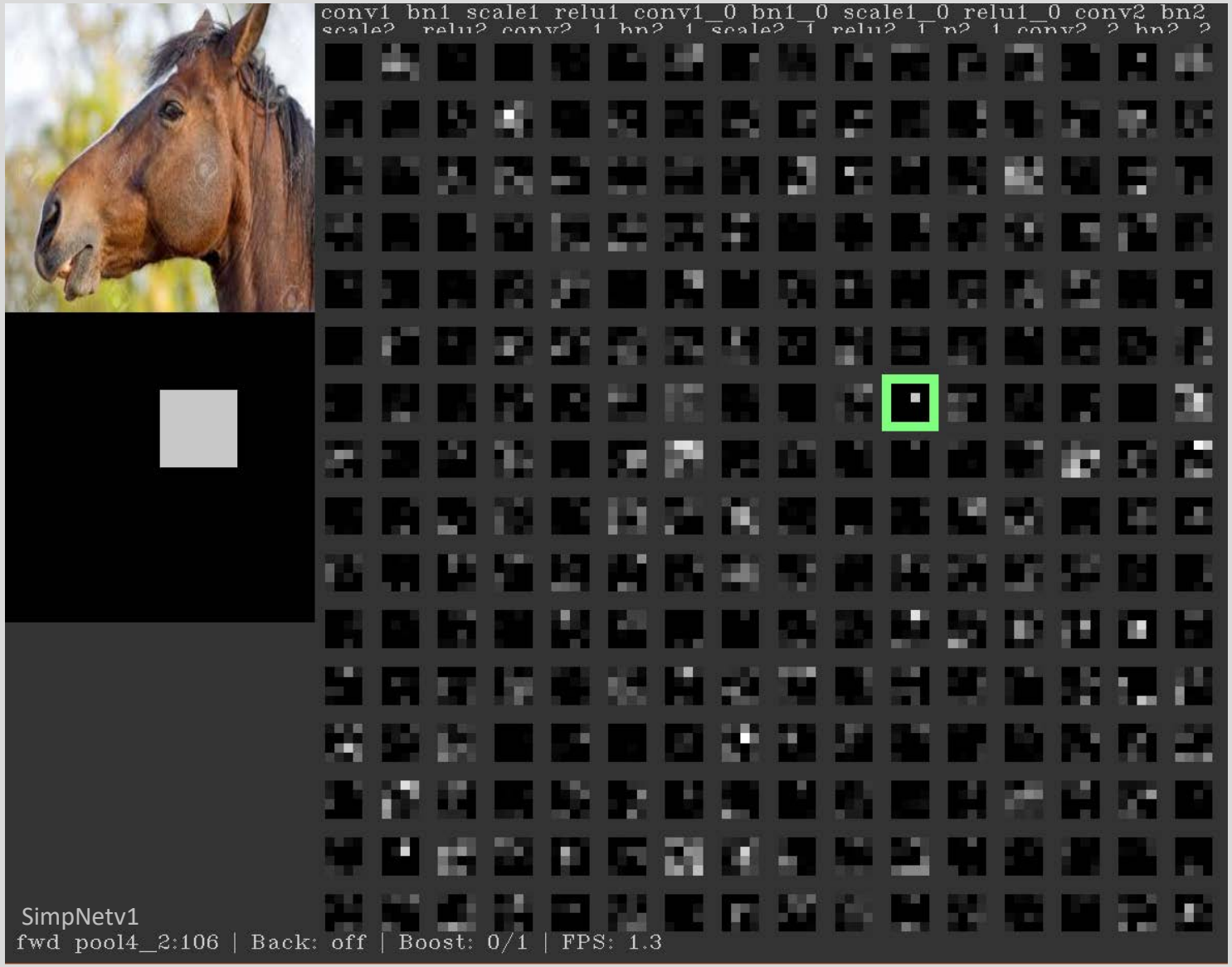
TABLE XV: Slimmed version results on different datasets.

Model	Param	CIFAR10	CIFAR100
Ours	300K - 600K	93.25 - 94.03	68.47 - 71.74
Maxout [15]	6M	90.62	65.46
DSN [67]	1M	92.03	65.43
ALLCNN [42]	1.3M	92.75	66.29
dasNet [71]	6M	90.78	66.22
ResNet [2] (Depth32, tested by us)	475K	93.22	67.37-68.95
WRN [9]	600K	93.15	69.11
NIN [41]	1M	91.19	—

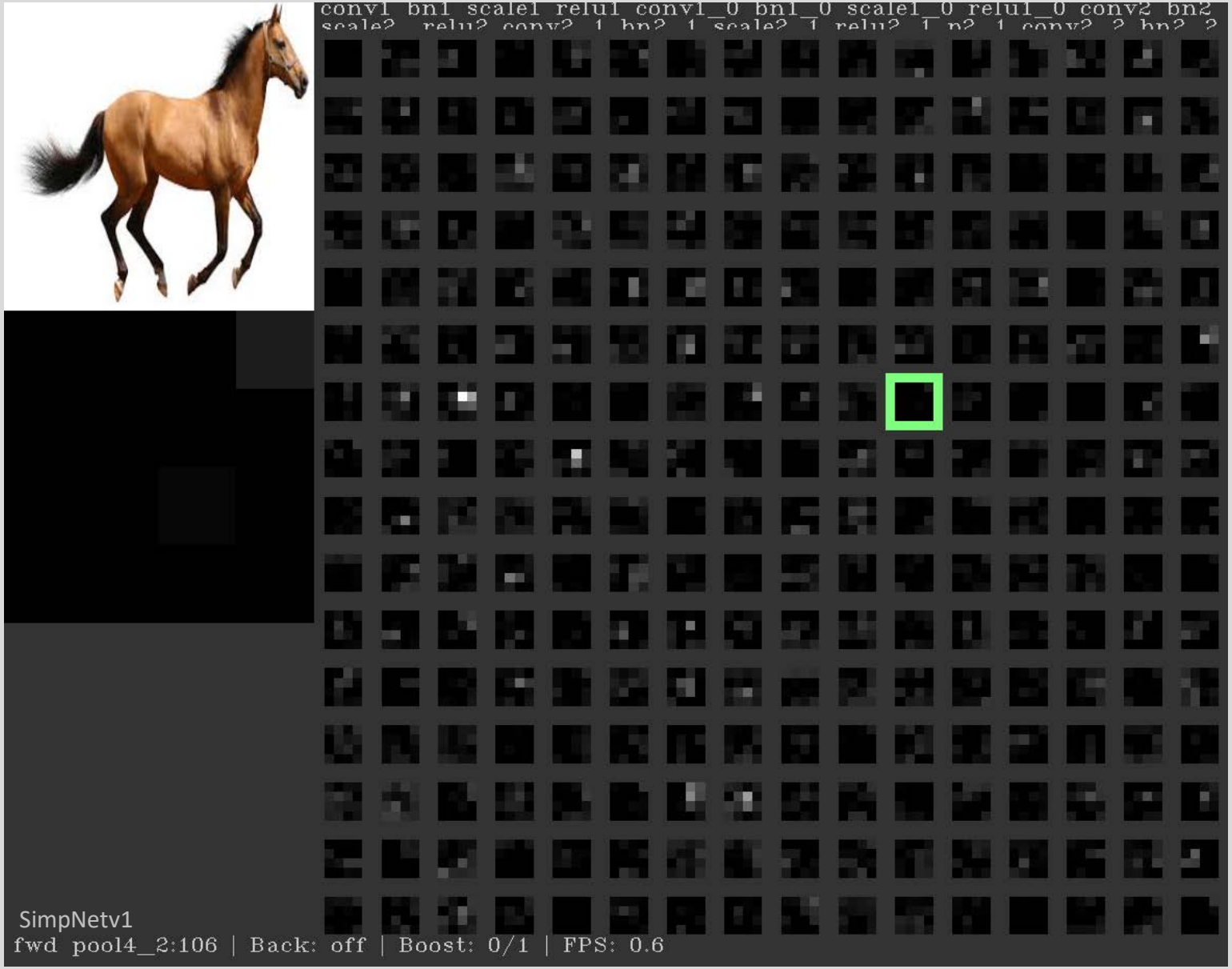
Simple Adaptive Feature Composition Pooling

- What's special about pooling ?
- Invariance ?
- Down-sampling ?
- What's the intuition ?

