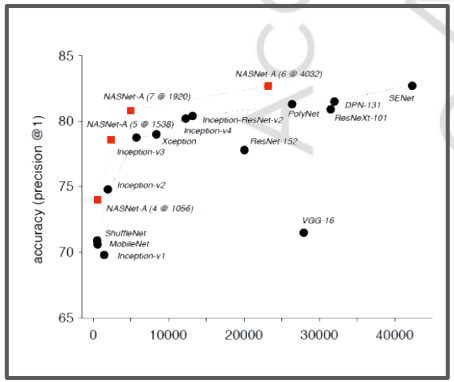
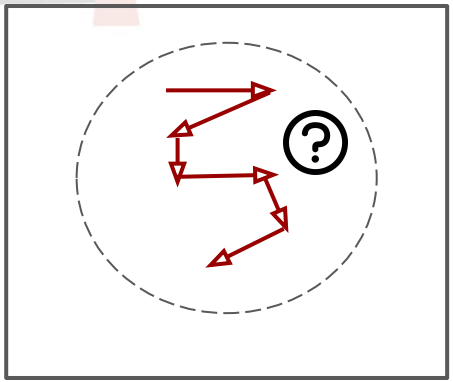
# Advance Deep Learning for Computer Vision

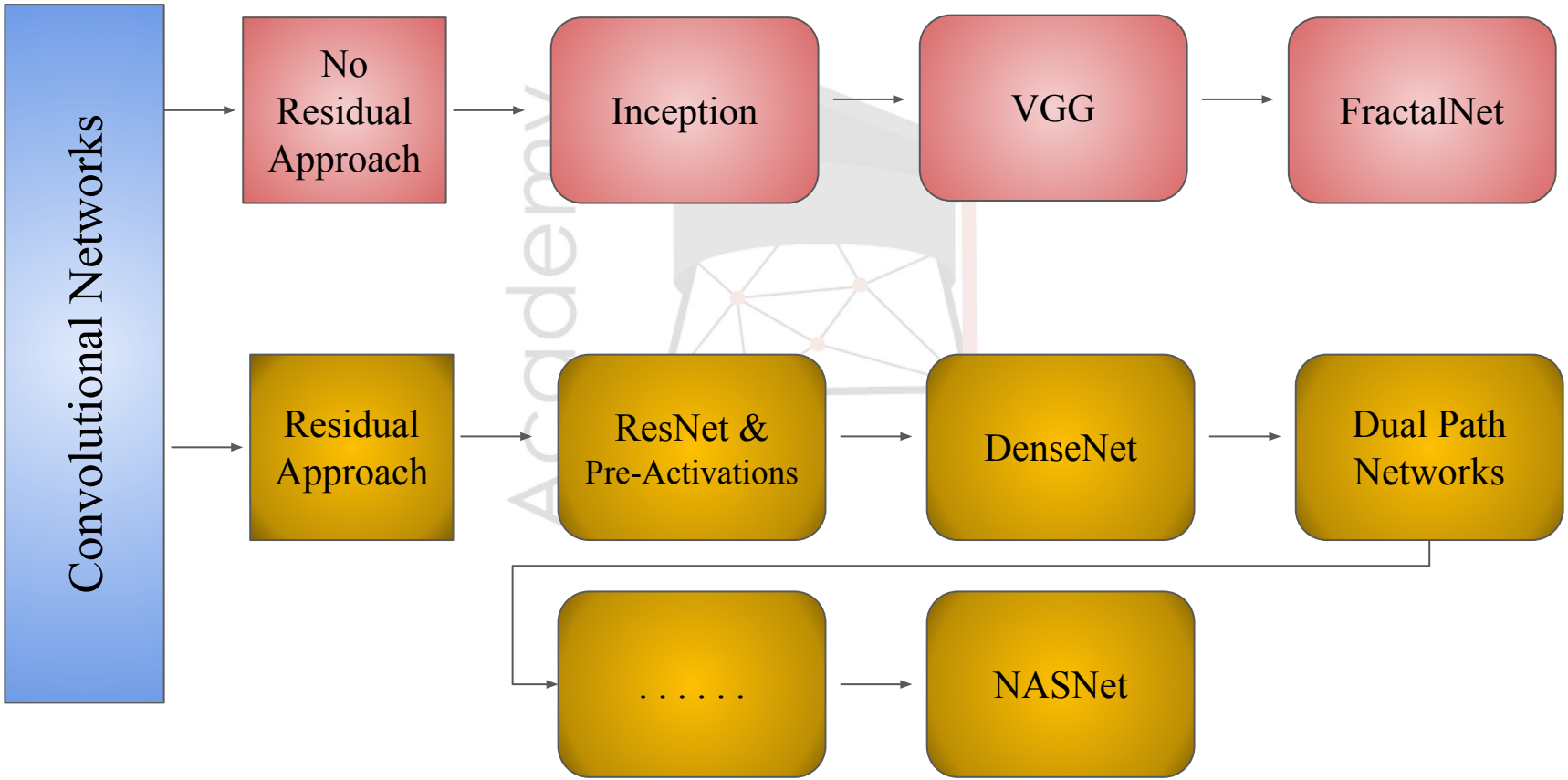
*Architectures and Transfer Learning Methodologies*

*Module 1*

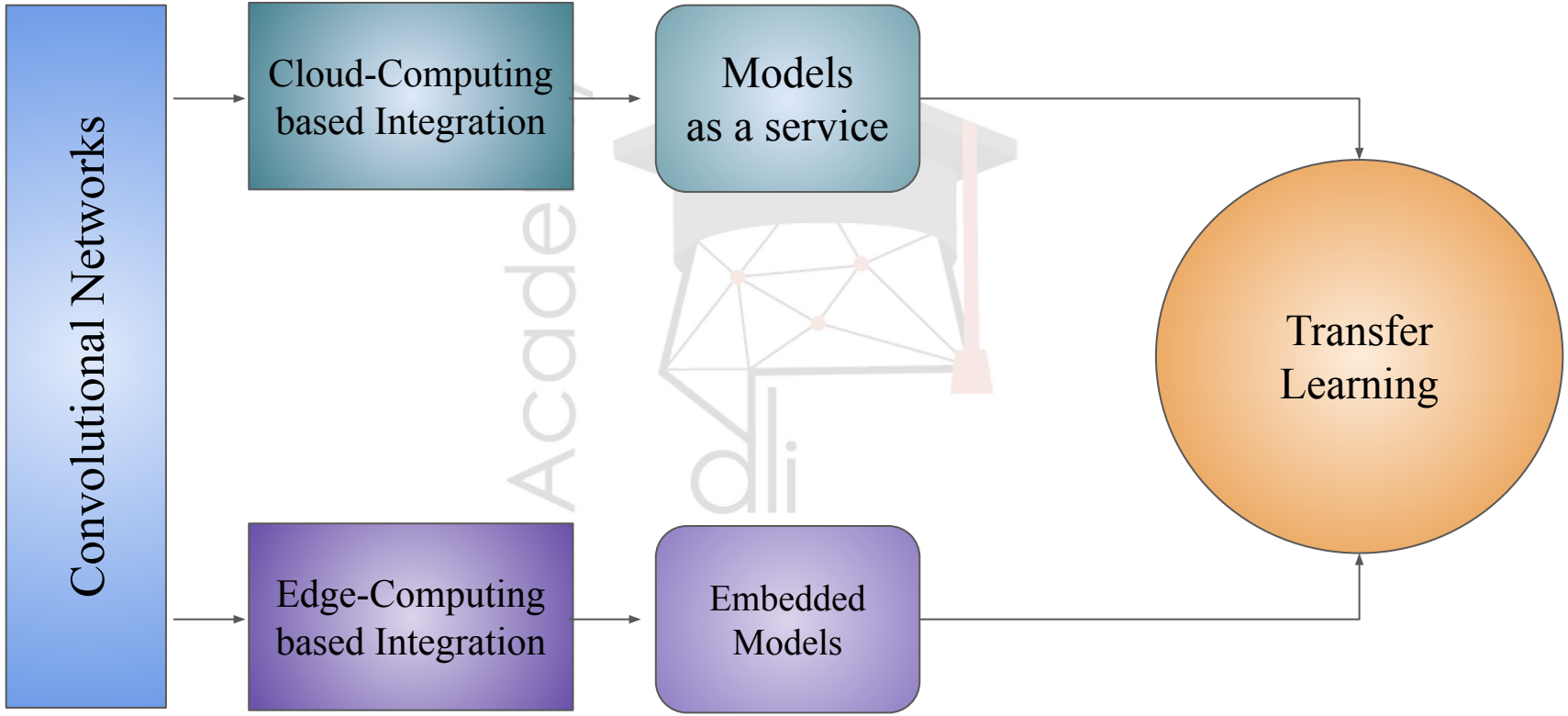


# Where architectural research is going?



# Where research's applications is going?



# Recap on Convolution and Pooling Operators

We have seen convolutions and pooling operators as mathematical functions. Now we need to think them under an architectural perspective, like “layers”

Why is so important? Debugging!

# Convolutional Layers

- ➔ Give an Input A0 x L0 x D0
- ➔ Hyperparameter settings
  - ◆ # kernels (K)
  - ◆ receptive field (F)
  - ◆ stride (S)
  - ◆ padding (P)
- ➔ Produce A1 x L1 x D1
  - ◆ Where : A1 and L1 volume is given by : $Output\,(V_o) = \frac{A - F + 2P}{S} + 1$ equal to # kernels
- ➔ Produce multiple features maps

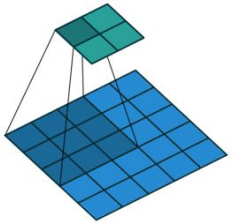
# Convolutional Layers

- What are input dimensions?

| | | |
|---|---|---|
| conv2d_1input: InputLayer | input | (None, 32, 32,3) |
| | output | (None, 32, 32,3) |

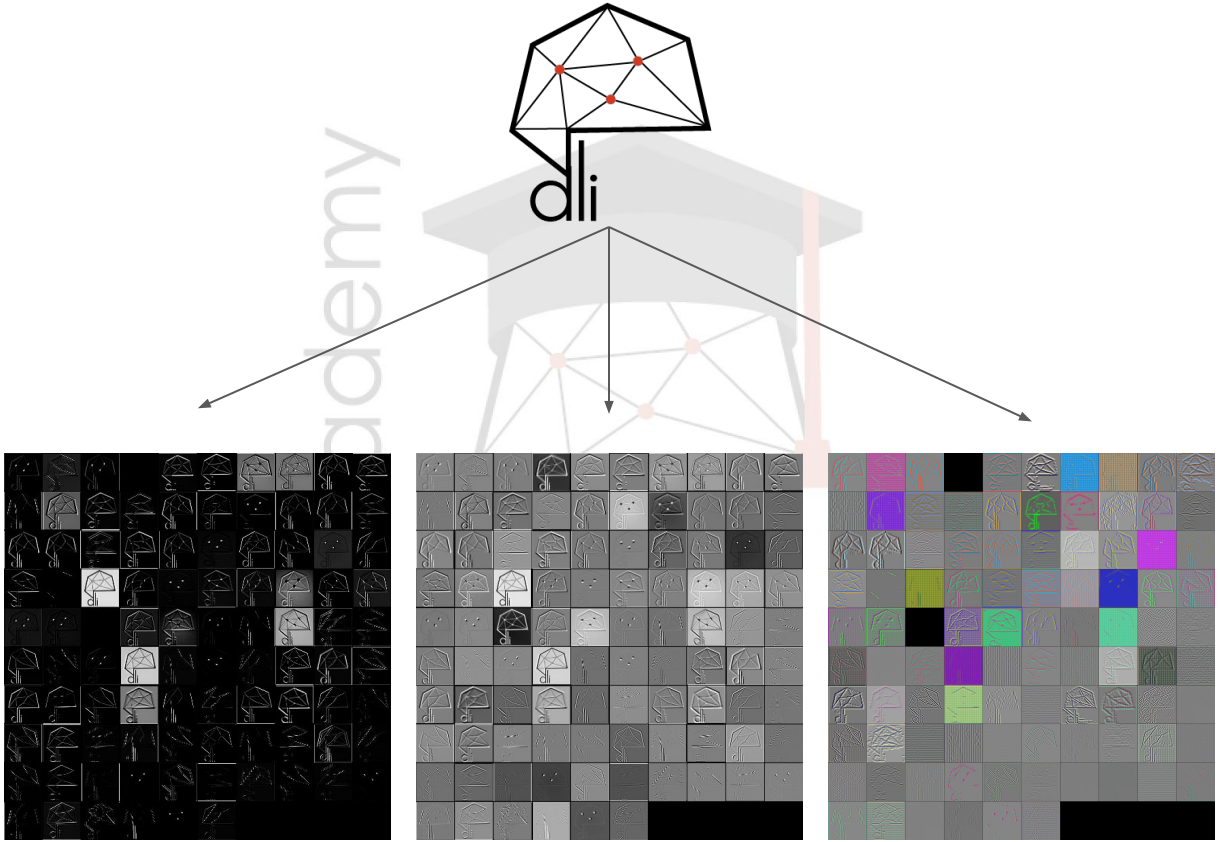
- First Convolution Layer :
  - 6 kernels
  - (5,5) receptive fields
  - no padding

| | | |
|---|---|---|
| conv2d_1: Conv2D | input | (None, 32, 32,3) |
| | output | (None, 28, 28,6) |



[Convolution Gif from vdumoulin, github]

# Convolutional Layers



# Pooling Layers

- Input Ai x Li x Di
- Hyper-parameters :
  - Kernel Dimension K
  - Sliding S
- Produce the following output

$$A = \left(\frac{Ax - F}{S}\right) + 1$$

$$L_{x+1} = \left(\frac{Lx - F}{S}\right) + 1$$

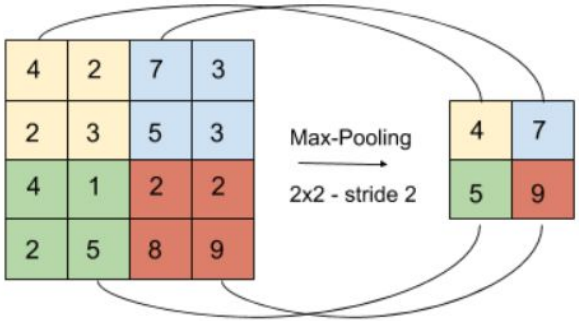
$$P_x = P_{x+1}$$

# Pooling Layers

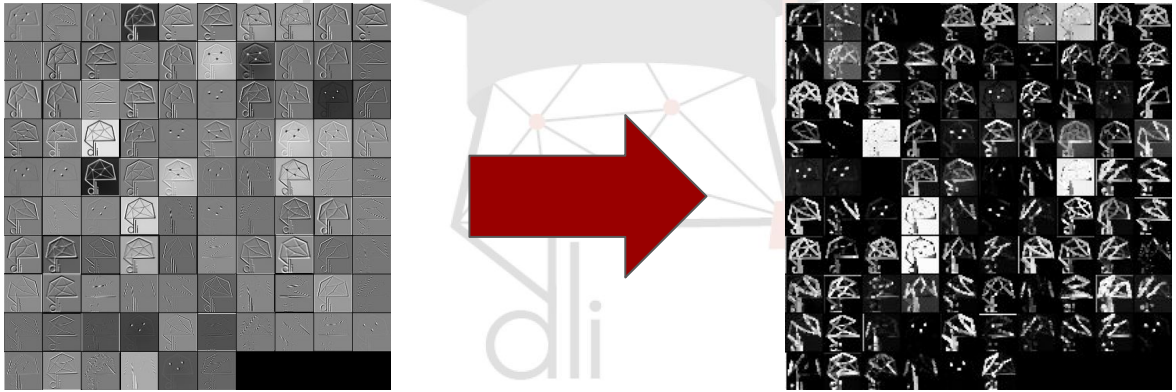
➔ What are input dimensions?