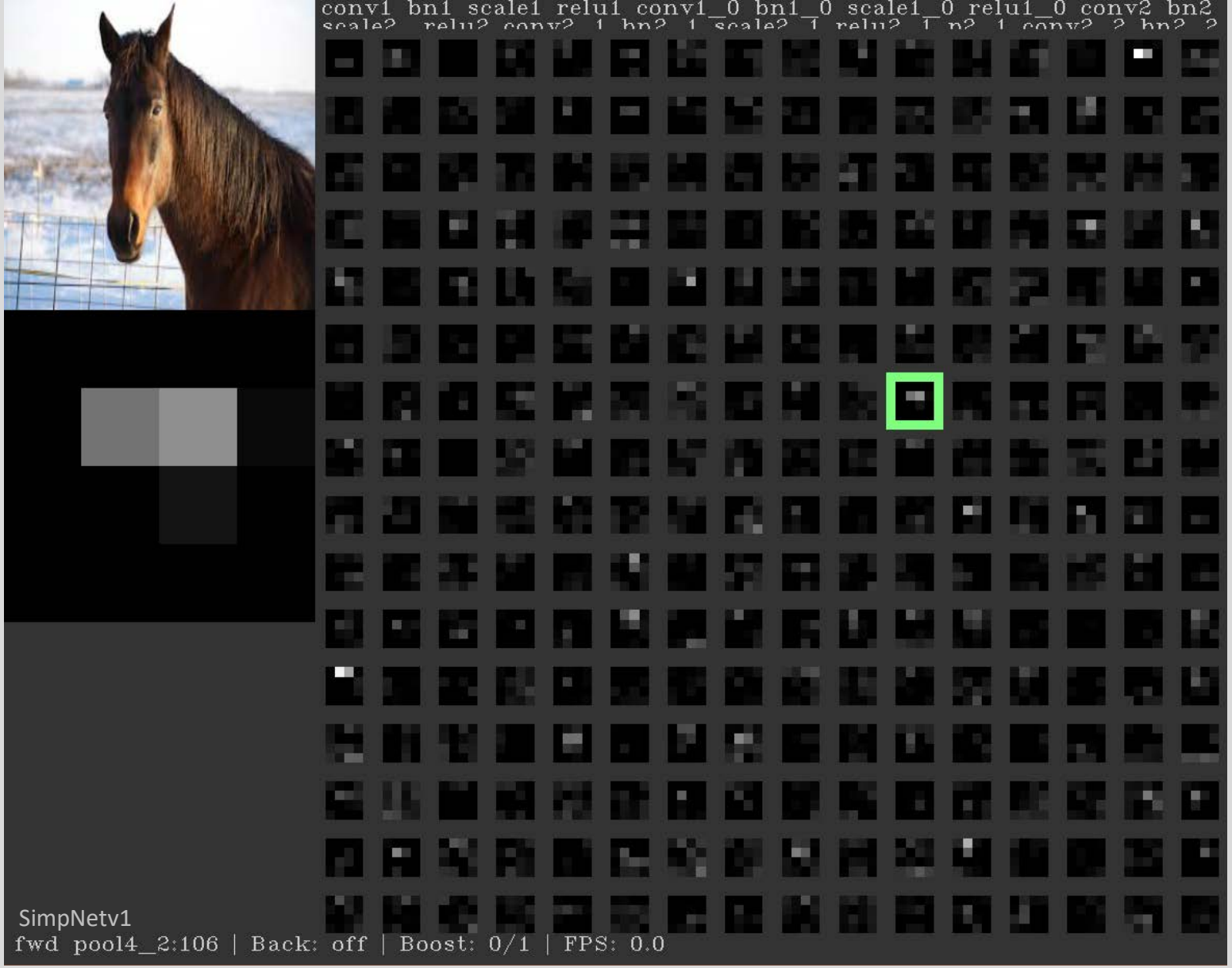


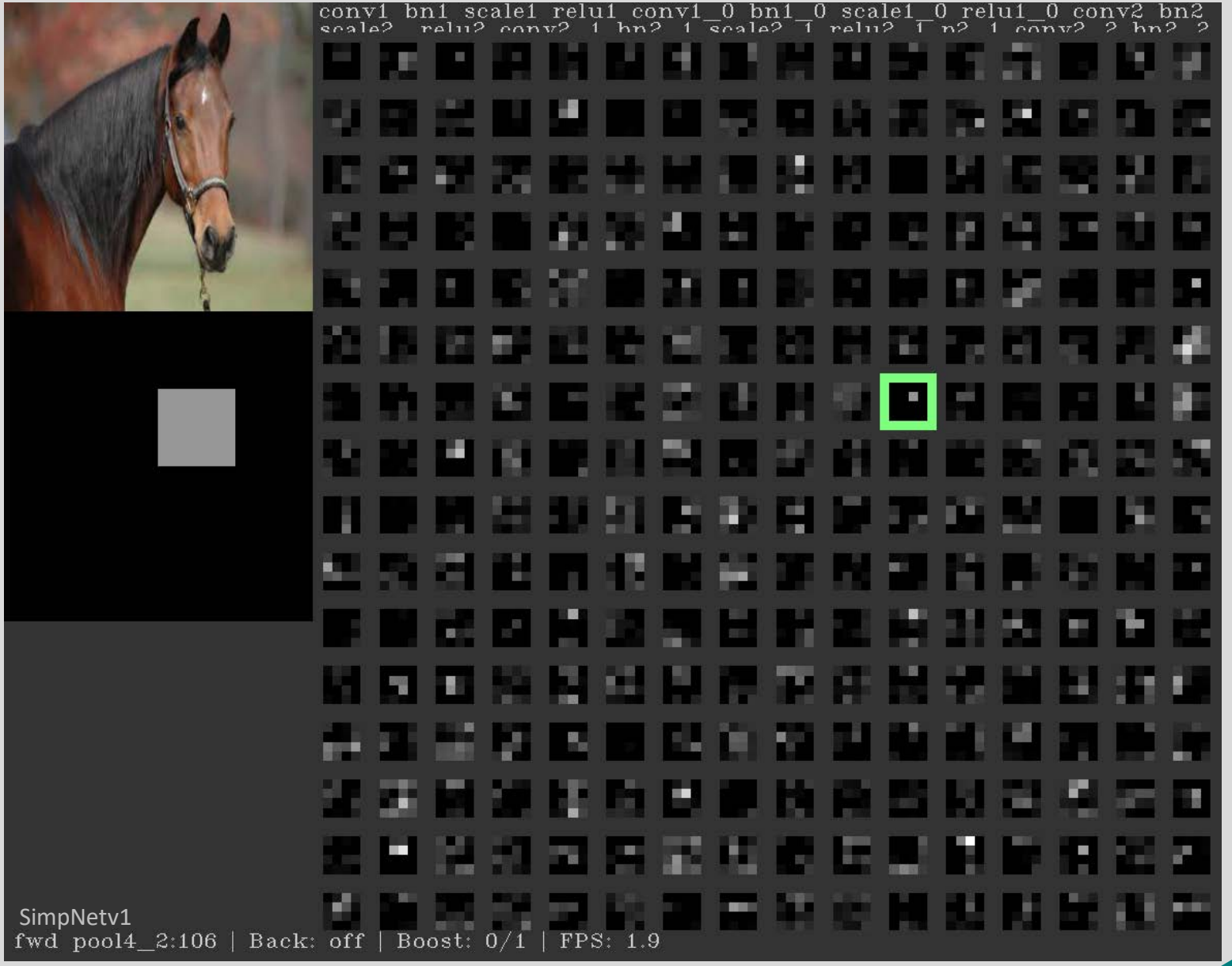

conv1 bn1 scale1 relu1 conv1_0 bn1_0 scale1_0 relu1_0 conv2 bn2
SimpNetv1
fwd pool4_2:106 | Back: off | Boost: 0/1 | FPS: 1.3

HAMIM©

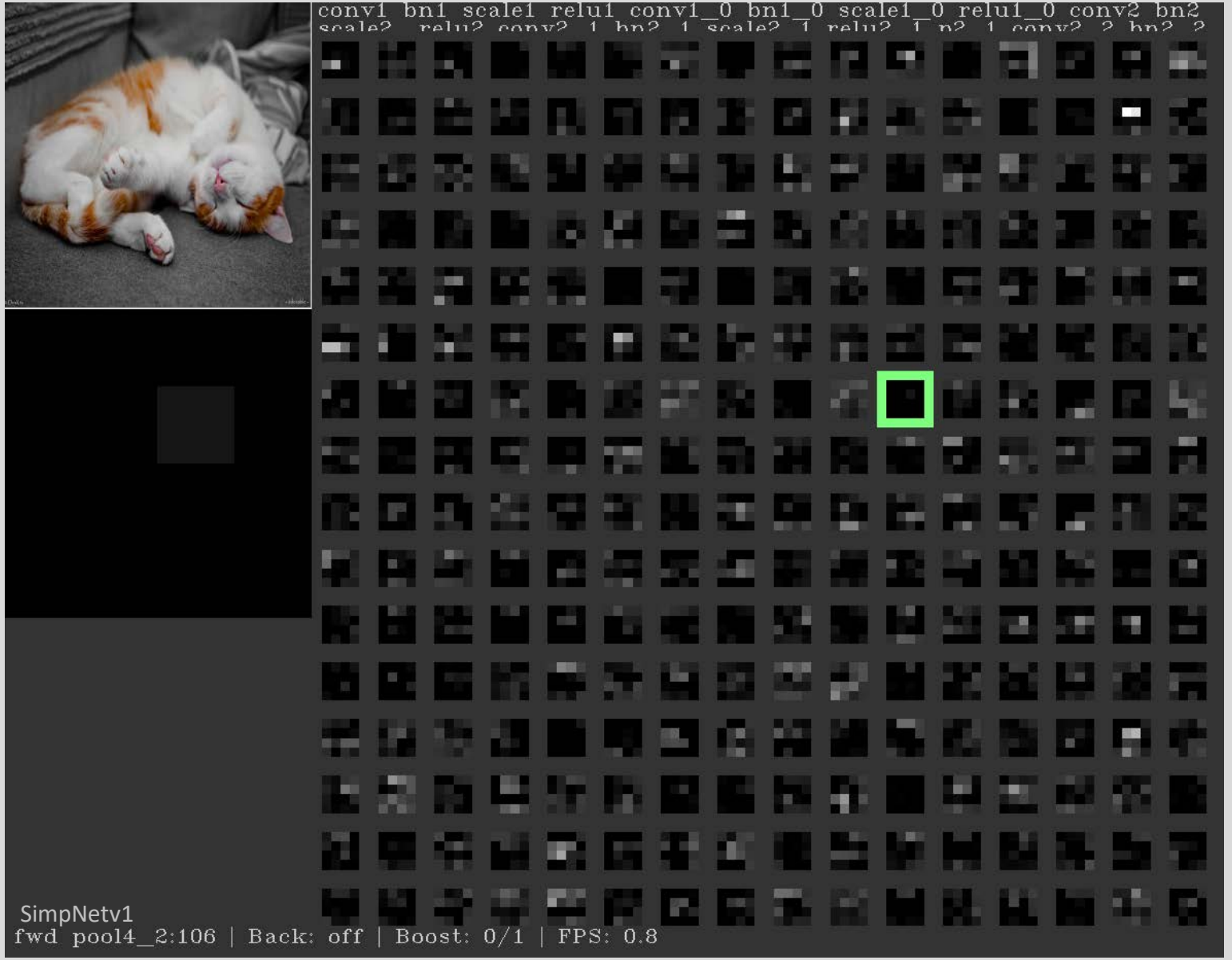

conv1 bn1 scale1 relu1 conv1_0 bn1_0 scale1_0 relu1_0 conv2 bn2
SimpNetv1
fwd pool4_2:106 | Back: off | Boost: 0/1 | FPS: 0.6

HAMIM©

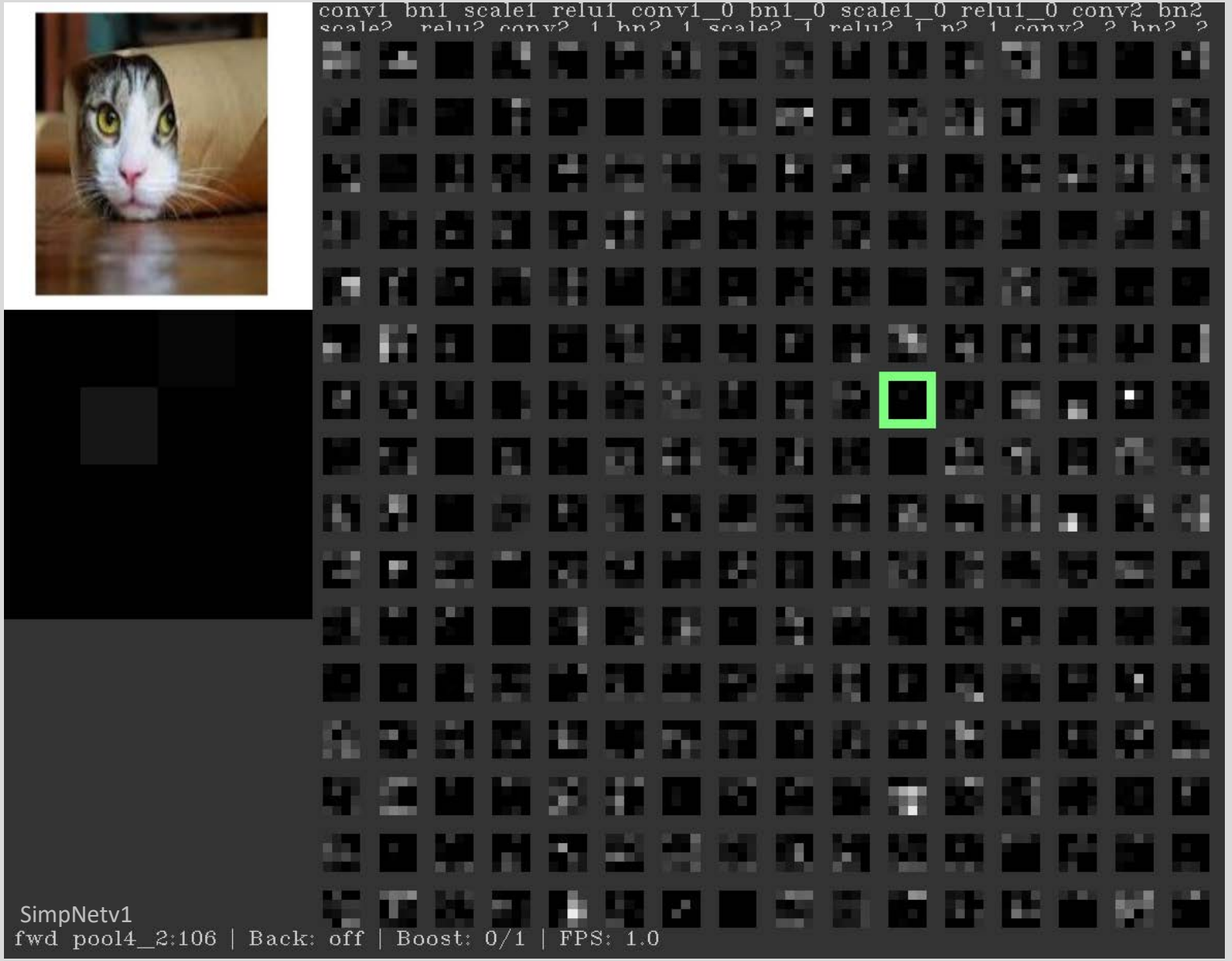

conv1 bn1 scale1 relu1 conv1_0 bn1_0 scale1_0 relu1_0 conv2 bn2
SimpNetv1
fwd pool4_2:106 | Back: off | Boost: 0/1 | FPS: 0.0


conv1 bn1 scale1 relu1 conv1_0 bn1_0 scale1_0 relu1_0 conv2 bn2
SimpNetv1
fwd pool4_2:106 | Back: off | Boost: 0/1 | FPS: 1.9