➔ First Pooling Layer :
- ◆ (2,2) receptive fields
- ◆ (2,2) stride

| | | |
|---|---|---|
| conv2d_1: Conv2D | input | (None, 32, 32,3) |
| | output | (None, 28, 28,6) |
| max_pooling2d_1: MaxPooling2D | input | (None, 28, 28,6) |
| | output | (None, 14, 14,6) |



# Pooling Layers



# Are we forgetting something?

➔ *How many parameters have we to train?*

*Please use only paper and pen in this section*

**Convolution Layer**

- (K*F*F*Channels) * Bias

**Pooling Layer**

- No one

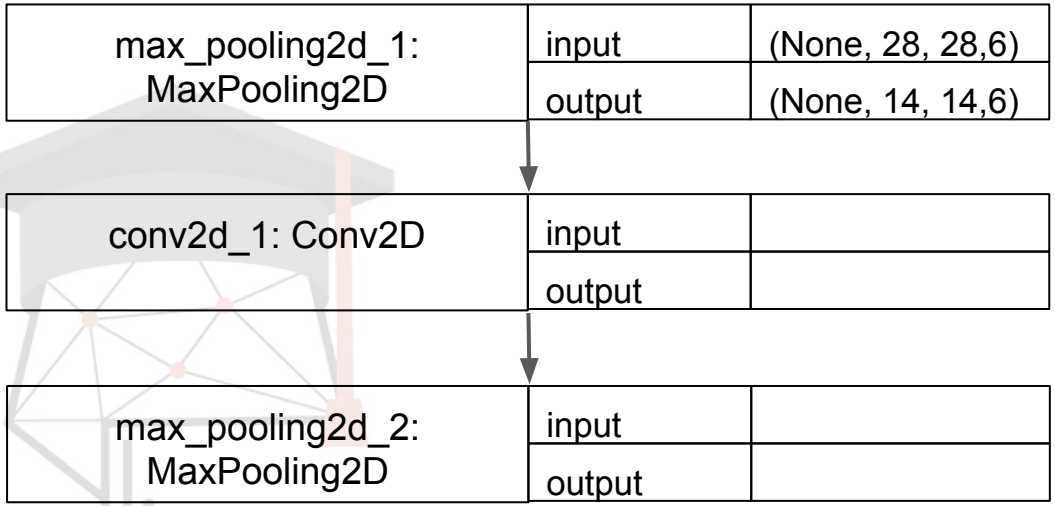
**Fully-Connected Layer (Dense)**

- ( $\pi$ Activation size previous layer * $\pi$ Activation size actual layer) +1

[How calculate network parameteres : http://cs231n.github.io/convolutional-networks/](http://cs231n.github.io/convolutional-networks/)

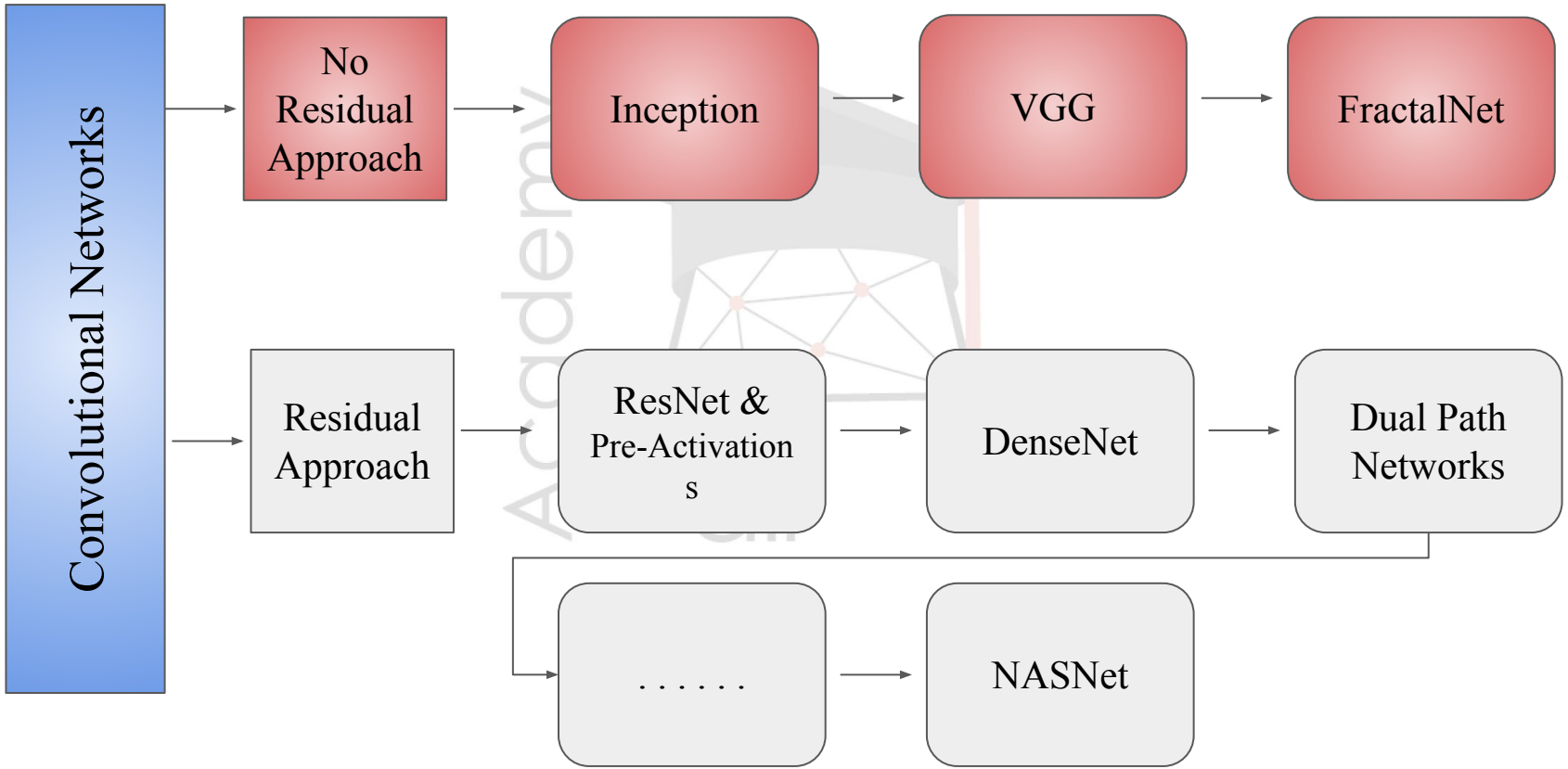
# Try yourself

- ➔ Second Conv Layer :
  - ◆ K : 16
  - ◆ F : (5,5)
  - ◆ no padding
- ➔ Second Pooling Layer:
  - ◆ (2,2) receptive fields
  - ◆ (2,2) stride

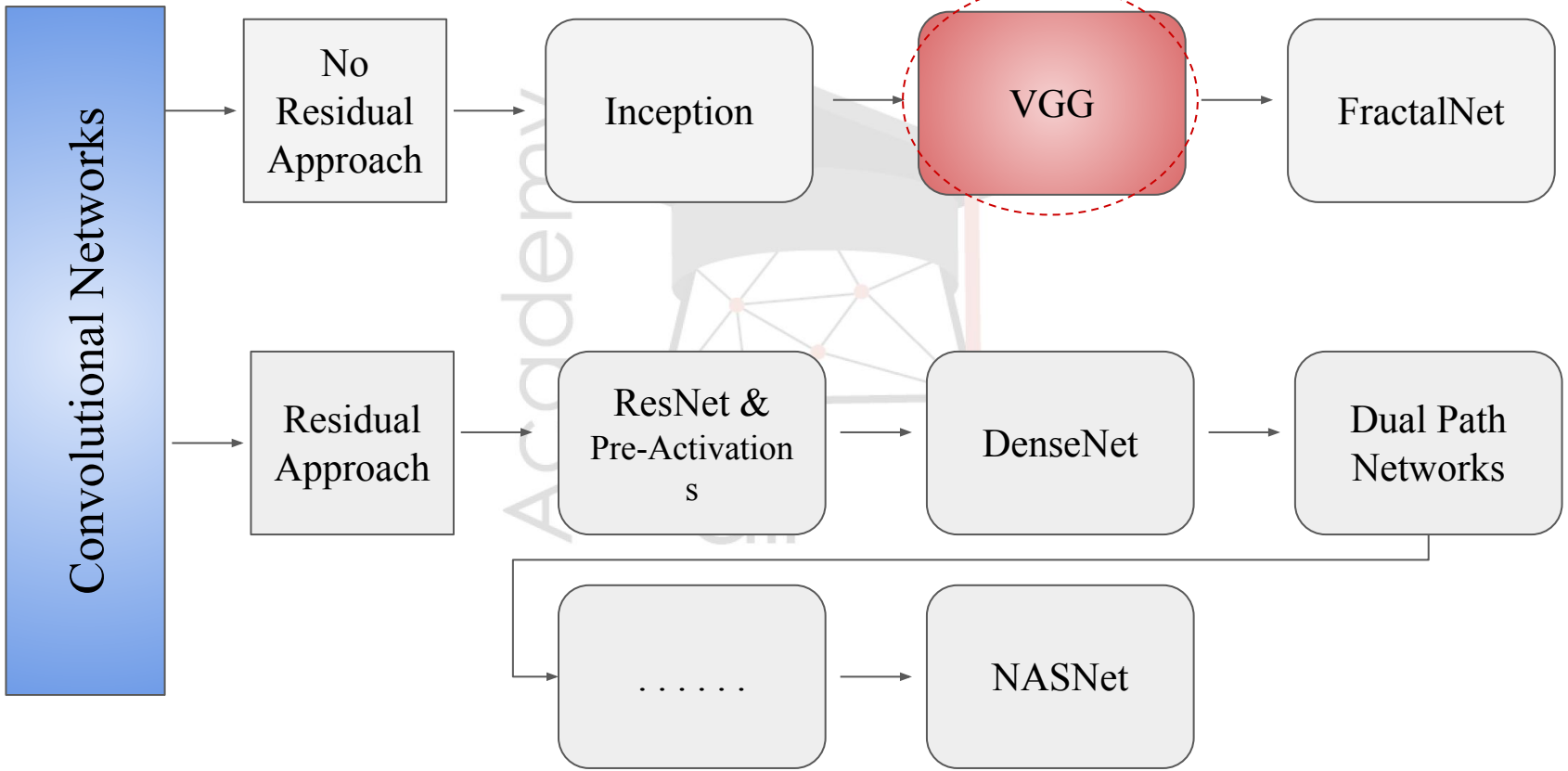



Try yourself with paper and pen with different padding, strides, receipt fields . . .

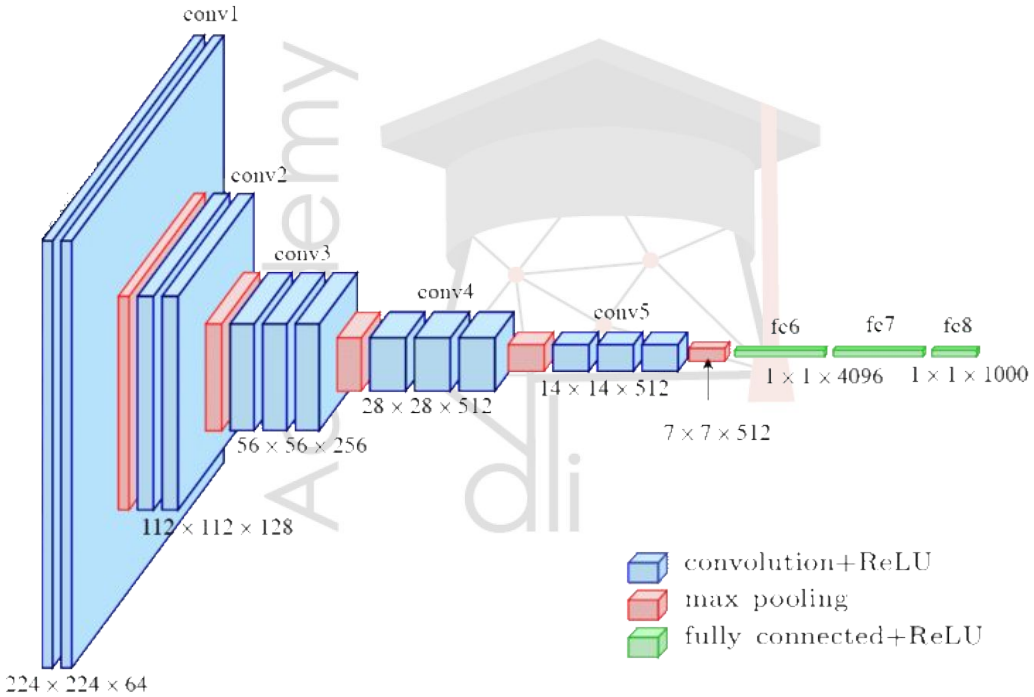
# Where research in *universities* and *academic world* is going?



# Where research in *universities* and *academic world* is going?



# Introduction to VGG Neural Networks

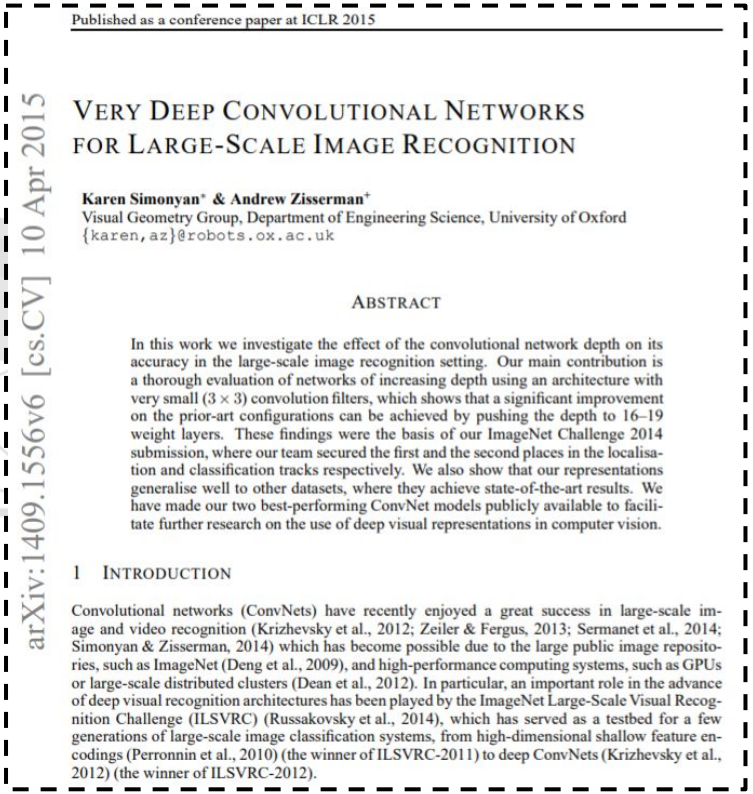


VGG Net

# Visual Geometry Group Network

- Good baseline for computer vision models
- Excellent results on Imagenet competition (ILSVRC-2014)
- *Easy to develop*
- Official Keras implementation

*... try it yourself!*



Published as a conference paper at ICLR 2015



## VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

**Karen Simonyan*** **& Andrew Zisserman***
Visual Geometry Group, Department of Engineering Science, University of Oxford
{karen,az}@robots.ox.ac.uk

ABSTRACT

In this work we investigate the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. Our main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small (3 × 3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19 weight layers. These findings were the basis of our ImageNet Challenge 2014 submission, where our team secured the first and the second places in the localisation and classification tracks respectively. We also show that our representations generalise well to other datasets, where they achieve state-of-the-art results. We have made our two best-performing ConvNet models publicly available to facilitate further research on the use of deep visual representations in computer vision.

1 INTRODUCTION

Convolutional networks (ConvNets) have recently enjoyed a great success in large-scale image and video recognition (Krizhevsky et al., 2012; Zeiler & Fergus, 2013; Sermanet et al., 2014; Simonyan & Zisserman, 2014) which has become possible due to the large public image repositories, such as ImageNet (Deng et al., 2009), and high-performance computing systems, such as GPUs or large-scale distributed clusters (Dean et al., 2012). In particular, an important role in the advance of deep visual recognition architectures has been played by the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) (Russakovsky et al., 2014), which has served as a testbed for a few generations of large-scale image classification systems, from high-dimensional shallow feature encodings (Perronnin et al., 2010) (the winner of ILSVRC-2011) to deep ConvNets (Krizhevsky et al., 2012) (the winner of ILSVRC-2012).

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION , Karen Simonyan, Andrew Zisserman, University of Oxford

# How should I proceed?

***Step 0 :***

- ➔ Read all the paper
- ➔ Choose which configuration achieve the best results

***Step 1 :***

- ➔ Focus on network architecture
  - ◆ receptive field
  - ◆ stride
  - ◆ padding
  - ◆ activation functions
  - ◆ pooling
  - ◆ fc-connected
  - ◆ input data size

*Please use only paper and pen in this section*

# How should I proceed?

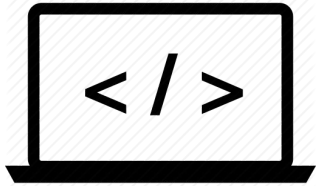
***Step 2 :***

- ➔ Calculate shape for each layer
- ➔ Calculate parameters for each layer

*Please use only paper and pen in this section*

***Step 3 :***

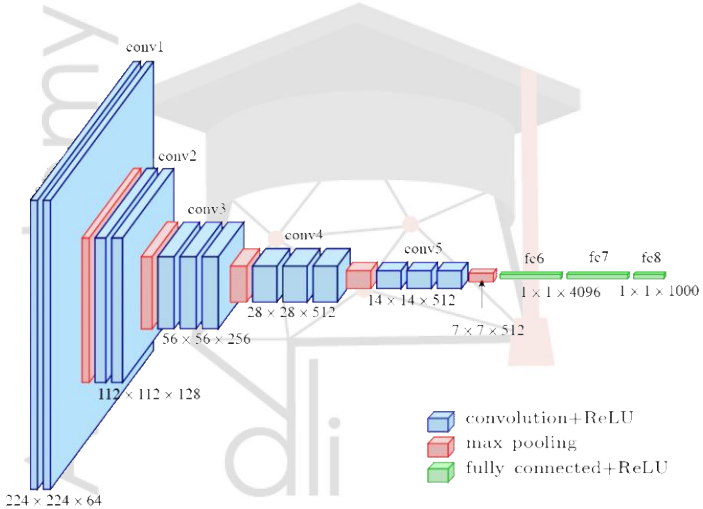
- ➔ Develop VGG net in Keras on Cifar10 *(not train the network!)*
- ➔ Check shapes and parameters with *model.summary()*
- ➔ Check differences in terms of total parameters between different VGG Net configurations



VGG Net

# Solutions

*Visual Geometry Group Network*



***Step 0 :***

- ➔ Read all the paper ✔
- ➔ Choose which configuration achieve the best results

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

| 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<br>**LRN** | conv3-64<br>**conv3-64** | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128<br>**conv3-128** | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 |
| maxpool | | | | | |
| conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256<br>**conv1-256** | conv3-256<br>conv3-256<br>**conv3-256** | conv3-256<br>conv3-256<br>conv3-256<br>**conv3-256** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |

***Step 0 :***

- ➔ Read all the paper ✔
- ➔ Choose which configuration achieve the best results ✔

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

| 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<br>**LRN** | conv3-64<br>**conv3-64** | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128<br>**conv3-128** | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 |
| maxpool | | | | | |
| conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256<br>**conv1-256** | conv3-256<br>conv3-256<br>**conv3-256** | conv3-256<br>conv3-256<br>conv3-256<br>**conv3-256** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |

***Step 1 :***

➔ Focus on network architecture

- *receptive field*
- *stride*
- *padding*
- *activation functions*
- *pooling*
- *fc-connected*
- *input data size*

*Please use only paper and pen in this section*

***Step 1 :***

➔ Focus on network architecture

- *receptive field* **(3x3)** ✓
- *stride* **(1)** ✓
- *padding* **(1)** ✓
- *activation functions* **(Relu)** ✓
- *pooling* **(2x2), stride 2** ✓
- *fc-connected* **(4096)** ✓
- *input data size* **(224x224x3)** ✓

*we will use different data input :*
***(32 x 32 x 3)***

*Please use only paper and pen in this section*

***Step 2 :***

- ➔ Calculate shape for each layer
- ➔ Calculate parameters for each layer

*Please use only paper and pen in this section*

***Step 2 :***

➔ Calculate shape for each layer ✔

➔ Calculate parameters for each layer

*Please use only paper and pen in this section*

***Step 2 :***

- ➔ Calculate shape for each layer ✔
- ➔ Calculate parameters for each layer ✔

*Please use only paper and pen in this section*

***Step 3 :***

- ➔ Develop VGG net in Keras on Cifar10 *(not train the network!)*
- ➔ Check shapes and parameters with *model.summary()*
- ➔ Check differences in terms of total parameters between different VGG Net configurations *(from paper)*

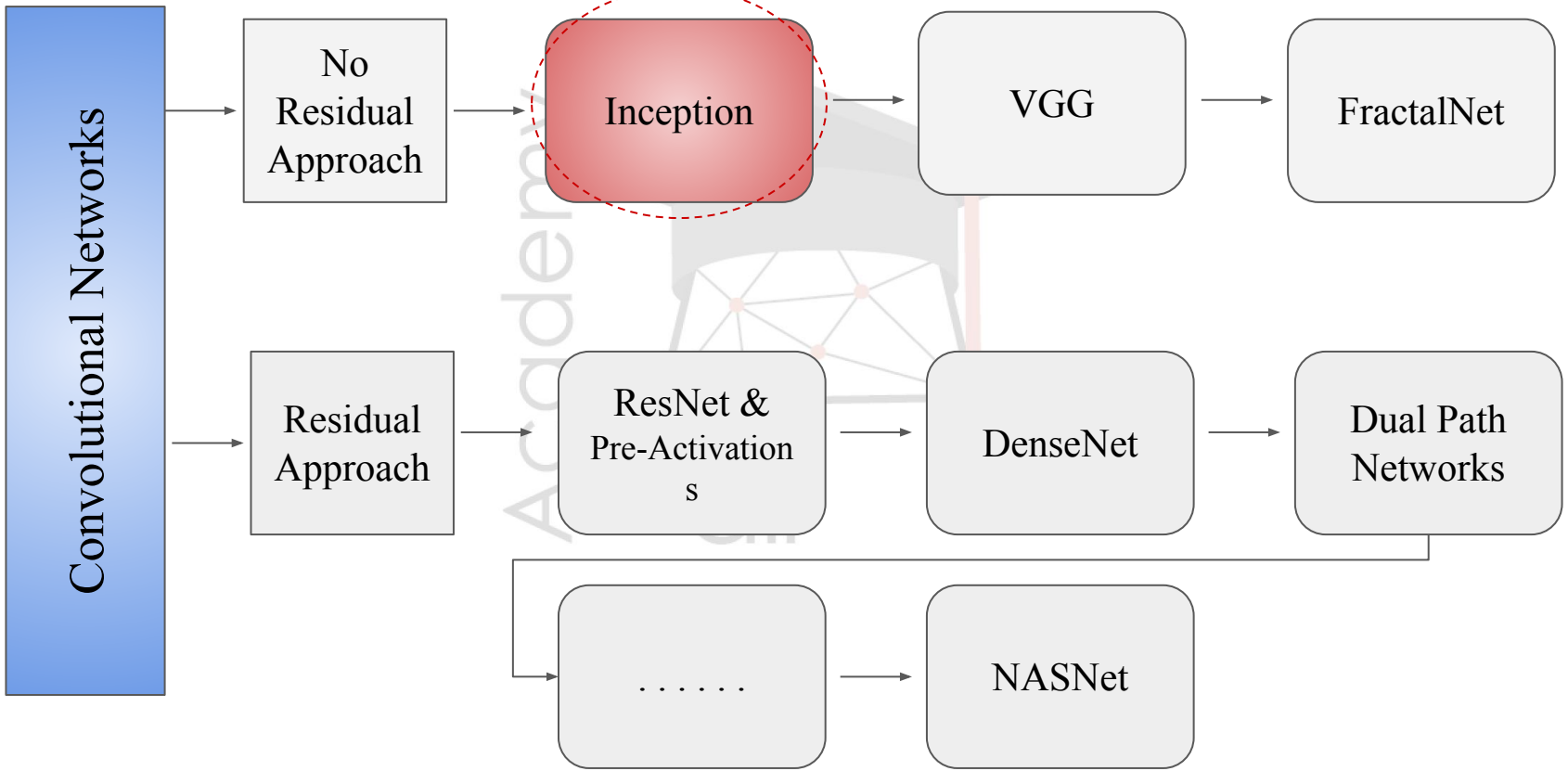
***Step 3 :***

- ➔ Develop VGG net in Keras on Cifar10 *(not train the network!)* ✔
- ➔ Check shapes and parameters with *model.summary()* ✔
- ➔ Check differences in terms of total parameters between different VGG Net configurations *(from paper)*

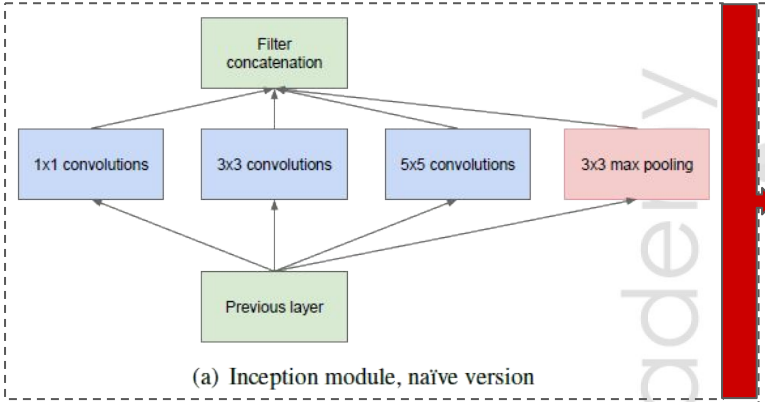
***Step 3 :***

- ➔ Develop VGG net in Keras on Cifar10 *(not train the network!)* ✔
- ➔ Check shapes and parameters with *model.summary()* ✔
- ➔ Check differences in terms of total parameters between different VGG Net configurations *(from paper)* ✔

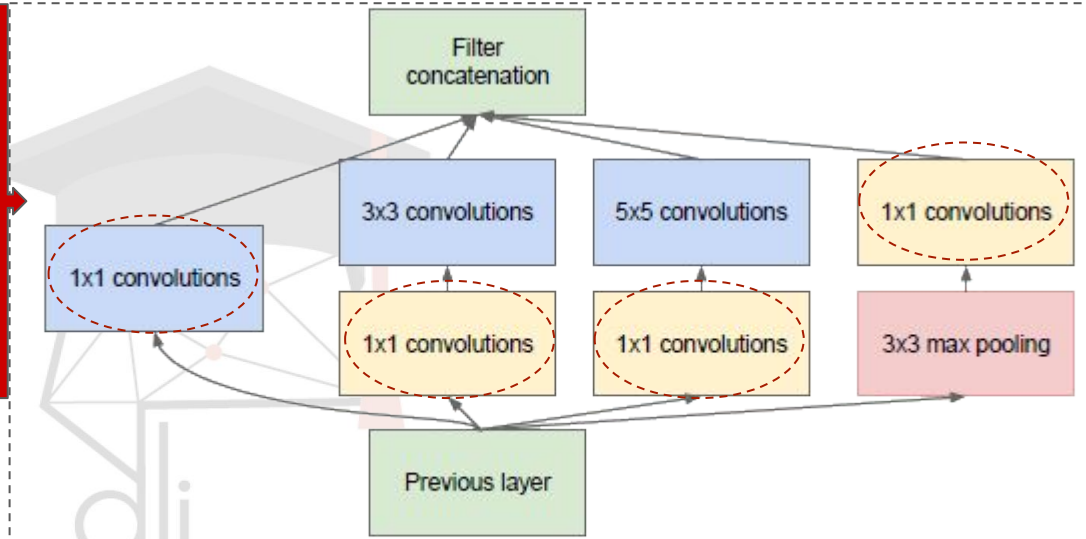
# Where research in *universities* and *academic world* is going?