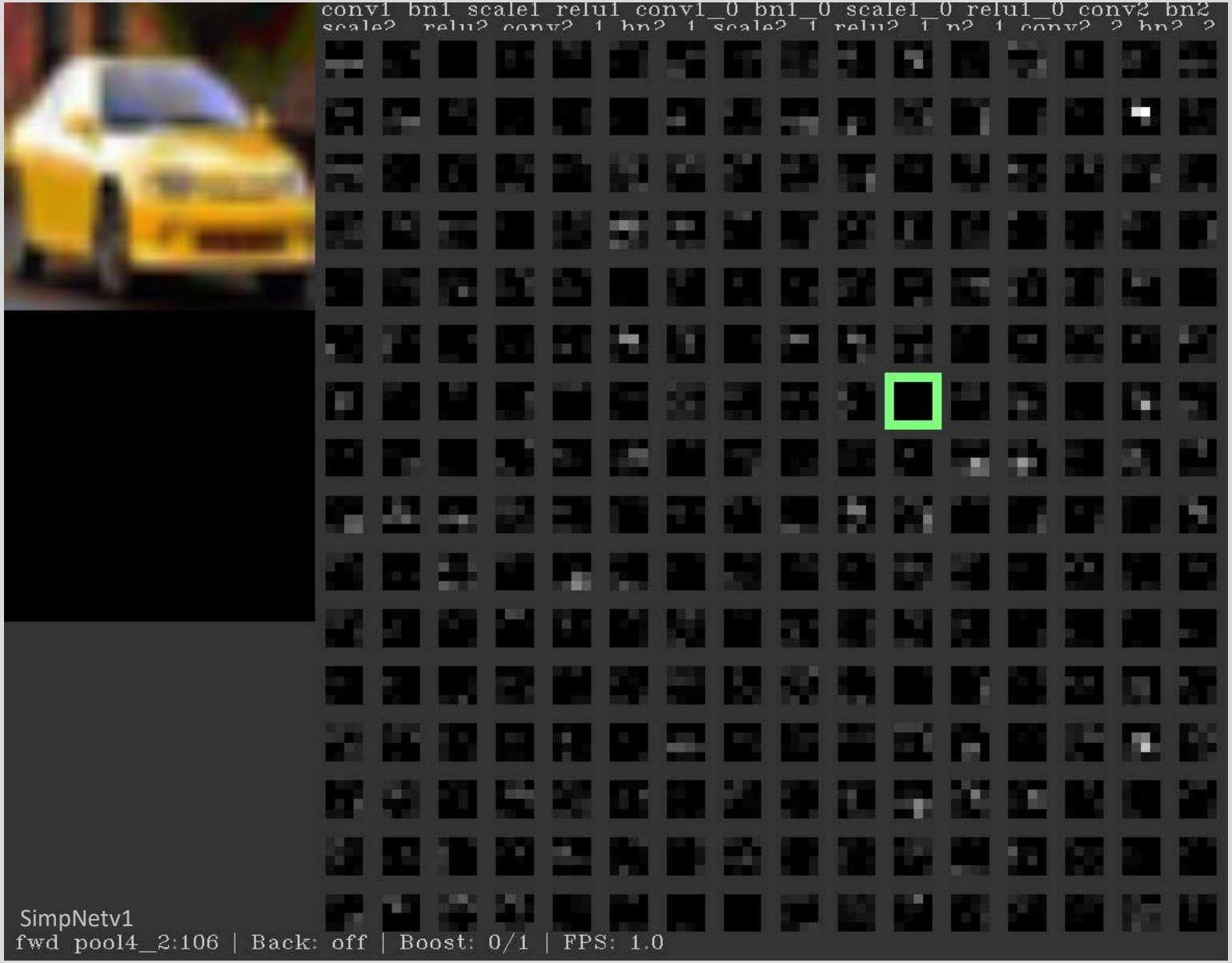
HAMIM©


conv1 bn1 scale1 relu1 conv1_0 bn1_0 scale1_0 relu1_0 conv2 bn2
SimpNetv1
fwd pool4_2:106 | Back: off | Boost: 0/1 | FPS: 0.8

HAMIM©

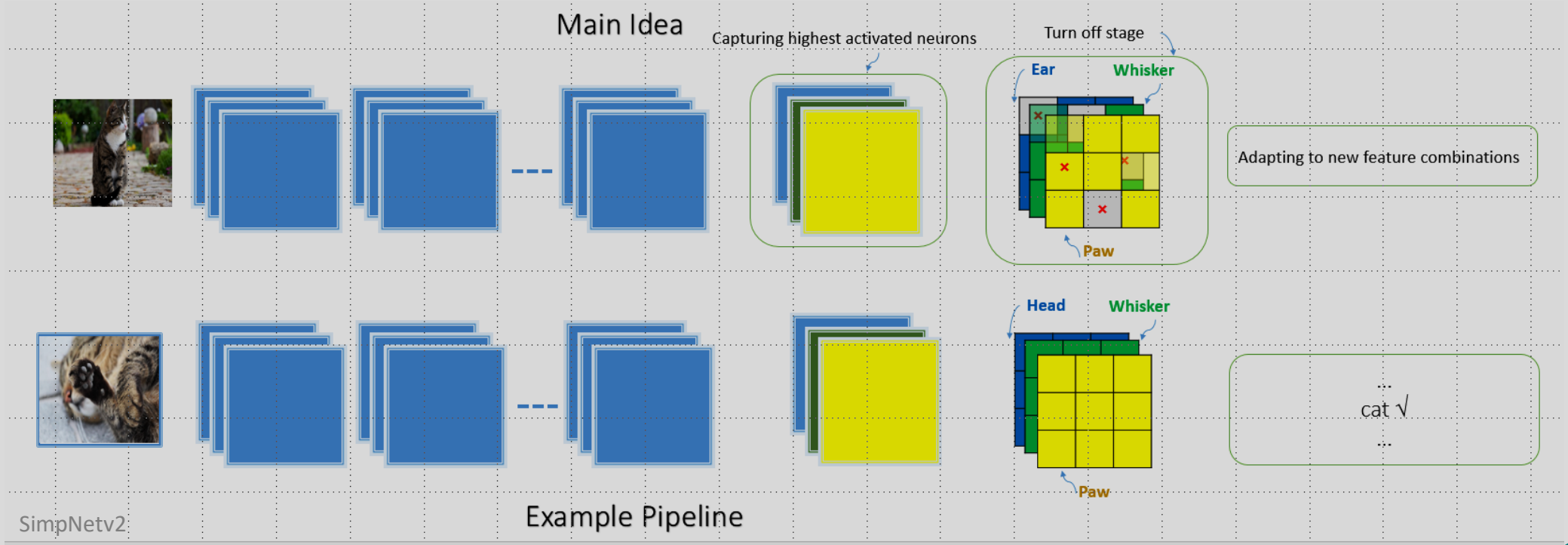

conv1 bn1 scale1 relu1 conv1_0 bn1_0 scale1_0 relu1_0 conv2 bn2
SimpNetv1
fwd pool4_2:106 | Back: off | Boost: 0/1 | FPS: 1.0

HAMIM©


conv1 bn1 scale1 relu1 conv1_0 bn1_0 scale1_0 relu1_0 conv2 bn2
SimpNetv1
fwd pool4_2:106 | Back: off | Boost: 0/1 | FPS: 1.0

HAMIM©

Simple Adaptive Feature Composition Pooling



HAMIM©

Simple Adaptive Feature Composition Pooling

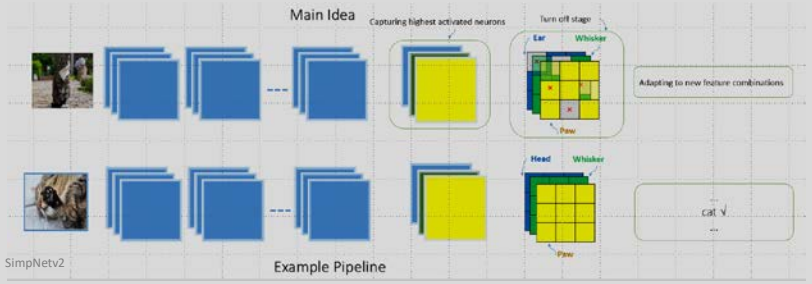
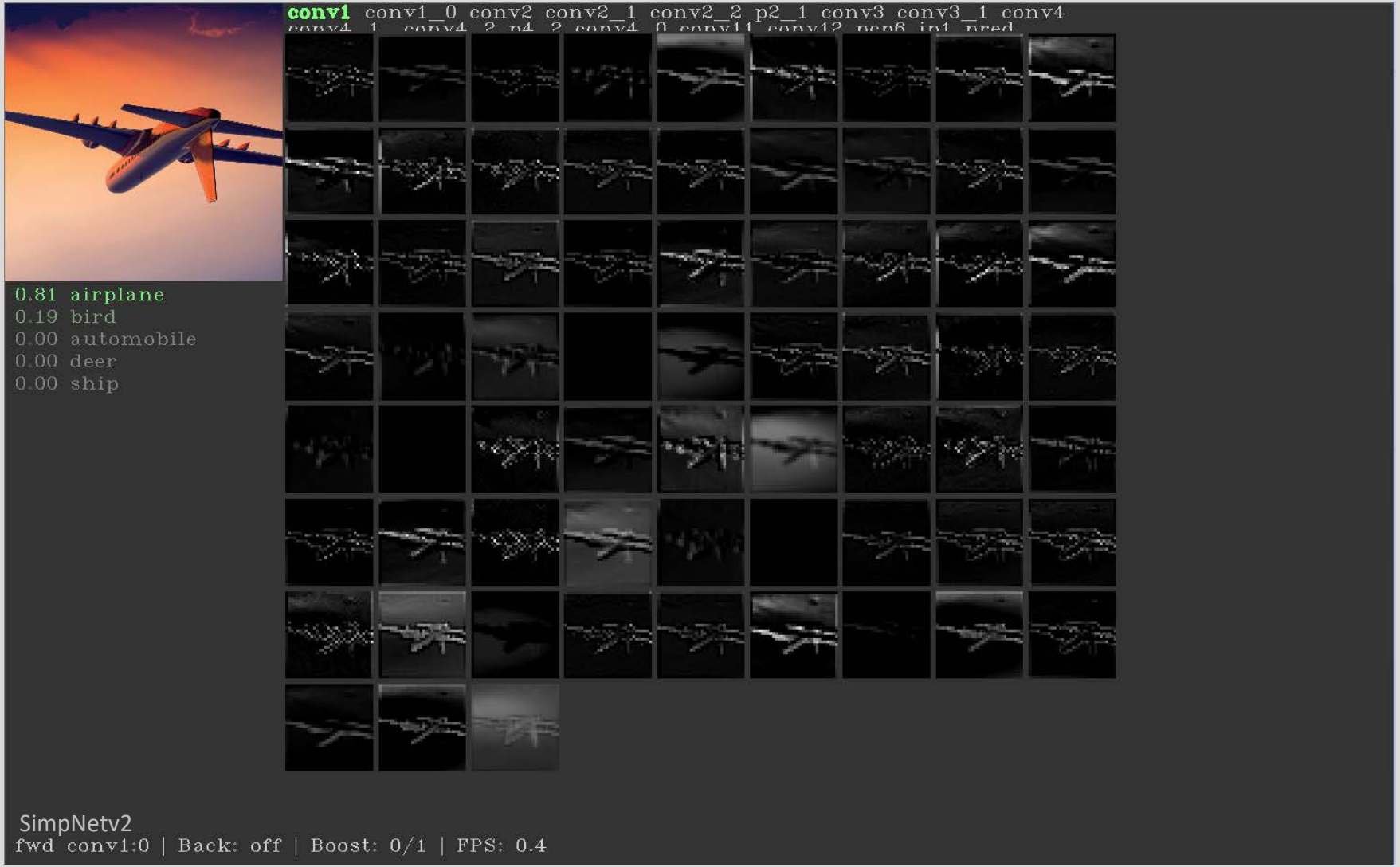
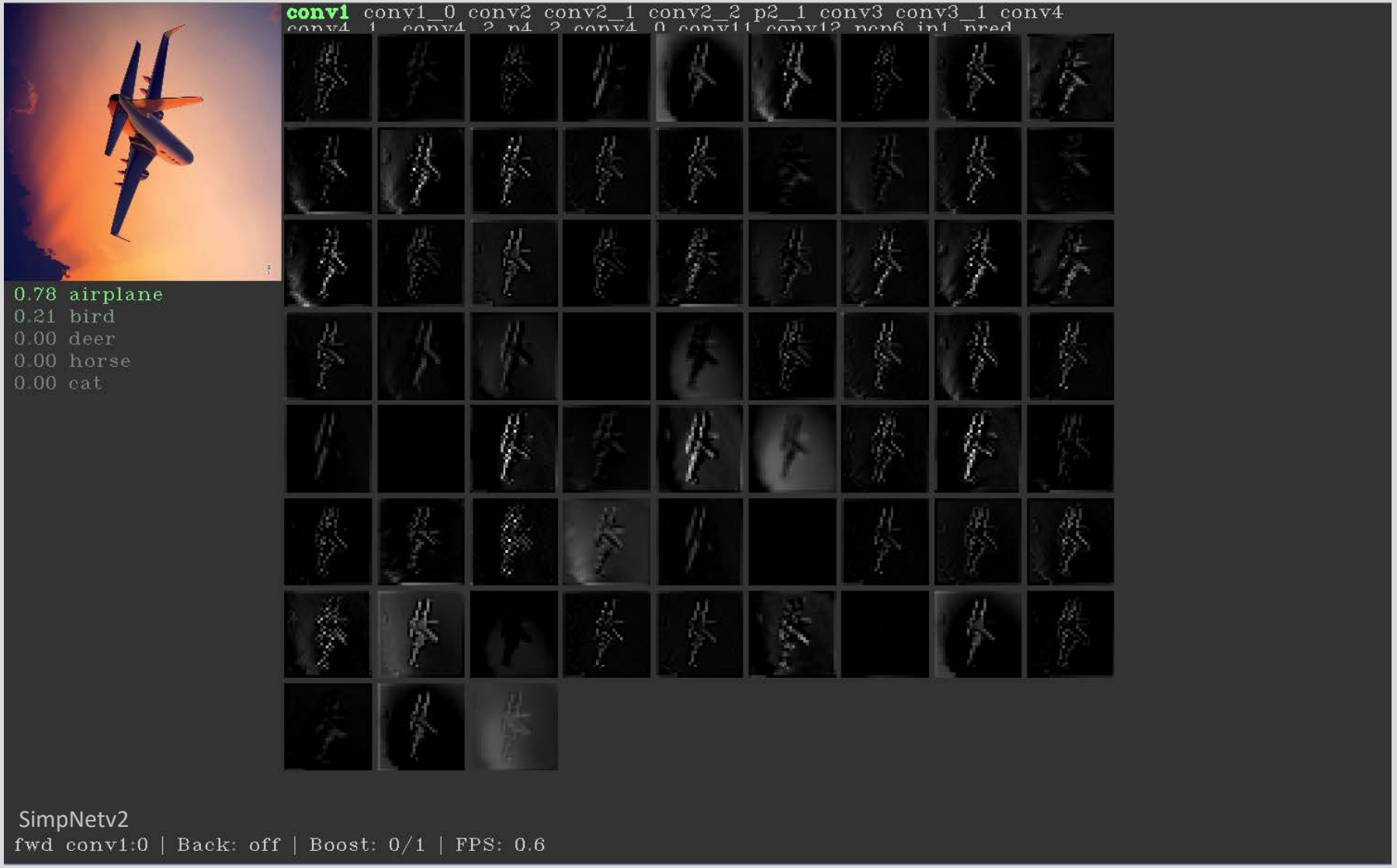


TABLE XI: Using *X-pooling* operation improves architecture performance. tests are run on CIFAR10

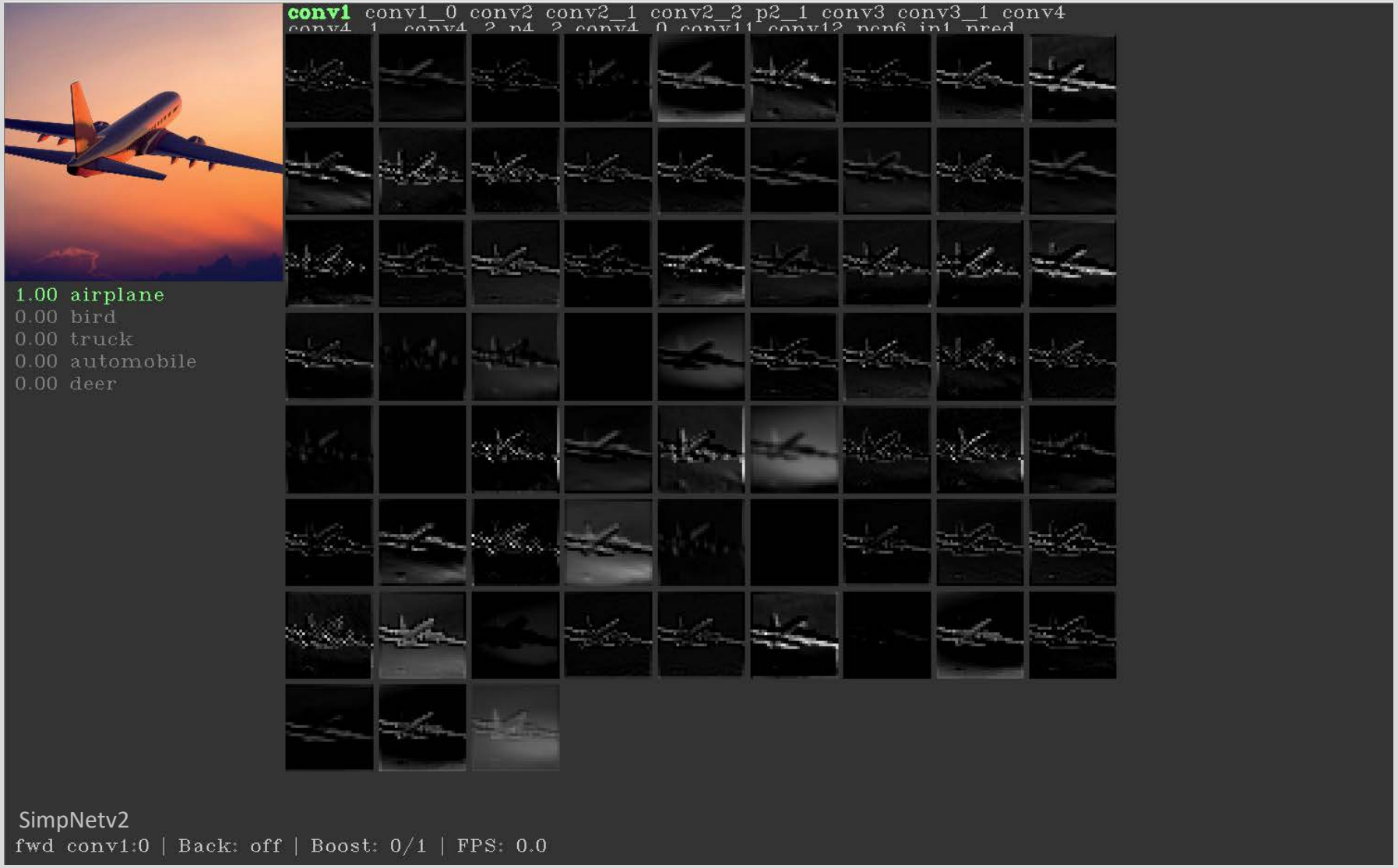
Network Properties	Accuracy (%) With–without PoolX
SqueezeNetv1.1	88.06(avg)–87.51(avg)
SimpNet	94.76–94.68


conv1 conv1_0 conv2 conv2_1 conv2_2 p2_1 conv3 conv3_1 conv4
0.99 airplane
0.01 bird
0.00 deer
0.00 horse
0.00 cat
SimpNetv2
fwd conv1:0 | Back: off | Boost: 0/1 | FPS: 1.0

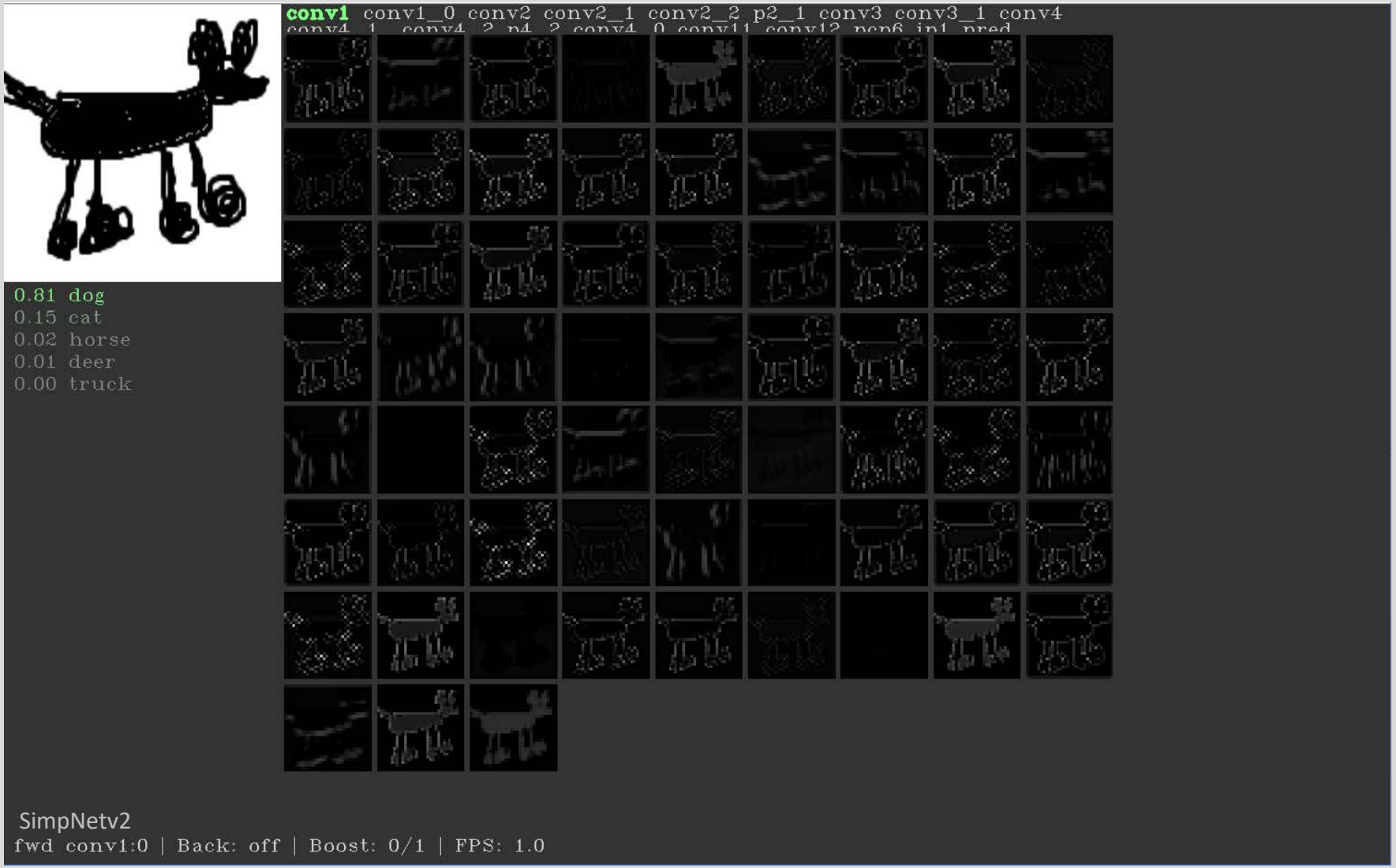

conv1 conv1_0 conv2 conv2_1 conv2_2 p2_1 conv3 conv3_1 conv4
0.81 airplane
0.19 bird
0.00 automobile
0.00 deer
0.00 ship
SimpNetv2
fwd conv1:0 | Back: off | Boost: 0/1 | FPS: 0.4


conv1 conv1_0 conv2 conv2_1 conv2_2 p2_1 conv3 conv3_1 conv4
0.78 airplane
0.21 bird
0.00 deer
0.00 horse
0.00 cat
SimpNetv2
fwd conv1:0 | Back: off | Boost: 0/1 | FPS: 0.6


conv1 conv1_0 conv2 conv2_1 conv2_2 p2_1 conv3 conv3_1 conv4
1.00 airplane
0.00 bird
0.00 deer
0.00 ship
0.00 truck
SimpNetv2
fwd conv1:0 | Back: off | Boost: 0/1 | FPS: 0.9