# *Brief Overview :*
# Inception Module




(a) Inception module, naïve version



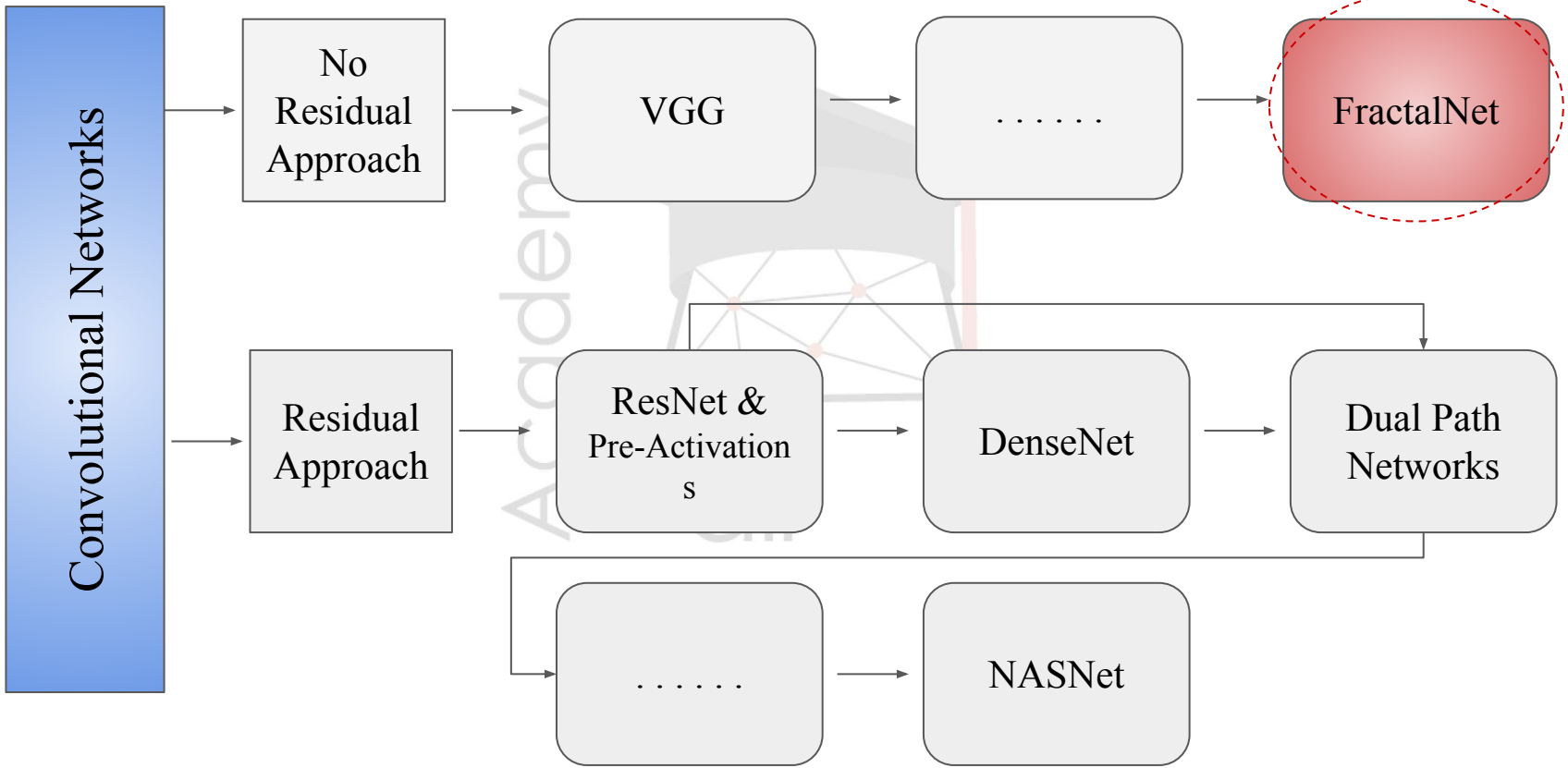

(b) Inception module with dimensionality reduction

*Googlenet Architecture*

- *A way to increase not only the depth of a neural networks but also the width*
- *Parameters grows dramatically!*
- *We focus on concatenation*

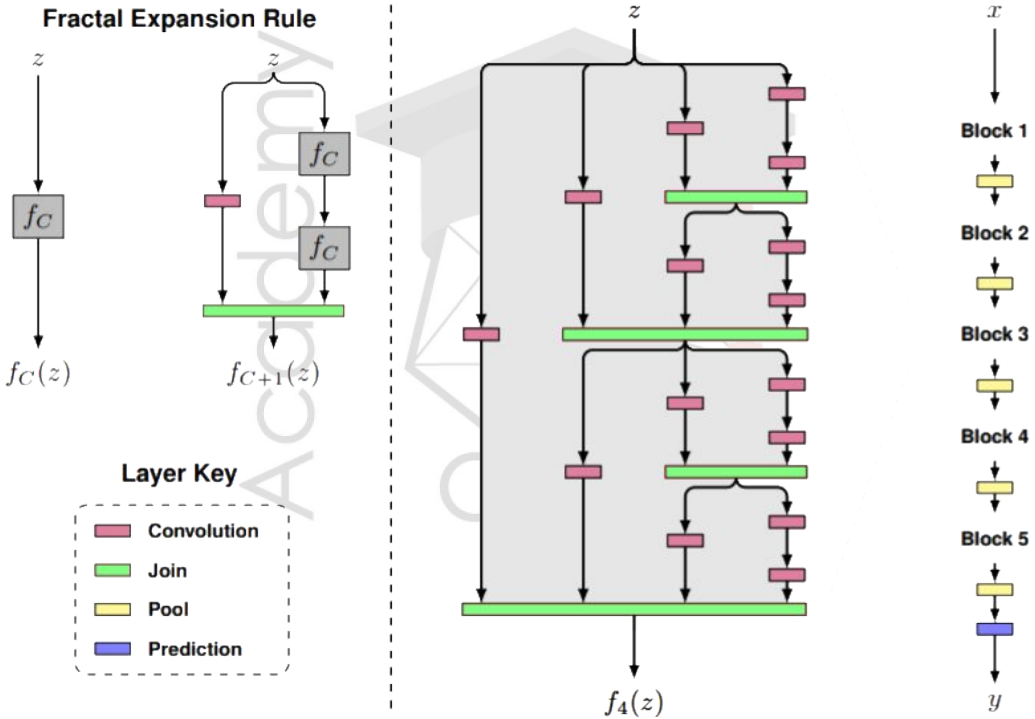
Going Deeper with Convolutions, Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich

# Where research in *universities* and *academic world* is going?



# FractalNet




Fractal Expansion Rule
$z$
$f_C$
$f_C(z)$
$z$
$f_C$
$f_C$
$f_{C+1}(z)$
Layer Key
Convolution
Join
Pool
Prediction
$z$
$f_4(z)$
$x$
Block 1
Block 2
Block 3
Block 4
Block 5
$y$

# Why Fractal Network?

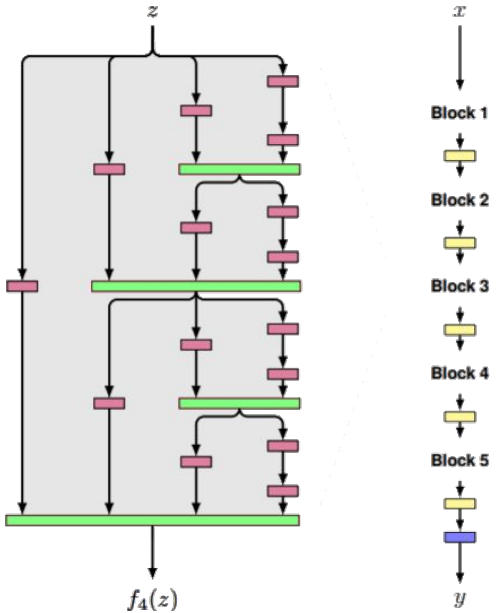
Fractal Net

*VGG Net* achieve its best depth around 16-19 layers

➔ After that the degrading due to *vanishing gradient problem* starts!

*Fractal Net*

➔ Can achieve the depth of **40 up to *80*** *layers without Residual approach*




FractalNet: Ultra-Deep Neural Networks without Residuals, Gustav Larsson, Michael Maire, Gregory Shakhnarovich

# Fractal Network - Basic idea behind

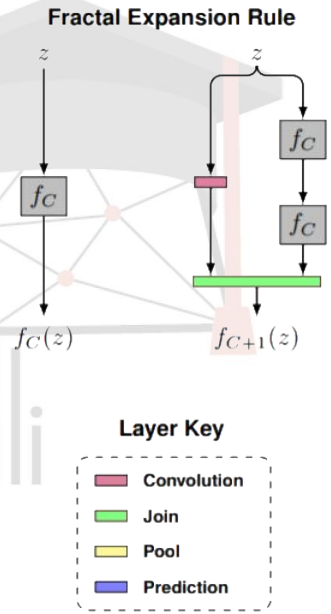
Fractal Net

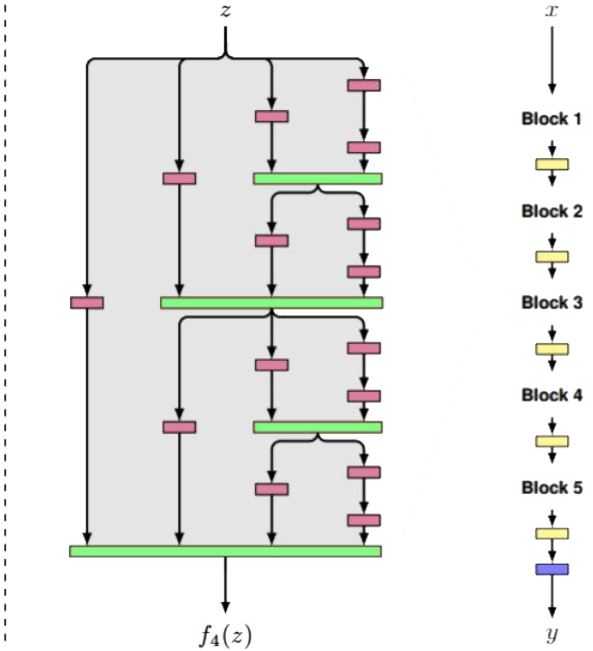
*Starting Point : <u>two branches</u>*

- ➔ Left contains one convolutional layer
- ➔ Right contains a *subnetwork*
  - *Left contains one convent*
  - *Right contains subnetworks*

In this way we are developing a *recursion,* where at last step of *subnetwork we insert two convolutional layers*

- ➔ Each *pair of branches* is <u>merged</u> ***pair-wise mean*** this produce that one of our two branches (for each pair) will be *skipped*



# Fractal Network - Architecture & Math

Fractal Net

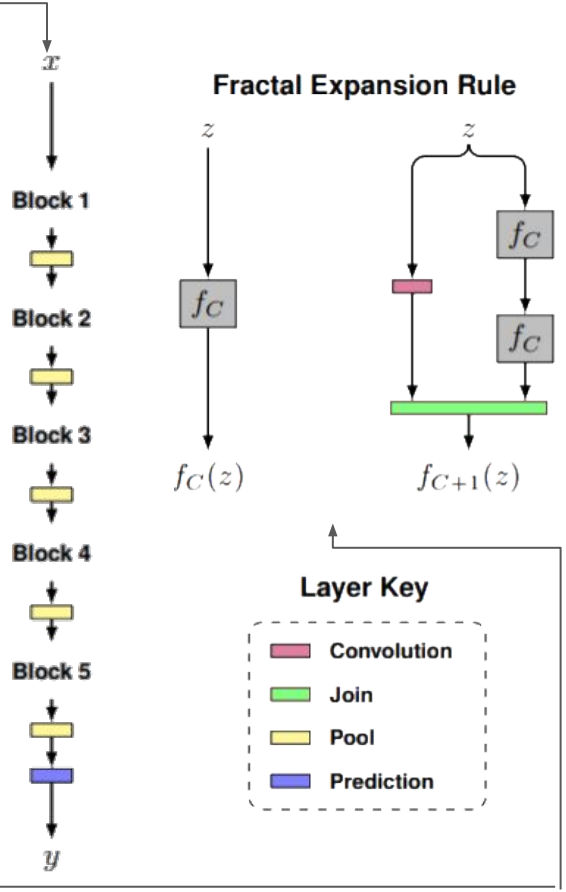
*Blocks :*

- ➔ We define Block each networks recursively generated
- ➔ Total networks contains 5 *blocks*
  - ◆ Kernel dimensions are : *{64-128-256-512-512}*
  - ◆ After each *blocks* we have a pooling layer (max-pooling)

*Analyse the first block :*

- ➔ *Left : convolutional layer → $F_1(z) = conv(z)$*
- ➔ *Right : recursive fractal → $F_{c+1}(z) = [(F_c \circ F_c)(z)] \oplus [conv(z)]$*

*Number of columns → C*
*Number of layers → 2^(C-1) → 8 layers*



# How many total convolutional layers are there?

## How many Blocks are there?

*5 Blocks in paper architecture*

*layers in one block x # of Blocks :*

*2^(C-1) x 5 =* ***40!***

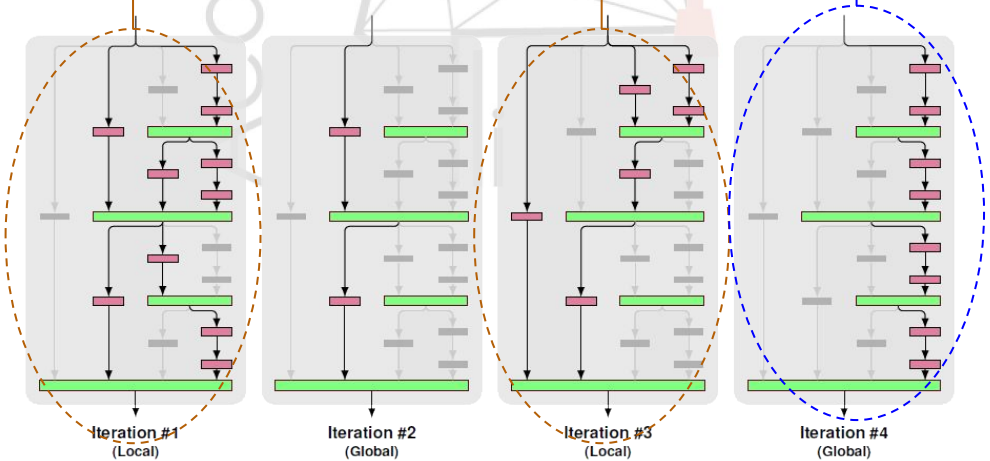
# Fractal Network - Architecture Details & Drop-path



*Drop-Path between merged-layers (**dropout**)*

➔ *Local Drop-path :*
*Drop with a fixed-probability between each input and merge layer.* *At least one survive*

➔ *Global Drop-path :*
*Drop until just **one** column survive*




Like "*ordinary **dropout***" ***drop-path*** avoid overfitting problems

# Fractal Network - Questions & Possible Answers

Fractal Net

***Open the paper and try to answer the following questions :***

*(these questions help you to find the right approach to a new paper reading - so a scientific approach as possible)*

*Which kind of data they used? What kind of data can you use to improve the robustness of accuracy achieved?*

*Which kind of models did they compare to? Could they test with others*
*(check publication date)*

*Can we see a good improvements of drop-path associated with data augmentation?*

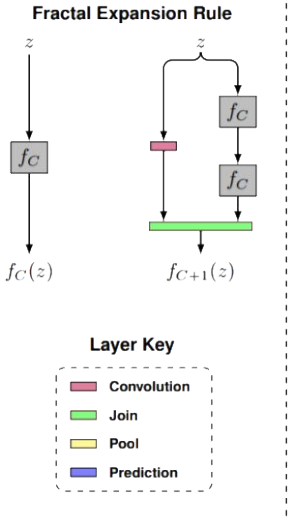
***When they extract the deepest column for tests performance seems to be nearly the same of entire networks.. what kind of considerations could we do?***

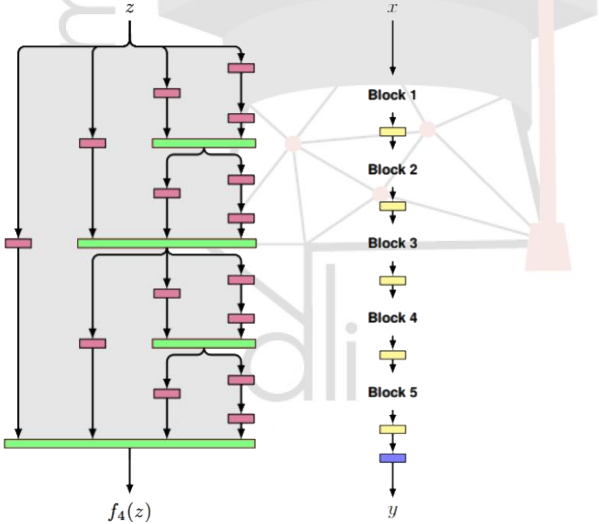
FractalNet: Ultra-Deep Neural Networks without Residuals, Gustav Larsson, Michael Maire, Gregory Shakhnarovich

Fractal Net

# Solutions

*Fractal Network Questions*



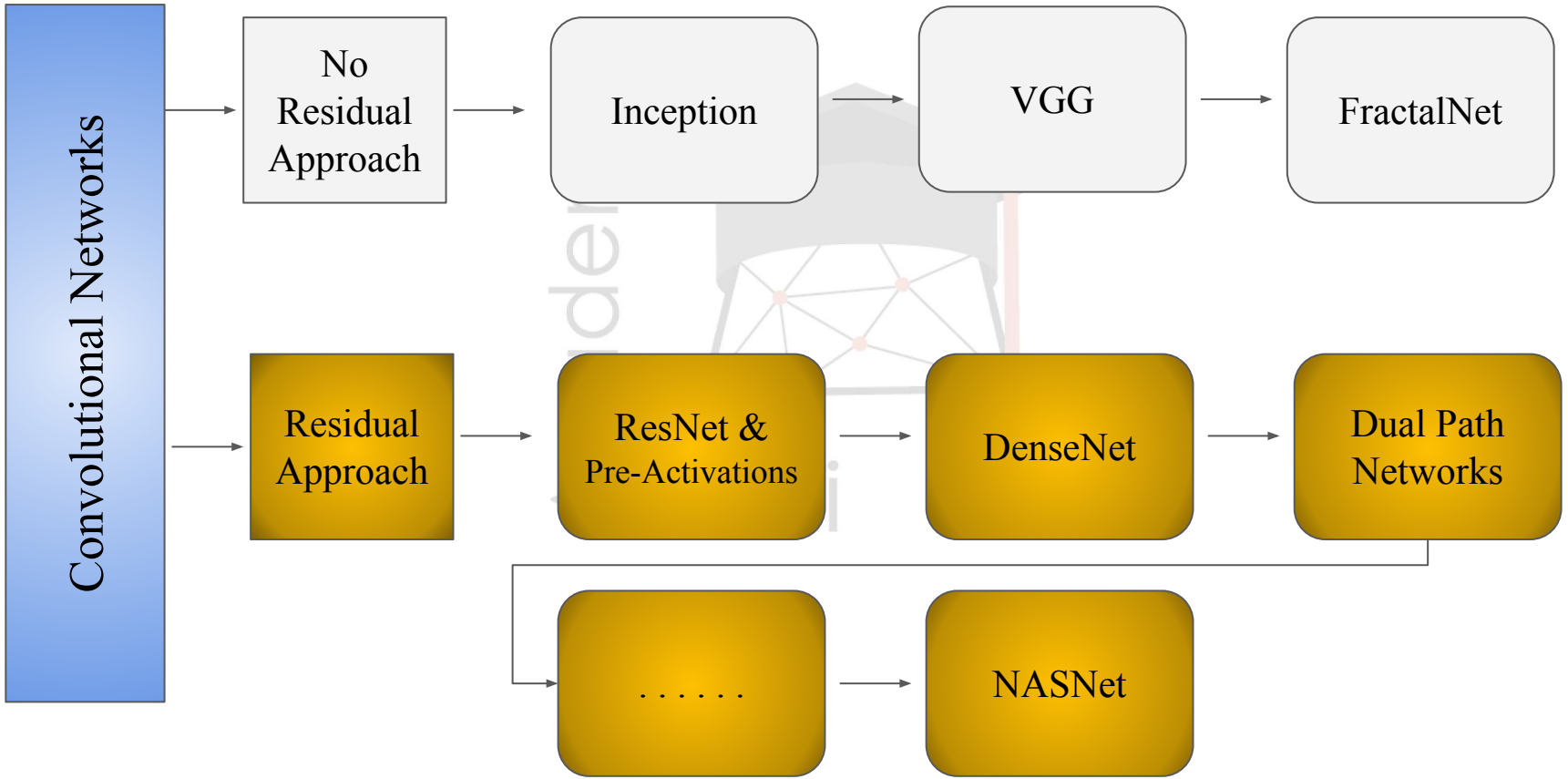

# Fractal Network - Questions & Possible Answers

Fractal Net

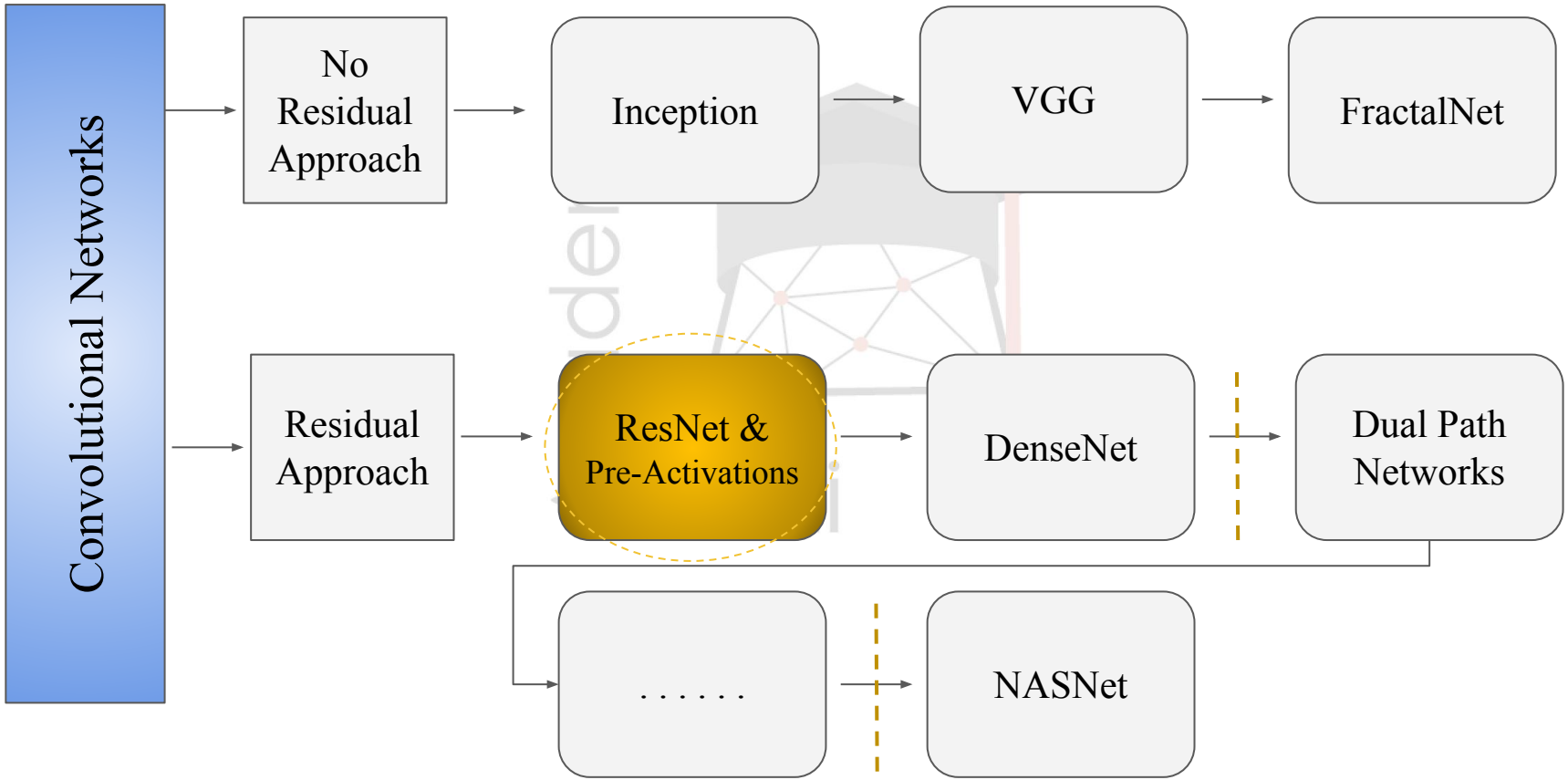
| | |
|---|---|
| *Which kind of data they used? What kind of data can you use to improve the robustness of accuracy achieved?* | *Cifar10 \| Cifar100 \| SVHN \| Imagenet*<br>➔ *10 - 100 - 1000 classes tests*<br>➔ *data shapes : **32x32x3 - 256x256x3*** |
| *Which kind of models did they compare to? Could they test with others*<br>*(check publication date)* | *They compare at least with ResNet with Identity Mapping (not with Wide ResNet)* |
| *Can we see a good improvements of drop-path associated with data augmentation?* | *Only very small improvement* |
| ***When they extract the deepest column for tests performance seems to be nearly the same of entire networks.. what kind of considerations could we do?*** | *Fractal architecture seems to be useful (for the most) to let the information acquired in first - medium layers be available for deepest one*<br>*(avoiding vanishing gradient problem)* |

FractalNet: Ultra-Deep Neural Networks without Residuals, Gustav Larsson, Michael Maire, Gregory Shakhnarovich

# Where research in *universities* and *academic world* is going?

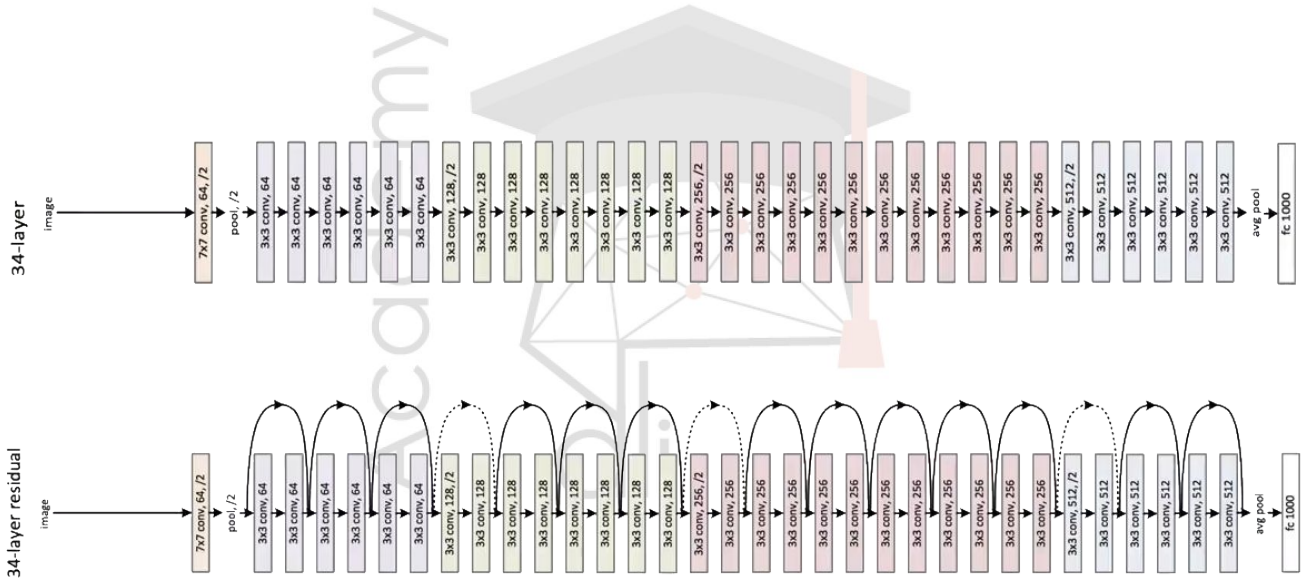


# Where research in *universities* and *academic world* is going?



ResNet

# Introduction to Residual Neural Networks



# Through Residual connections

ResNet

In a “standard” convolutional neural network like VGG, suppose the output from the th layer is written as .