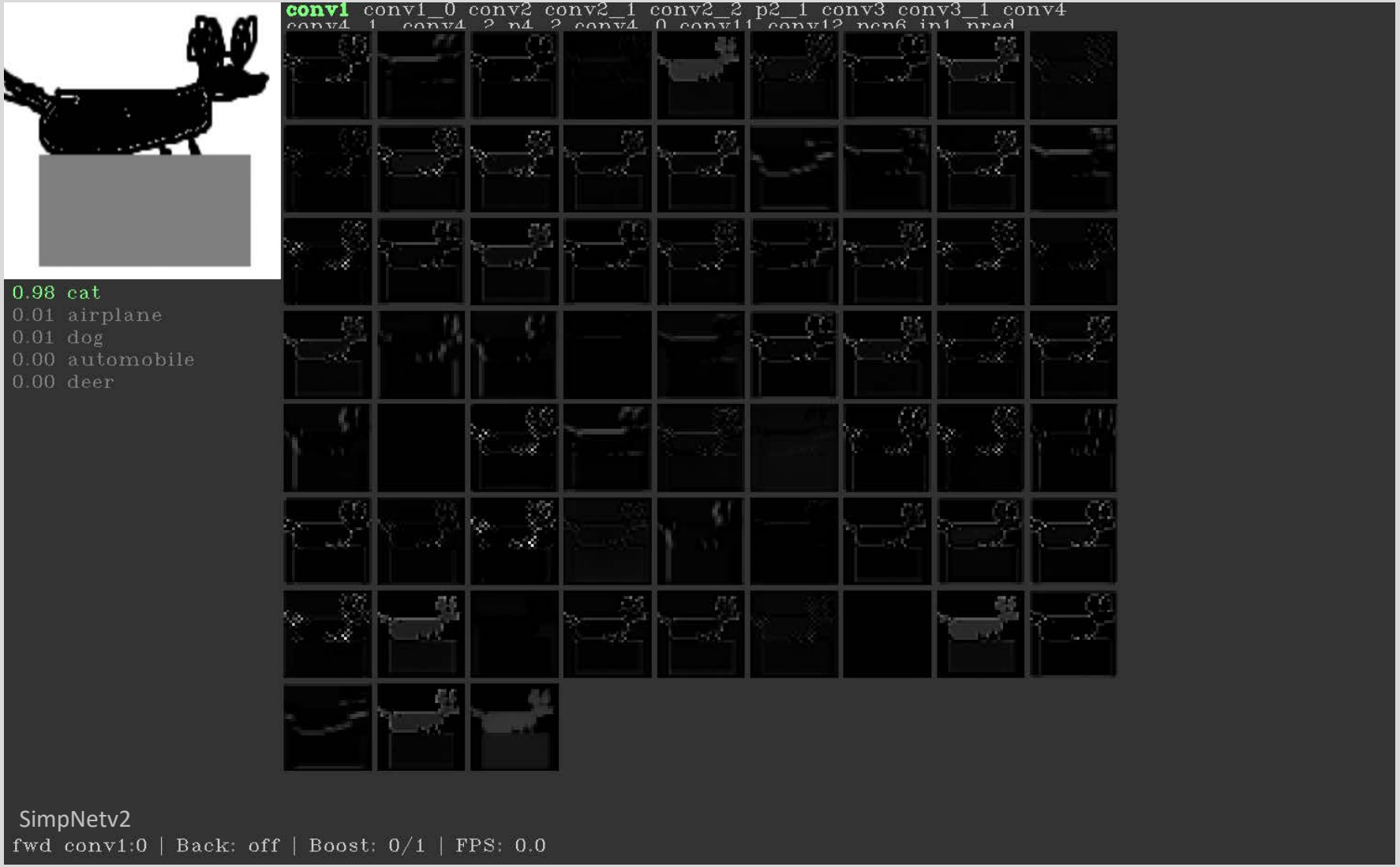

conv1 conv1_0 conv2 conv2_1 conv2_2 p2_1 conv3 conv3_1 conv4
1.00 airplane
0.00 bird
0.00 truck
0.00 automobile
0.00 deer
SimpNetv2
fwd conv1:0 | Back: off | Boost: 0/1 | FPS: 0.0


conv1 conv1_0 conv2 conv2_1 conv2_2 p2_1 conv3 conv3_1 conv4
0.63 cat
0.18 dog
0.08 horse
0.04 deer
0.03 automobile
SimpNetv2
fwd conv1:0 | Back: off | Boost: 0/1 | FPS: 0.4


conv1 conv1_0 conv2 conv2_1 conv2_2 p2_1 conv3 conv3_1 conv4
0.81 dog
0.15 cat
0.02 horse
0.01 deer
0.00 truck
SimpNetv2
fwd conv1:0 | Back: off | Boost: 0/1 | FPS: 1.0

HAMIM©

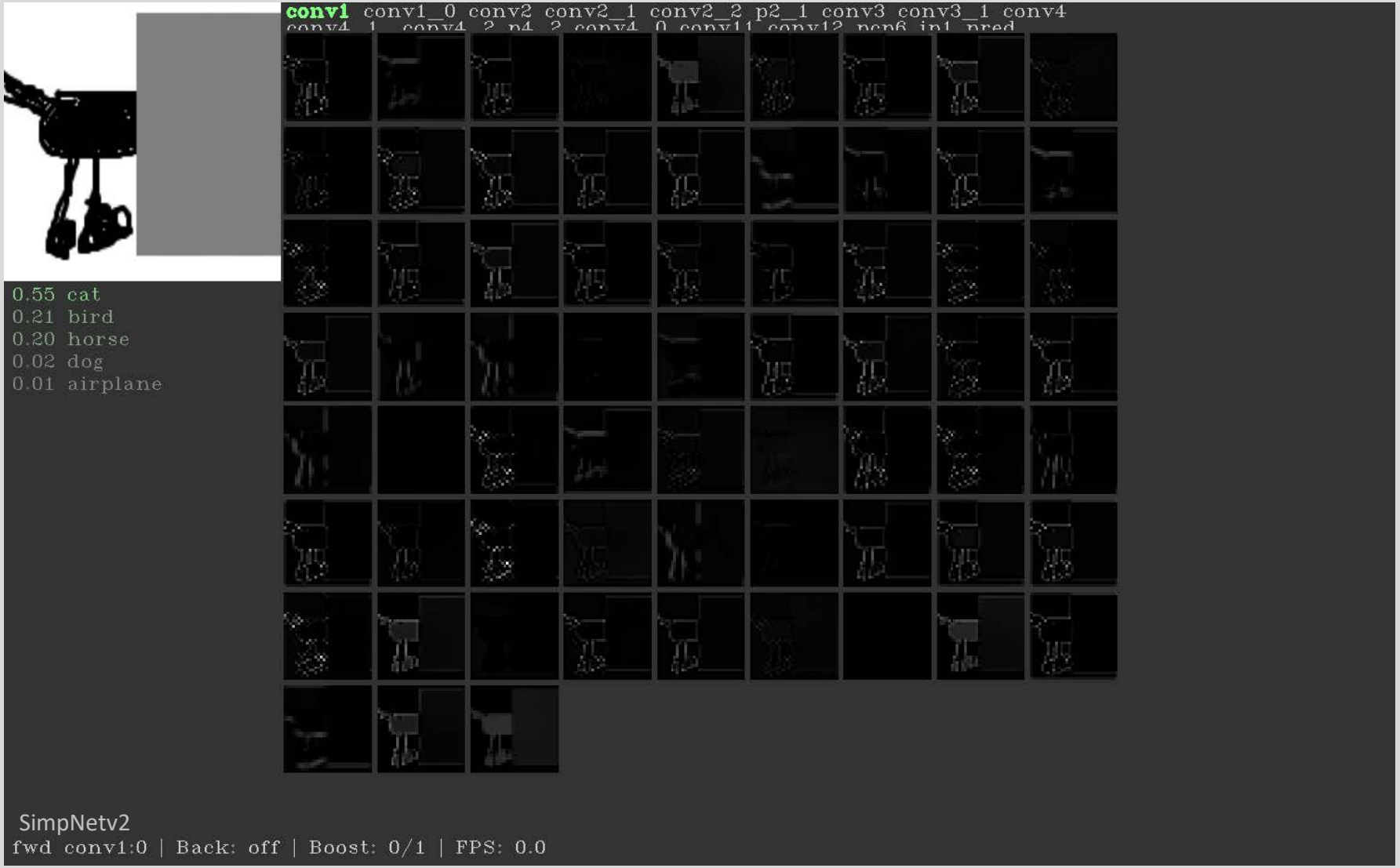

conv1 conv1_0 conv2 conv2_1 conv2_2 p2_1 conv3 conv3_1 conv4
0.74 cat
0.10 dog
0.09 horse
0.02 bird
0.02 airplane
SimpNetv2
fwd conv1:0 | Back: off | Boost: 0/1 | FPS: 0.9


conv1 conv1_0 conv2 conv2_1 conv2_2 p2_1 conv3 conv3_1 conv4
0.98 cat
0.01 airplane
0.01 dog
0.00 automobile
0.00 deer
SimpNetv2
fwd conv1:0 | Back: off | Boost: 0/1 | FPS: 0.0

HAMIM©


conv1 conv1_0 conv2 conv2_1 conv2_2 p2_1 conv3 conv3_1 conv4
0.97 truck
0.03 airplane
0.00 cat
0.00 horse
0.00 automobile
SimpNetv2
fwd conv1:0 | Back: off | Boost: 0/1 | FPS: 0.0