Mathematically, we can write: $y_n = f_n(y_{n-1})$

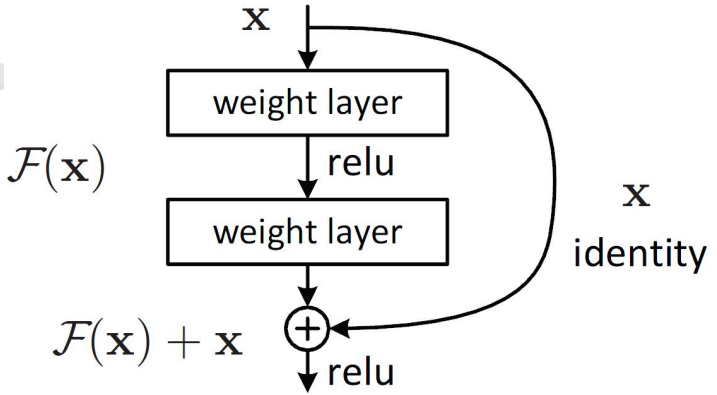
- ➔ $f_n$ vectorial function from $y_n$ and $y_{n-1}$
- ➔ the mathematical computation performed by f is same across all layers
- ➔ but between $f_n$ and other layers we have set of weights assigned to every element

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION , Karen Simonyan, Andrew Zisserman, University of Oxford

# Through Residual connections

ResNet

The new expression becomes : $y_n = f_n(y_{n-1}) + y_{n-1}$




➔ we can define $y_{n-1}$ as identity skip $f_n$

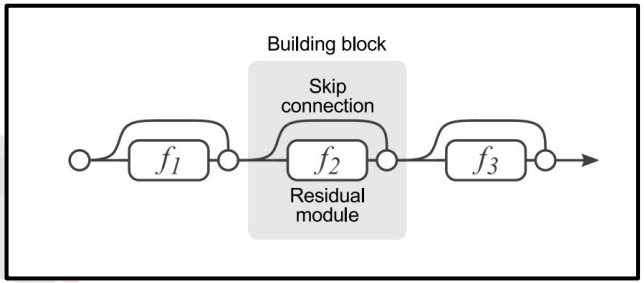
- Skip : we are skipping the computation of $y_{n-1}$
- Identity : there is no kind of multiplication between and any set of weights before addition

# Unravelling Residual Neural Networks

ResNet

Let's consider a simple residual NN with three layers :

➔ The *recursive definition* essentially defines $y_3$ as a sum of outputs from every previous layer in the network




$$y_3 = f_3(y_2) + y_2$$

$$y_3 = f_3(f_2(y_1) + y_1) + f_2(y_1) + y_1$$

$$y_3 = f_3(f_2(f_1(y_0) + y_0) + f_1(y_0) + y_0) + f_2(f_1(y_0) + y_0) + f_1(y_0) + y_0$$

➔ In a conventional CNN (like VGG) we have

$$y_3 = f_3(f_2(f_1(y_0)))$$



Residual Networks Behave Like Ensembles of Relatively Shallow Networks, Andreas Veit Michael Wilber Serge Belongie, Cornell University

# Unravelling Residual Neural Networks

ResNet

| *Standard Conv Nets* | *Residual Neural Networks* |
| --- | --- |
| $y_3 = f_3(f_2(f_1(y_0)))$ | $y_3 = f_3(f_2(f_1(y_0) + y_0) + f_1(y_0) + y_0) + f_2(f_1(y_0) + y_0) + f_1(y_0) + y_0$ |

Consider a situation where y_0 and the weights for f_2 are such that $f_2(y_1)$ turns out to be a vector of very low values (*Close to zero vector*)

➔ Any input basically follows one path through the NN

➔ $f_3$ can't do much

$$y_3 = f_3(f_1(y_0) + y_0) + f_1(y_0) + y_0$$

➔ the input signal ***bypassed*** the second layer completely

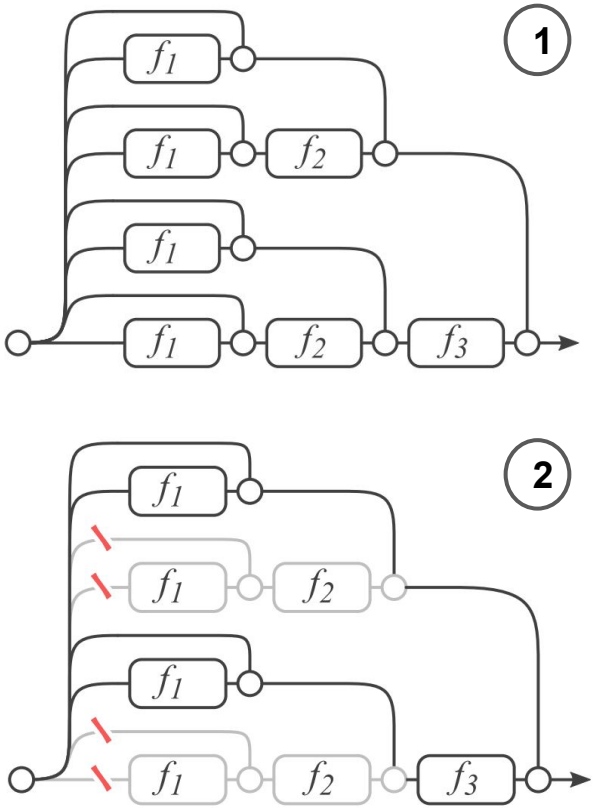
➔ We can derive the above expression for our network without $f_2$

# Skip-Connections of Residual Neural Networks

ResNet

The ability to ***bypass*** the input signal is *not restricted to one layer*.

*Skip-Connections* depend on input and can be for every combination of weights in each layer




➔ *In figure 1*, our network is fully unravelled for $y_0$ We can see that every *node* add a set of possible paths, where each output get contributions from every paths.

➔ *In figure 2*, represent $f_2$ diagram of all the cutting off of all paths where v makes no contributions

How many paths are there?

How many layers are there?

*Suppose N layers*

$2^N$

*total cases * each layer * (layer on, layer off)*

# Any kind of analogy with skip-connections??

Decision Tree, Random Forest, . . . but also DropOut!

***Ensemble***

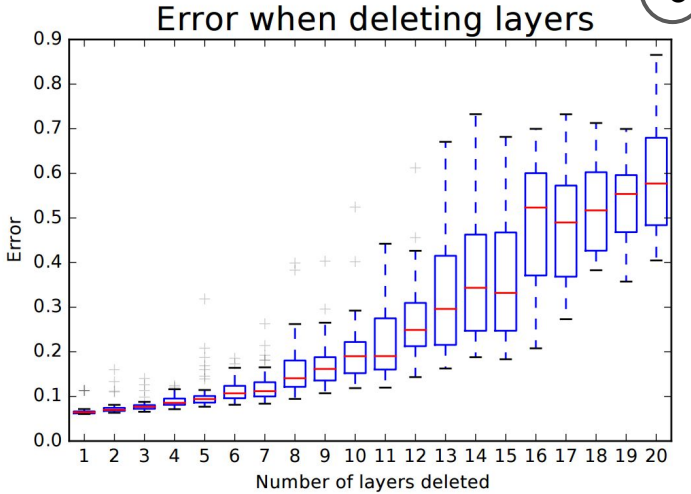
# Extra on Residual Learning

ResNet

**“robustness” to layer removal**

➔ Despite traditional ConvNets and ANNs with ResNets the removal of layers produce only a small loss of accuracy
*(as you can see in the figure 3)*

3




➔ *What is the reason?*
*(think mathematically)*

$$2^N \Rightarrow 2^{N-1}$$

$$y_3 = f_3(f_2(f_1(y_0) + y_0) + f_1(y_0) + y_0) + f_2(f_1(y_0) + y_0) + f_1(y_0) + y_0$$

$$\Downarrow$$

$$\boxed{y_3 = f_3(f_1(y_0) + y_0) + f_1(y_0) + y_0}$$

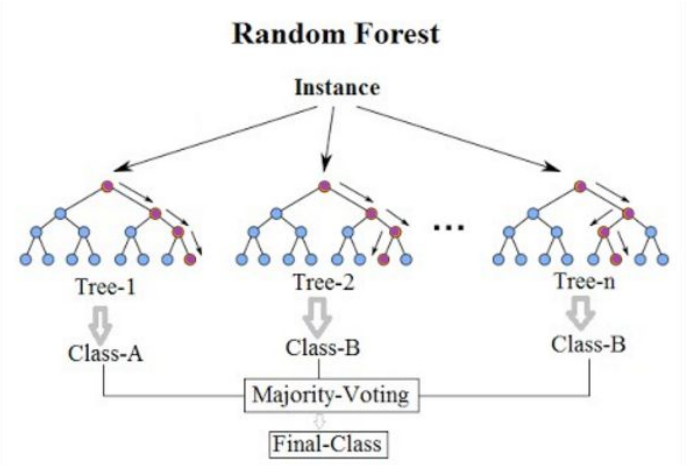
# Extra on Residual Learning

**Shallow vs Deep Networks**

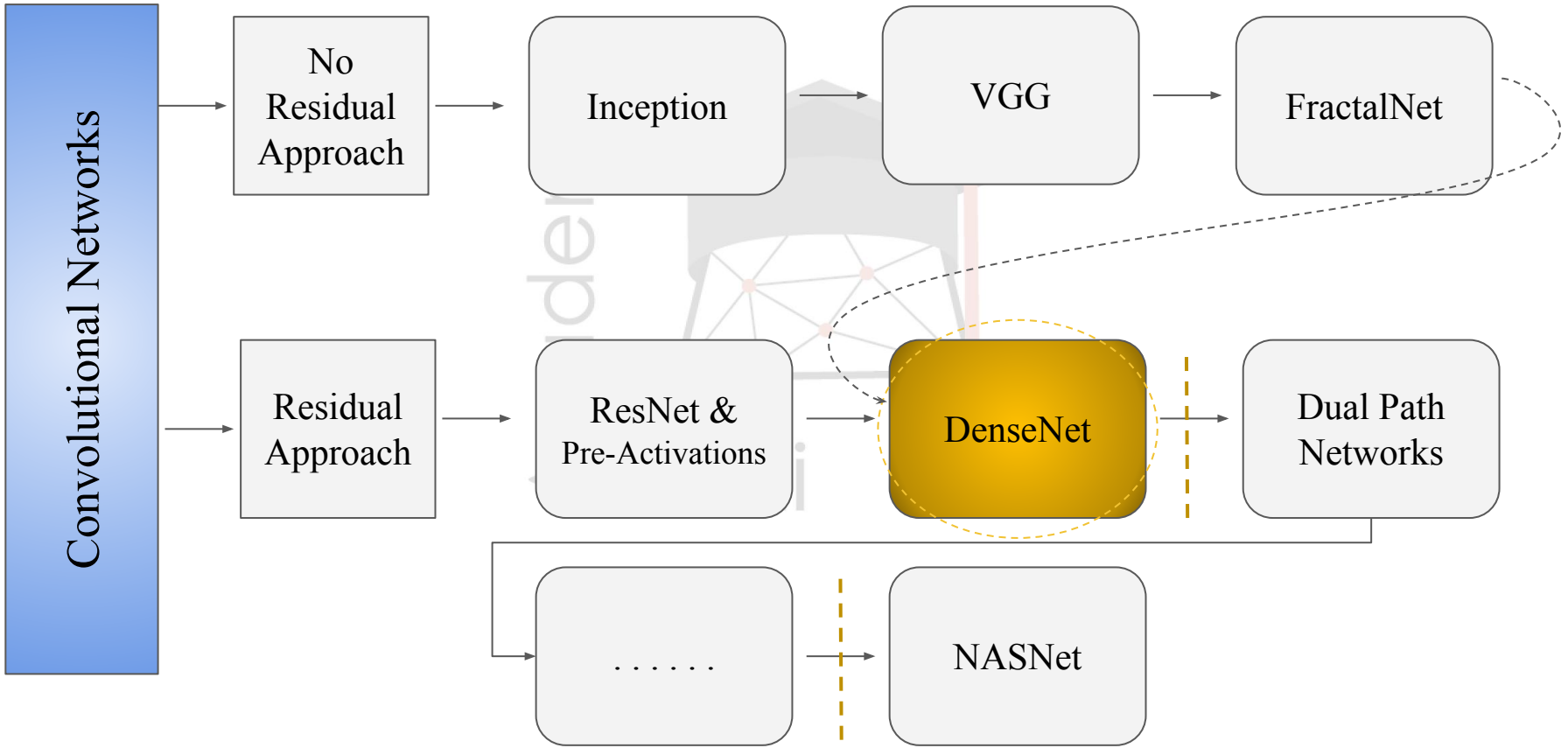
➔ Empirical results shows that updates don't occur uniformly across all the layers

➔ While Deep Residual Neural Networks can achieve the depth of *152* layers, paths are usually of 20-30 layers!