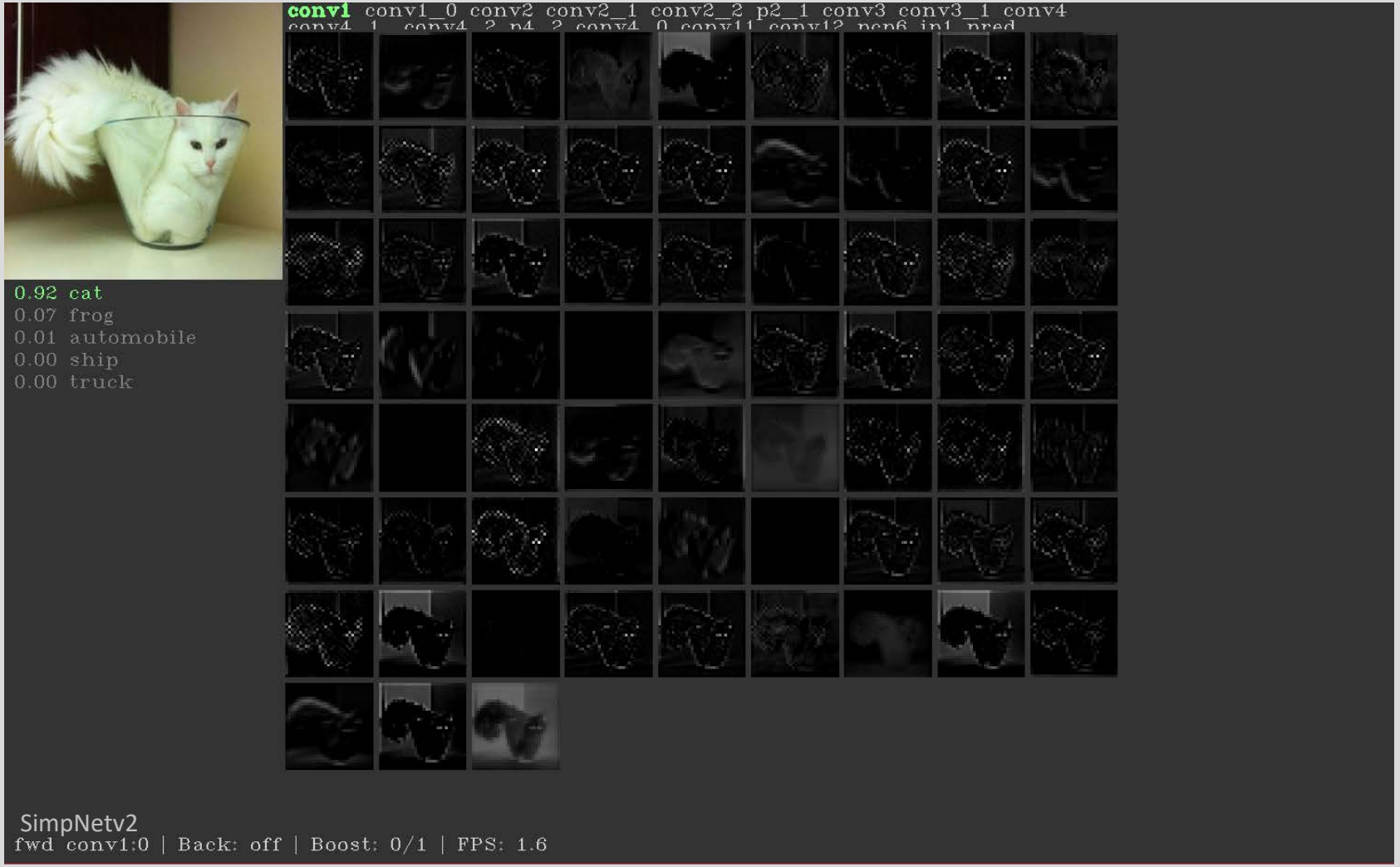
HAMIM©


conv1 conv1_0 conv2 conv2_1 conv2_2 p2_1 conv3 conv3_1 conv4
0.55 cat
0.21 bird
0.20 horse
0.02 dog
0.01 airplane
SimpNetv2
fwd conv1:0 | Back: off | Boost: 0/1 | FPS: 0.0

HAMIM©


conv1 conv1_0 conv2 conv2_1 conv2_2 p2_1 conv3 conv3_1 conv4
0.46 cat
0.36 horse
0.09 dog
0.07 deer
0.00 bird
SimpNetv2
fwd conv1:0 | Back: off | Boost: 0/1 | FPS: 0.0

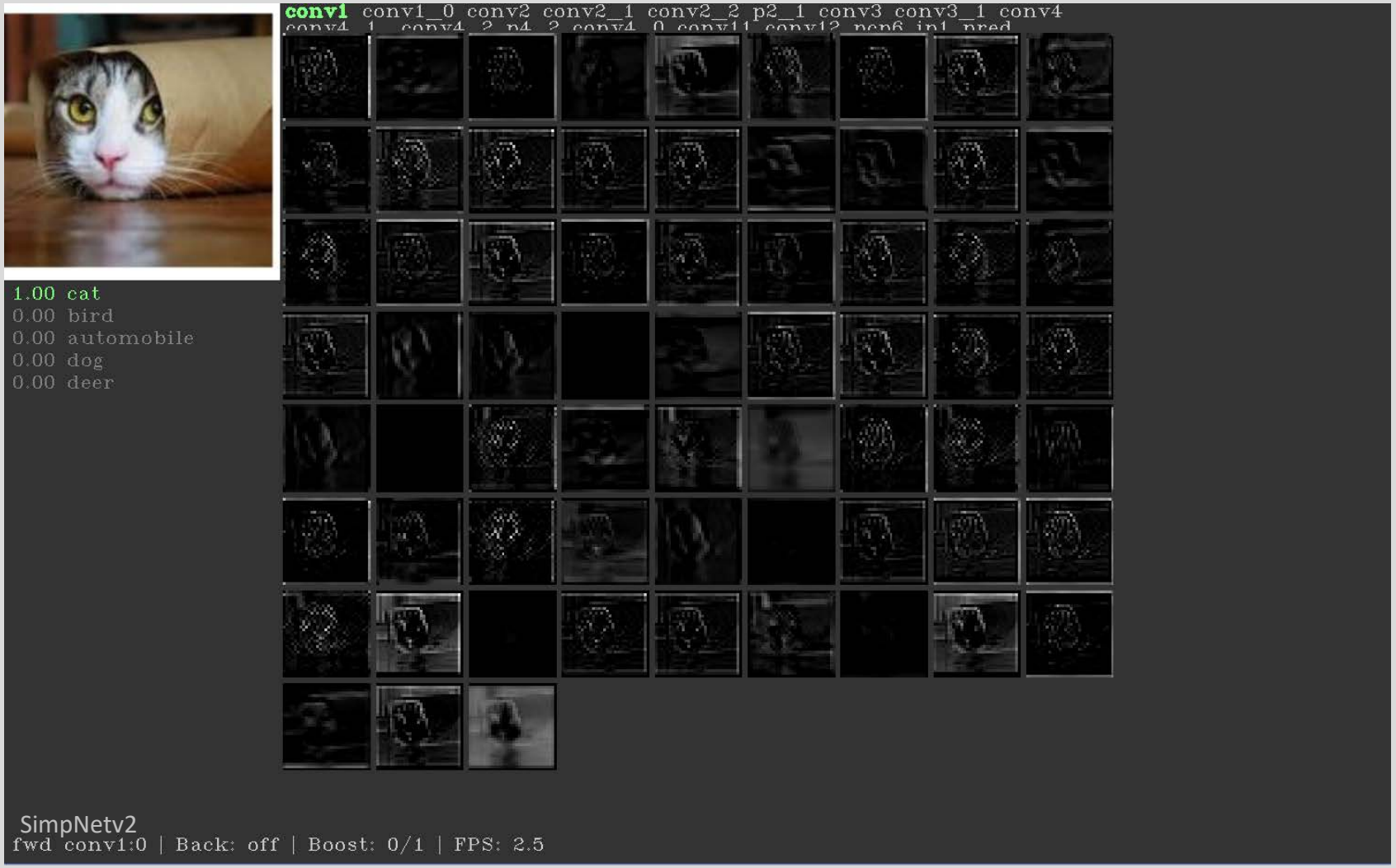
HAMIM©


conv1 conv1_0 conv2 conv2_1 conv2_2 p2_1 conv3 conv3_1 conv4
0.92 cat
0.07 frog
0.01 automobile
0.00 ship
0.00 truck
SimpNetv2
fwd conv1:0 | Back: off | Boost: 0/1 | FPS: 1.6

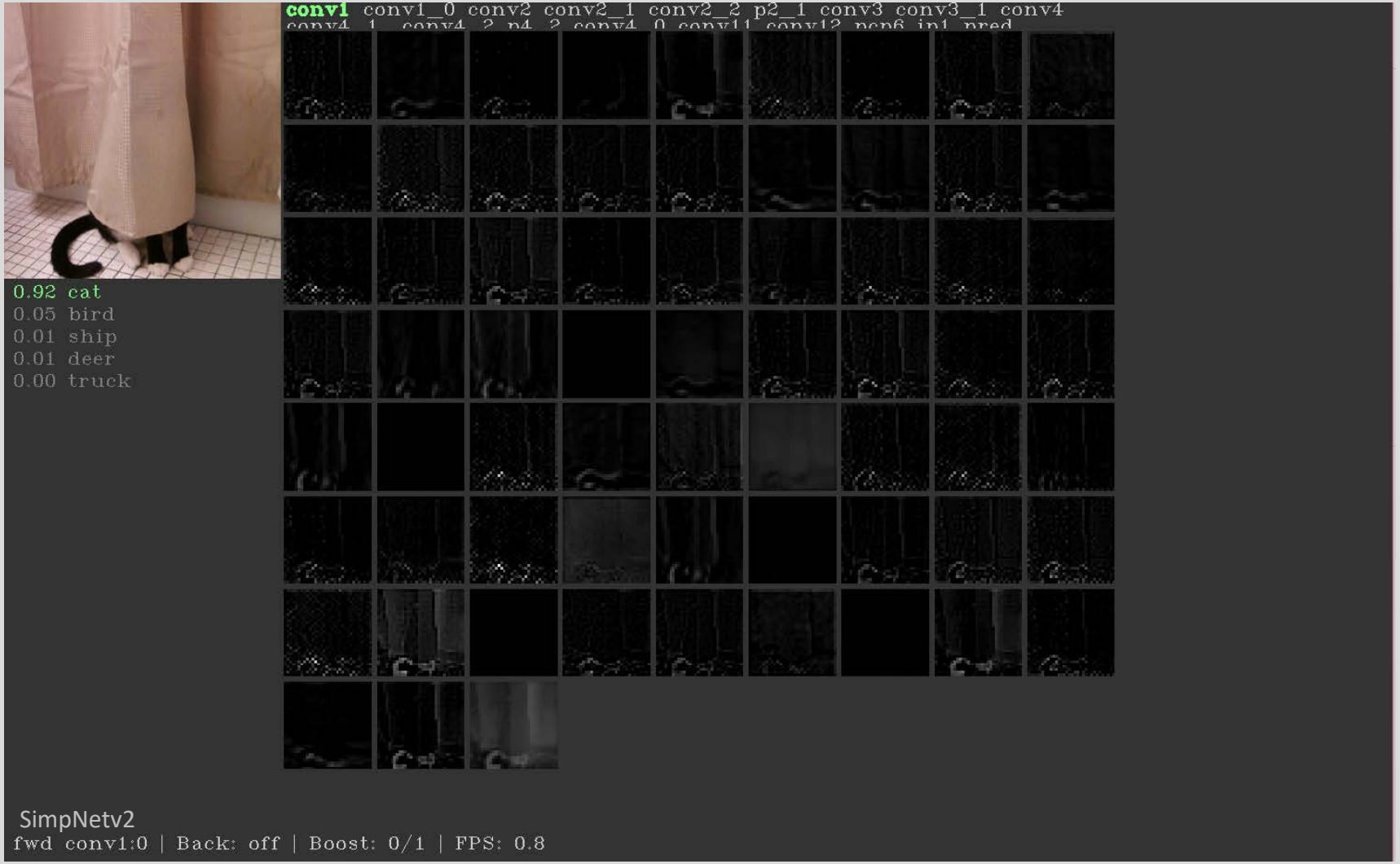
HAMIM©


conv1 conv1_0 conv2 conv2_1 conv2_2 p2_1 conv3 conv3_1 conv4
0.74 cat
0.13 frog
0.12 bird
0.00 truck
0.00 airplane
SimpNetv2
fwd conv1:0 | Back: off | Boost: 0/1 | FPS: 0.4

HAMIM©


conv1 conv1_0 conv2 conv2_1 conv2_2 p2_1 conv3 conv3_1 conv4
1.00 cat
0.00 bird
0.00 automobile
0.00 dog
0.00 deer
SimpNetv2
fwd conv1:0 | Back: off | Boost: 0/1 | FPS: 2.5

HAMIM©


conv1 conv1_0 conv2 conv2_1 conv2_2 p2_1 conv3 conv3_1 conv4
0.92 cat
0.05 bird
0.01 ship
0.01 deer
0.00 truck
SimpNetv2
fwd conv1:0 | Back: off | Boost: 0/1 | FPS: 0.8


conv1 conv1_0 conv2 conv2_1 conv2_2 p2_1 conv3 conv3_1 conv4
1.00 cat
0.00 dog
0.00 airplane
0.00 horse
0.00 truck
SimpNetv2
fwd conv1:0 | Back: off | Boost: 0/1 | FPS: 0.0

HAMIM©

Rapid prototyping

- Most of the time it’s the optimization policy that needs to be changed
- Run several tests in order to account for different initialization values and also different optimization values.
- Start with automated ones then fine-tune your architecture, after that use other policies and work on optimization policy

Final Stage :

- You don’t need to follow all the rules exclusively , start with the principles and then tailor the architecture according to your needs, if you need less memory consumption, try applying more pooling or at least earlier in the network for example.
- Or use 1x1 at the end, to decrease computation overhead
- After all, all of these principles are here to give you intuitions about different criterion and their effects on your problem.

Dark Knowledge / knowledge Distillation

- Teacher-student paradigm
  - Dark Knowledge
  - Net2Net

Dark Knowledge / knowledge Distillation

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$$

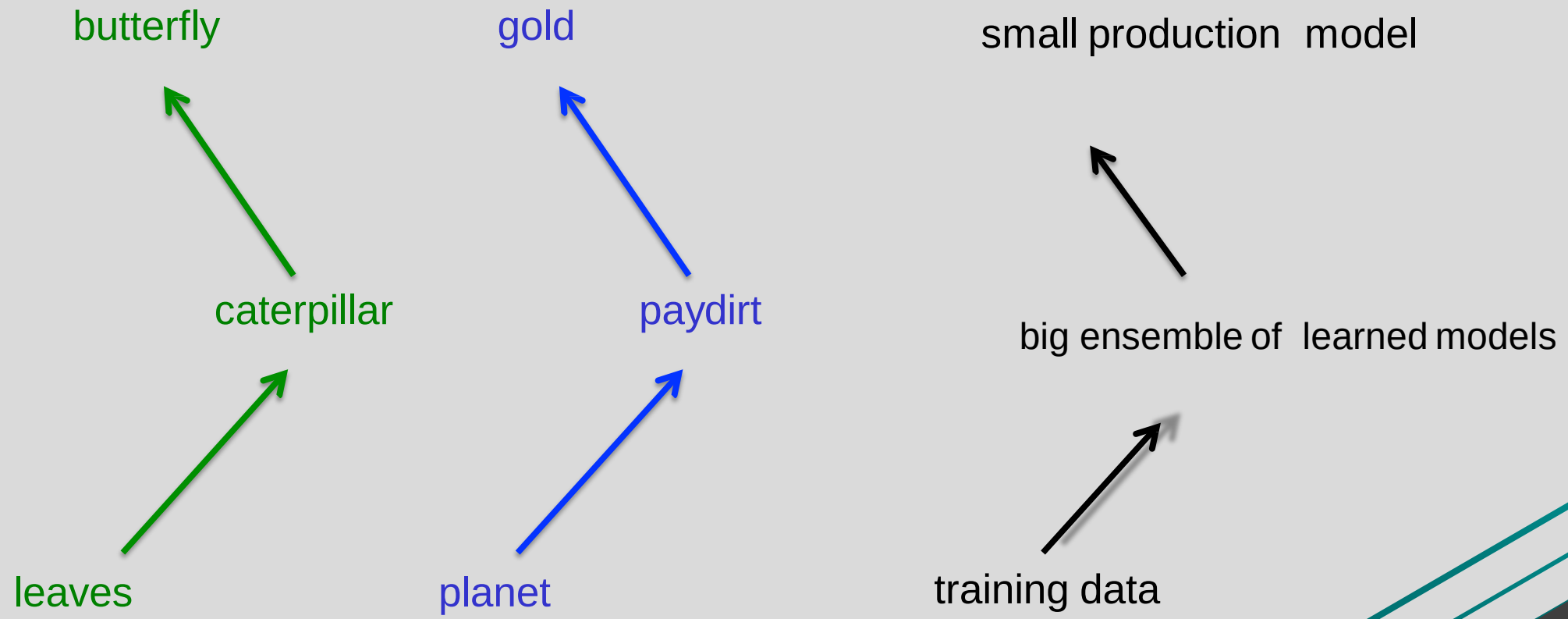
The conflicting constraints of learning and using

- The easiest way to extract a lot of knowledge from the training data is to learn many different models in parallel.
  - We want to make the models as different as possible to minimize the correlations between their errors.
  - We can use different initializations or different architectures or different subsets of the training data.
  - It is helpful to over-fit the individual models.
- A test time we average the predictions of all the models or of a selected subset of good models that make different errors.
  - That's how almost all ML competitions are won (e.g. Netflix)

Why ensembles are bad at test time

- A big ensemble is highly redundant. It has very very little knowledge per parameter.
- At test time we want to minimize the amount of computation and the memory footprint.
  - These constraints are generally much more severe at test time than during training.

An analogy



The main idea

- The ensemble implements a function from input to output. Forget the models in the ensemble and the way they are parameterized and focus on the function.
  - After learning the ensemble, we have our hands on the function.
  - Can we transfer the knowledge in the function into a single smaller model?
- Caruana et. al. 2006 had the same idea but used a different way of transferring the knowledge.

Soft targets: A way to transfer the function

- If the output is a big N-way softmax, the targets are usually a single 1 and a whole lot of 0's.
  - On average each target puts at most log N bits of constraint on the function.
- If we have the ensemble, we can divide the averaged logits from the ensemble by a "temperature" to get a much softer distribution.

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$$

This reveals **much more information** about the function on each training case.

An example of hard and soft targets

cow	dog	cat	car	
0	1	0	0	original hard targets

cow	dog	cat	car	
10^{-6}	.9	.1	10^{-9}	output of geometric ensemble

cow	dog	cat	car	
.05	.3	.2	.005	softened output of ensemble

Softened outputs reveal the dark knowledge in the ensemble.

Adding in the true targets

- If we just train the final model on the soft targets from the ensemble, we do quite well.
- We learn fast because each training case imposes much more constraint on the parameters than a single hard target.
- But it works better to fit both the hard targets and the soft targets from the ensemble.

How to add hard targets during distillation

- We try to learn logits in the distilled model that minimize the sum of two different cross entropies.
- Using a high temperature in the softmax, we minimize the cross entropy with the soft targets derived from the ensemble at high temperature.
- Using the very same logits at a temperature of 1, we minimize the cross entropy with the hard targets.

Relative weighting of the hard and soft cross entropies

- The derivatives for the soft targets tend to be much smaller.
  - They also have much less variance from case to case.
- So we down-weight the cross entropy with the hard targets.
  - Even though its down-weighted, this extra term is important for getting the best results.

Experiment on MNIST

- Vanilla backprop in a 784 -> 800 -> 800 -> 10 net with rectified linear hidden units gives 146 test errors.

  RELU: y = max(0, x)

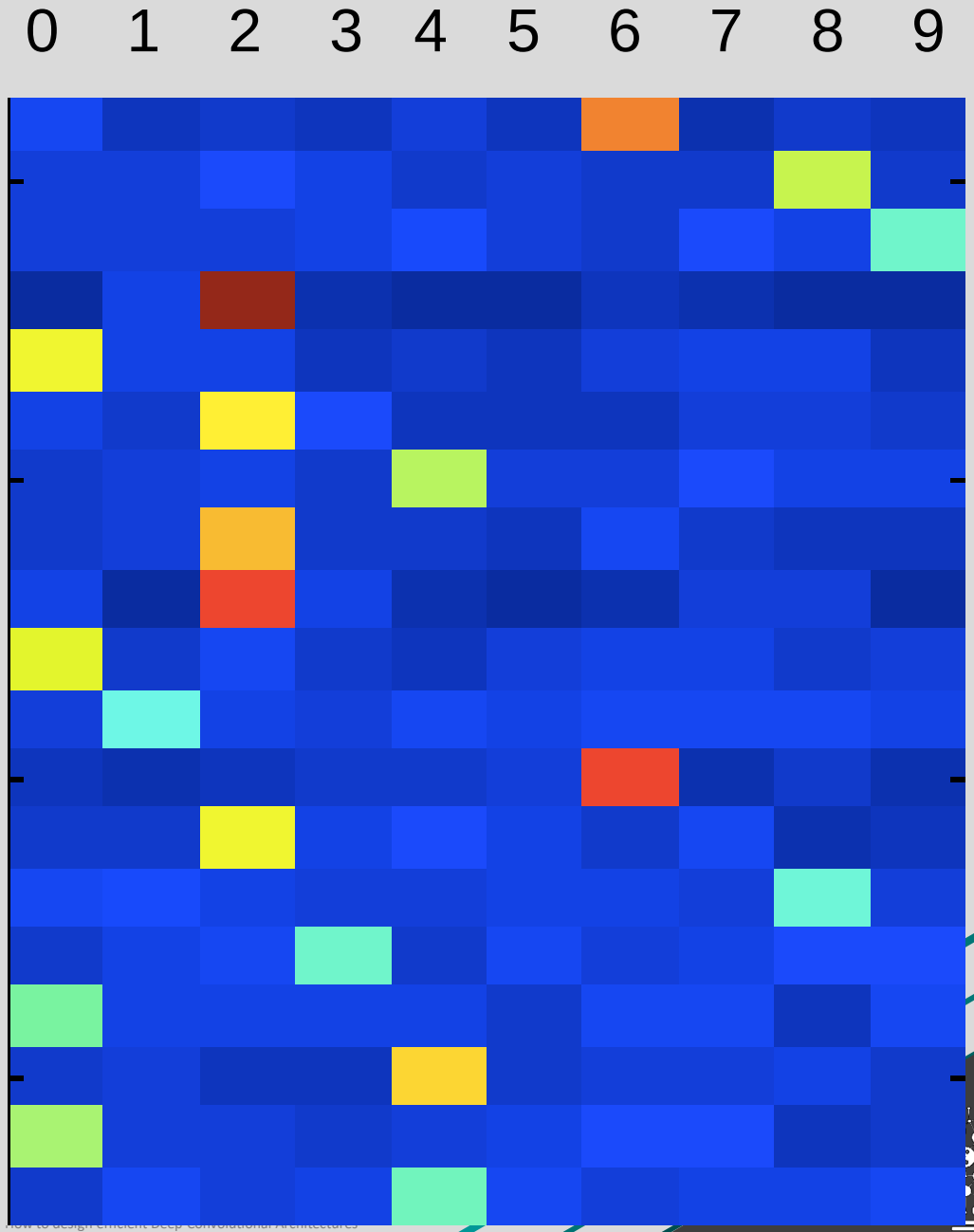
- If we train a 784 -> 1200 -> 1200 -> 10 net using dropout and weight constraints and jittering the input, we eventually get 67 errors.
- How much of this improvement can be transferred to the 784 -> 800 -> 800 -> 10 net?

Transfer to the small net

- Using both the soft targets obtained from the big net and the hard targets, we get 74 errors in the 784 -> 800 -> 800 -> 10 net.
  - The transfer training uses the same training set but with no dropout and no jitter.
  - Its just vanilla backprop (with added soft targets).
- The soft targets contain almost all the knowledge.
  - The big net learns a similarity metric for the training digits even though this isn't the objective function for learning.