(*According to ensemble studies)*

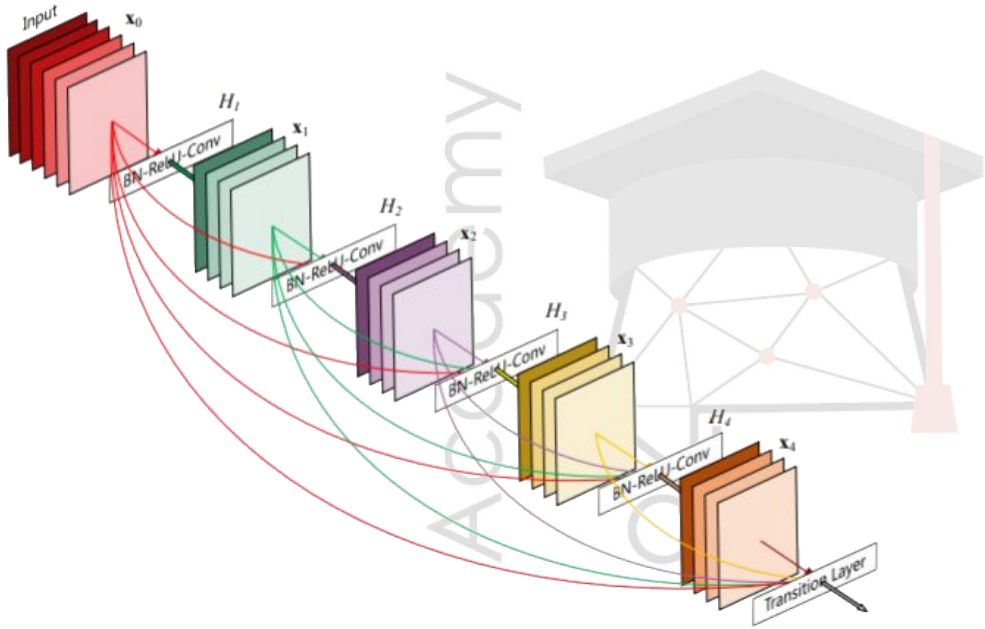


# Where research in *universities* and *academic world* is going?



# Dense Convolutional Networks [DenseNet]

DenseNet




*Why DenseNet?*

➔ Better performance that ResNet & Pre-Activation Resnet

➔ *Less parameters*

*How?*

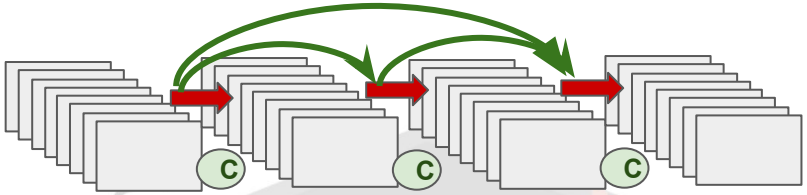
➔ ***DenseNet*** *simplify the connectivity pattern between layers introduced by* ResNet and FractalNet

Densely Connected Convolutional Networks, Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger

# DenseNet - Basic Idea behind

DenseNet

***DenseNet Approach***




C Channel-wise concatenation

***Alternative Approach*** *: Not always deep or wide architecture is the solution to our problem.. even if the field is called Deep Learning!! (CapsNet can be another example of this approach!)*

*DenseNet's performances depends on its ability to select features of all kind of complexity!*

***What about parameters?*** If we connect all these layers *DenseNet* should be harder to train despite ResNet

***Wrong!*** because there is no need to learn redundant features maps! So that our kernels inside each convolutional layer is narrow, around 12 convolutions x layer!

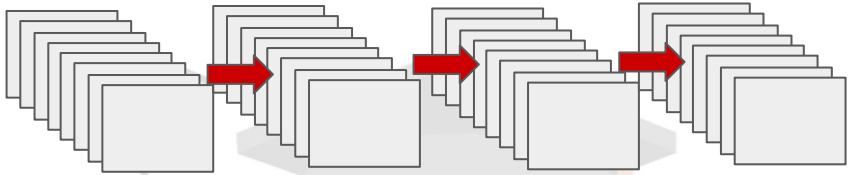
***What about training issues?***
Typical problems with NN consists in gradients computation during backpropagation phrase

With *DenseNet* each layer has direct access to gradients not only of previous layer but up to input image

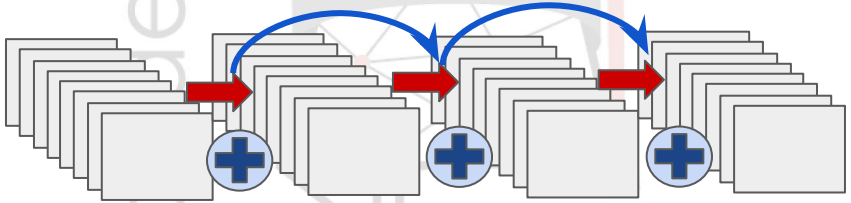
# DenseNet vs CNN & ResNet Approach

DenseNet

***Standard CNN***

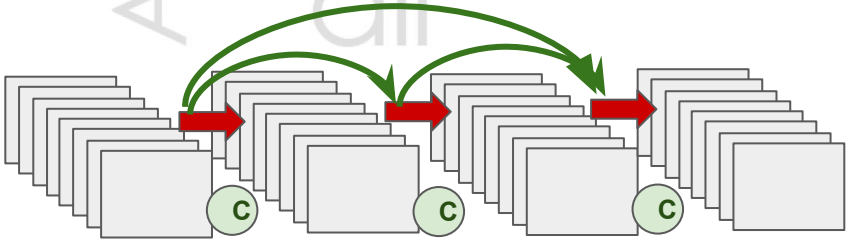


***ResNet Approach***



Element-wise addition

***DenseNet Approach***

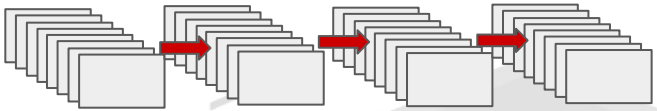



C Channel-wise concatenation

# Math : *CNN*

DenseNet

***Standard CNN***



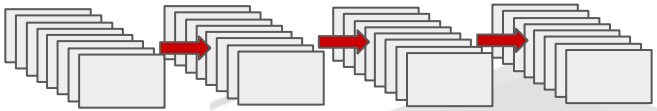
$$y_3 = f_3(f_2(f_1(y_0)))$$

$$y_3 = f_3(y_2)$$

# Math : CNN, *ResNet*

DenseNet

***Standard CNN***



$$y_3 = f_3(f_2(f_1(y_0)))$$
$$y_3 = f_3(y_2)$$

***ResNet Approach***

$$y_3 = f_3(y_2) + y_2$$

⊕ Element-wise addition

# Math : CNN, ResNet, ***DenseNet***

DenseNet

***Standard CNN***

$$y_3 = f_3(f_2(f_1(y_0)))$$
$$y_3 = f_3(y_2)$$

***ResNet Approach***

$$y_3 = f_3(y_2) + y_2$$

⊕ Element-wise addition

***DenseNet Approach***


$$y_3 = f_3([y_0, y_1, y_2])$$

C Channel-wise concatenation

# DenseNet - Architecture & Math

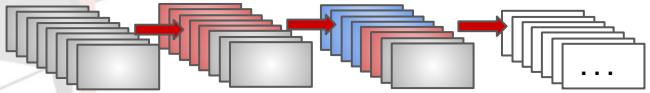
DenseNet

**DenseNet Approach**


$$y_3 = f_3([y_0, y_1, y_2])$$

C Channel-wise concatenation

*Do we have any kind of constraints between convolutional layers and block?*




*Like ResNet,* each group of feature maps in each block must have same dimensions *but we have no constraints on number of filters*

***Our features maps dimensions (k) increase layer by layer, how can we keep track of them?***

***growth rate :***

$$k_n = k_0 + k * (n-1)$$

$$\rightarrow k_3 = k_0 + k * (3-1)$$

Between Blocks we can apply our *downsampling* and *batch normalization* called **Transition Layer**:

- ➔ *(1x1) convolution*
- ➔ *(2x2) max-pooling*
- ➔ *batch normalization*

# DenseNet Paper

DenseNet



## Densely Connected Convolutional Networks

Gao Huang*
Cornell University
gh349@cornell.edu

Zhuang Liu*
Tsinghua University
liuzhuang13@mails.tsinghua.edu.cn

Laurens van der Maaten
Facebook AI Research
lvdmaaten@fb.com

Kilian Q. Weinberger
Cornell University
kqw4@cornell.edu

**Abstract**

*Recent work has shown that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output. In this paper, we embrace this observation and introduce the Dense Convolutional Network (DenseNet), which connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections—one between each layer and its subsequent layer—our network has $\frac{L(L+1)}{2}$ direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters. We evaluate our proposed architecture on four highly competitive object recognition benchmark tasks (CIFAR-10, CIFAR-100, SVHN, and ImageNet). DenseNets obtain significant improvements over the state-of-the-art on most of them, whilst requiring less computation to achieve high performance. Code and pre-trained models are available at https://github.com/liuzhuang13/DenseNet.*

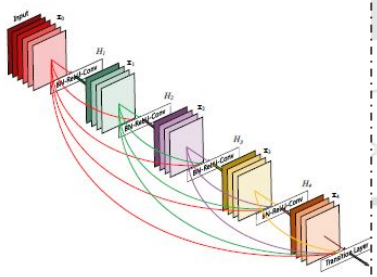



**Figure 1:** A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

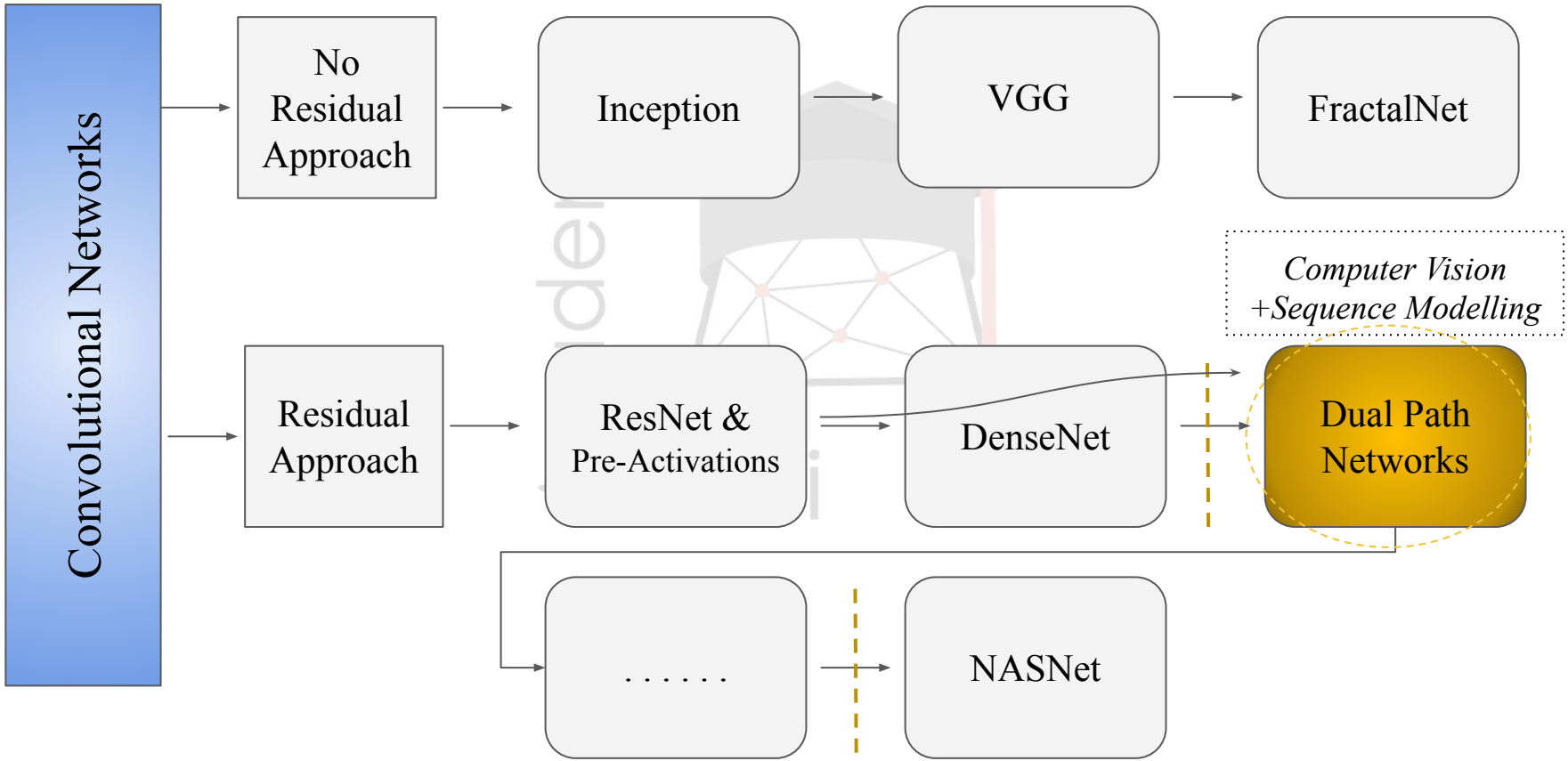
Networks [34] and Residual Networks (ResNets) [11] have surpassed the 100-layer barrier.