The soft outputs

(one row per training case)



this 2 resembles a 1
and nothing much else

this 2 resembles
0, 3, 7, 8

this 2 resembles
4 and 7

A very surprising result on MNIST

- Train the 784 -> 800 -> 800 -> 10 net on a transfer set that does not contain any examples of a 3. After this training, raise the bias of the 3 by the right amount.
  - The distilled net then gets 98.6% of the test threes correct even though it never saw any threes during the transfer training.

An even more surprising result on MNIST

- Train the 784 -> 800 -> 800 -> 10 net on a transfer set that only contains images of 7 and 8.
- After training, lower the biases of 7 and 8 by the optimal amount.
- The net then gets 87% correct over all classes.

Conclusion so far

- It is well known that object recognition is greatly improved by transforming the input images in ways that do not change the label.
  - But this brute-force method means we need to train on a lot more images.
- Transforming the targets has similarly big effects on generalization.
  - This does not change the size of the training set.
  - But you have to get the soft targets from somewhere.

A popular way to transform the targets

- Organize the labels into a tree and instead of just predicting a label, predict all of the labels on the path to the root.
  - Many groups have tried this and it helps, but not nearly as much as using soft targets produced by a good model.
- Visual similarity cannot be modeled well by any tree.

How to make an ensemble mine knowledge more efficiently

- We can encourage different members of the ensemble to focus on resolving different confusions.
  - In ImageNet, one “specialist” net could see examples that are enriched in mushrooms.
  - Another specialist net could see examples enriched in sports cars.
- We can choose the confusable classes in several ways.
  - K-means clustering on the soft target vectors produced by a generalist model works nicely.

The main problem with specialists

- Specialists tend to over-fit.
- To prevent this we need a very strong regularizer.
  - Making them small doesn't work well. They need all the lower levels of a general vision system to make the right fine distinctions.
  - Freezing the lower levels does not work well.The early filters need to be slightly adapted.
- So how can we regularize specialists effectively without making them too weak?

One way to prevent specialists over-fitting

- Each specialist uses a reduced softmax that has one dustbin class for all the classes it does not specialize in.
- The specialist estimates two things:
  - 1. Is this image in my special subset?
  - 2. What are the relative probabilities of the classes in my special subset?
- After training we can adjust the logit of the dustbin class to allow for the data enrichment.
- The specialist is initialized with the weights of a previously trained generalist model and uses early stopping to prevent over-fitting.

The JFT dataset

- This is a Google internal dataset with about 100 million images with 15,000 different class labels.
- A large convolutional neural net trained for about six months on many machines gets 25% correct on the test set (using top-1 criterion).
- Can we improve this significantly with only a few weeks of training?

Early stopping specialists on JFT

- Start from JFT model that gets 25% top-1 correct.

#spec	#cases	#win	relative accuracy
0	350037	0	0.0%
1	141993	+1421	+3.4%
2	67161	+1572	+7.4%
3	38801	+1124	+8.8%
4	26298	+835	+10.5%
5	16474	+561	+11.1%
6	10682	+362	+11.3%
7	7376	+232	+12.8%
8	4703	+182	+13.6%
9	4706	+208	+16.6%
10+	9082	+324	+14.1%

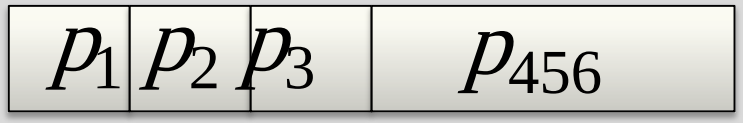
Combining models that have dustbin classes

- Its not trivial. A specialist is NOT claiming that everything in its dustbin class is equally probable. Its making a claim about the sum of those probabilities.
- Basic idea: For each test case, we iteratively revise the logits for the detailed classes to try to agree with all of the specialists.
  - i.e. We try to make the sum of the relevant detailed probabilities match the dustbin probability.

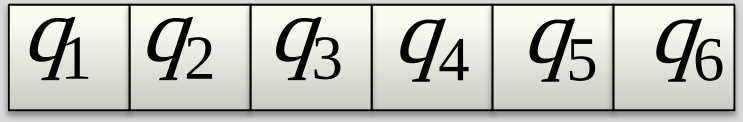
A picture of how to combine models that each have a dustbin class

- For each test or transfer case we run a fast iterative loop to find the set of logits that fit best

  with the partial distributions produced by the trained specialists.



target probs from a specialist



actual probs of combination

A better way to prevent specialists over-fitting?

- Each specialist gets data that is very enriched in its particular subset of classes but its softmax covers all of the classes.
- On data from its special subset (50% of its training cases) it just tries to fit the hard targets with T=1.
- On the remaining data it just tries to match the soft targets produced by a previously trained generalist model at high temperature.
  - The soft targets will prevent overfitting.
  - Remember the 3% effect in the speech experiment.

Conclusion

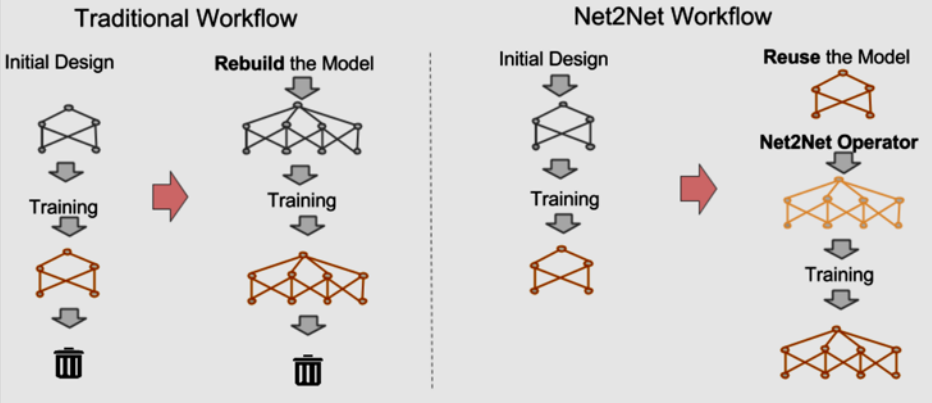
- When extracting knowledge from data we do not need to worry about using very

  big models or very big ensembles of models that are much too cumbersome to deploy.

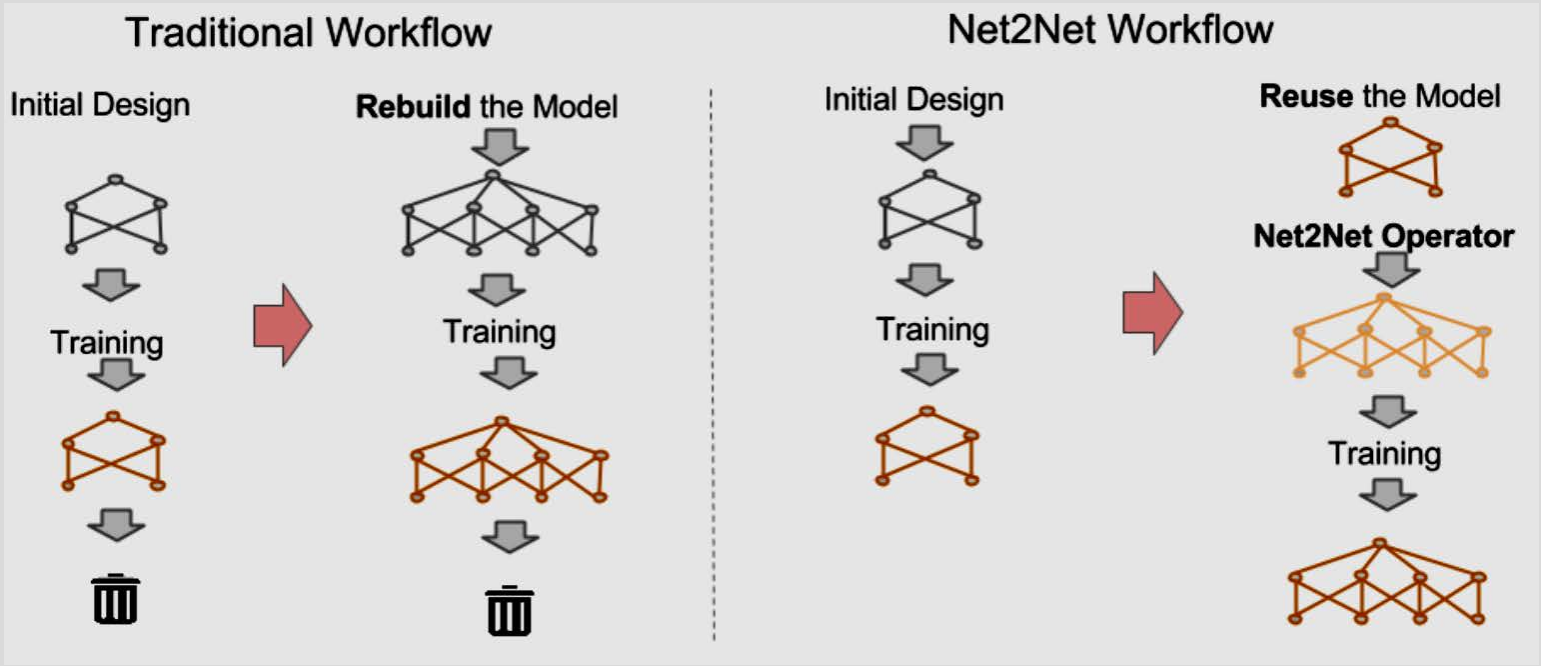
  - If we can extract the knowledge from the data it is quite easy to distill most of it into a much smaller model for deployment.
- Speculation: On really big datasets, ensembles of specialists should be more efficient at extracting the knowledge.
  - Soft targets for their non-special classes can be used to prevent them from over-fitting.

Net2Net

- Main Contributions:
  - Net2Deeper
    - Initialize layers with identity weight matrices to preserve the same output.
    - Only works when activation function f satisfies f(If(x)) = f(x) for example ReLU, but not sigmoid, Tanh.
  - Net2Wider
  - Additional units in a layer are randomly sampled from existing units. Incoming weights are kept the same while outgoing weights are divided by the number of replicas of that unit so that the output at the next layer remains the same.



Net2Net



The Net2DeeperNet Transformation

Net2Net

- Experiments on ImageNet
- Net2Deeper and Net2Wider models converge faster to the same accuracy as networks initialized randomly.
- A deeper and wider model initialized with Net2Net from the Inception model beats the validation accuracy (and converges faster).

Net2Net

- Strengths
- Weaknesses / Notes

Thank you

Contact Details

- Deeplearning.ir
- DeepLearning Group @Telegram : https://t.me/joinchat/BhIXCD3_zWM3ubjs7LopZA
- Email : Coderx7@gmail.com

HAMIM©

References

- Papers:
  - LeNet(gradient based learning applied to document recognition)
  - AlexNet(mageNet Classification with Deep Convolutionaal Neurak Networks)
  - NetworkInNetwork
  - GoogleNet (**Going Deeper with Convolutions)**
  - Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
  - Rethinking the Inception Architecture for Computer Vision
  - VGGNet(**Very Deep Convolutional Networks for Large-Scale Image Recognition)**
  - ResNet(Deep Residual Learning for Image Recognition)
  - Wide Residual Network
  - SqueezeNet(**SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size**)
  - Striving for Simplicity: The All Convolutional Net
  - DenseNet(**Densely Connected Convolutional Networks)**
  - MobileNets(**MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications)**
  - Lets keep it simple, Using simple architectures to outperform deeper and more complex architectures
  - Knowledge distillation(**Distilling the Knowledge in a Neural Network)**
  - **Do Deep Nets Really Need to be Deep?**
  - FitNet
  - Net2Net(**Net2Net: Accelerating Learning via Knowledge Transfer**)
  - Early introduction to Simpnet2: Towards principled design of Deep convolutional neural networks
- Nearly all Figures and charts used were borrowed from their respective papers, however there are some which were borrowed from
  - Stanford CS231n course lectures
  - Andrew Ng neural networks lectures
  - Why is gemm at the heart of deep learning blog post (https://petewarden.com/)