As CNNs become increasingly deep, a new research problem emerges: as information about the input or gradient passes through many layers, it can vanish and "wash out" by the time it reaches the end (or beginning) of the

*10 minutes reading*

Densely Connected Convolutional Networks, Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger

# Where research in *universities* and *academic world* is going?



# Brief overview : Dual Path Networks

DPN

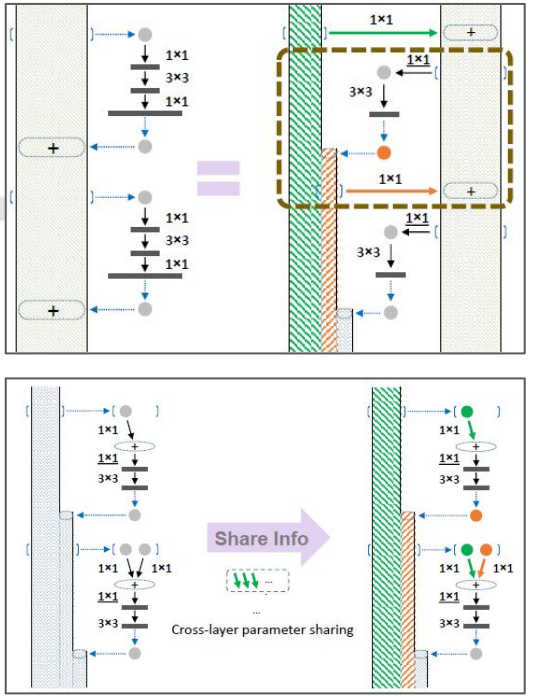
*"You don't need to reinvent the wheel"*

➔ *ResNet :* Representation power from depth
➔ *DenseNet :* Features Reuse

*Why don't we combine both proprieties?*

Rewriting DenseNet as an Higher Order Recurrent Neural Network HORNN we can show that :

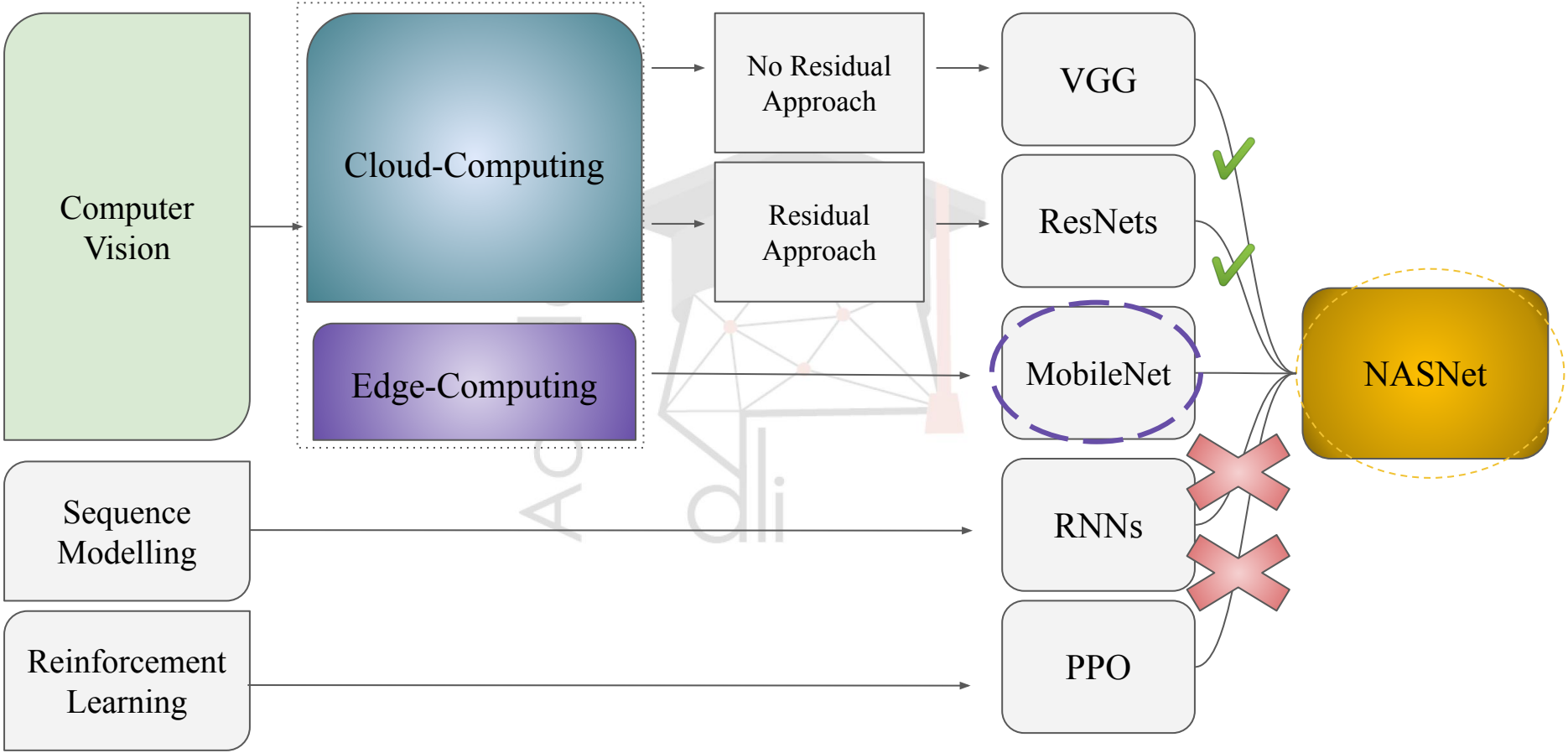
*ResNets* are a sort of *DenseNets with shared connections constraints*



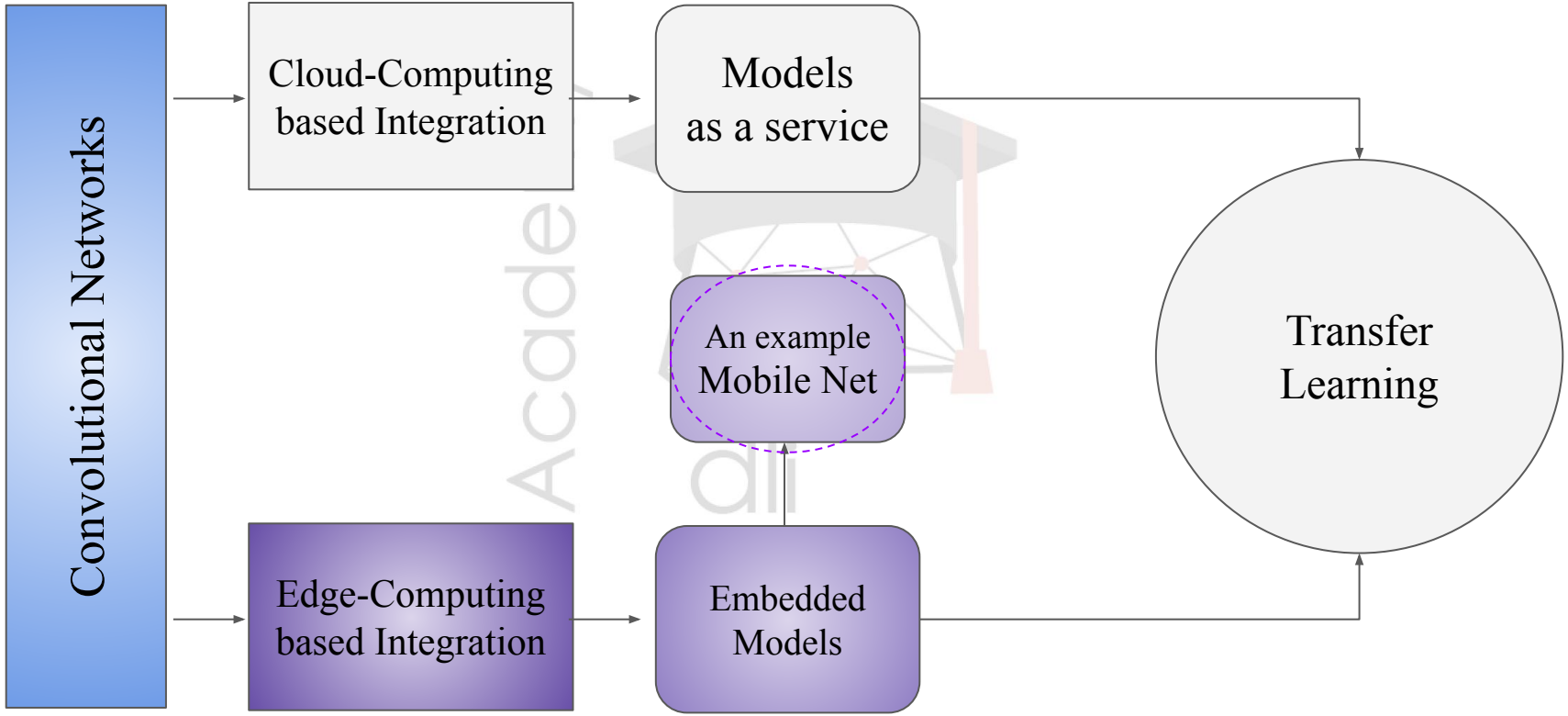

Dual Path Networks, Yunpeng Chen1 , Jianan Li1,2 , Huaxin Xiao1,3 , Xiaojie Jin1 , Shuicheng Yan4,1 , Jiashi Feng1, Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger

Higher Order Recurrent Neural Networks, Rohollah Soltani, Hui Jiang, Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger

# *Are we ready for NASNet?*



# Where research in *industries* is going?



# Brief Overview : Edge-Computing

Edge-Computing

***When we need it?***

Every time we have strong *Bandwidth and Latency* constraints
*(all embedded systems.. self driving cars but also like smartphones)*

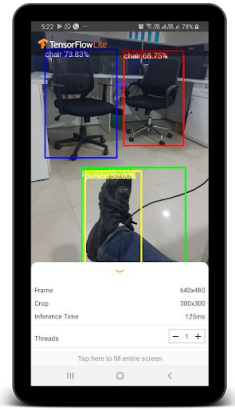
***What we need?***

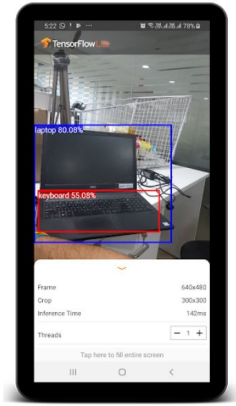
- ➔ Ability to work with network's shapes and parameters calculation
- ➔ Knowledge of *subnetworks*
- ➔ few other things

***What is the main difference?***

with embedded systems achieving the lowest *loss* isn't our first mission.
*What we need is speed!*
*(maybe without losing too much accuracy seems to be a good idea)*

→ *We need to be faster as possible in* ***inference****, not in training* ←




TensorFlow 2.0

TensorFlow Lite

# Neural Networks for Edge-Computing

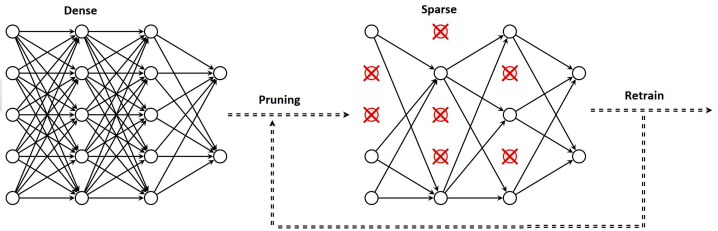
Edge-Computing

***We can adopt three kind of strategies due to lighten our networks***

**Pruning & Truncation :**

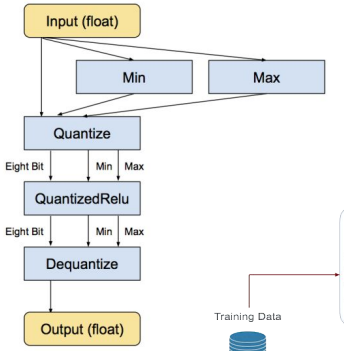
*As we have seen in different architectures a subnetwork can achieve accuracy levels similar to large networks. Many neurons don't contribute significantly*

➔ reduced up to **2x** in size while retaining **97%** of the original accuracy



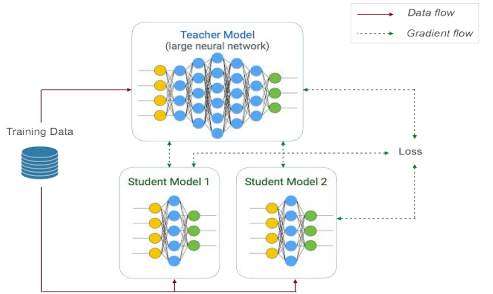

**Quantization :**

*Usually NN parameters are 32-bit float values but many embedded systems works with 8-bit encoding. Reducing a model encoded in 32-bit into 8-bit cut model size **x4***




**Distillation :**

*Smaller Networks trained from a large pretrained network*



# Convolutional Nets *for* Edge-Computing

Edge-Computing | Mobile Net

We have some ad-hoc convolutional networks for embedded systems application like :

- ➔ *SqueezieNet*
- ➔ *MobileNet (v1, v2)*

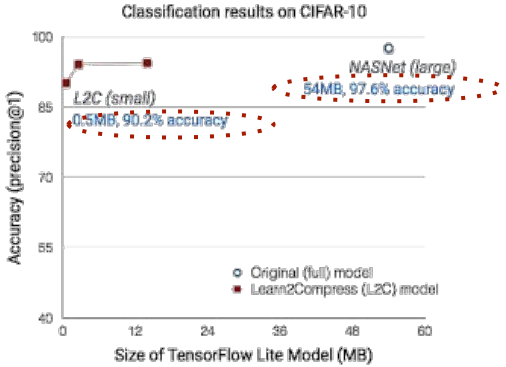
Table 8. MobileNet Comparison to Popular Models

| 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 |

**Raspberry Pi**

Google's Learn2Compress do exactly this!

If this practice works in collaboration with *transfer learning*, we can empirically demonstrate that the accuracy losing remains really low even with a strong the model size reduction



# Best Practices for Parameters Reductions : *MobileNet*

Mobile Net

***Depthwise Separable Convolution***

*(F x F) conv ↦ followed by (1 x 1) conv as dimensionality reduction*

F*F*C*Fm*Fm + C*O*Fm*Fm

F*F*C*O*Fm*Fm **Standard conv**

C : channels | Fm : features maps | O : output channels

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 |

$\frac{1}{O} + \frac{1}{K^2}$ *With a standard (3x3) receipt field we achieve a reduction of* ***8x***

MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam

# Best Practices for Parameters Reductions : *MobileNet*

Mobile Net

***Width Multiplier α for Thinner Models***

➔ control the input width of a layer ( α)

F * F *αC * Fm * Fm +αC *αO * Fm * Fm

*Where α is typical set between 0-1*

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 |

***Resolution Multiplier ρ for Reduced Representation***

➔ control the input image resolution (ρ)

F * F * α C *ρFm *ρFm + αC * αO *ρFm *ρFm

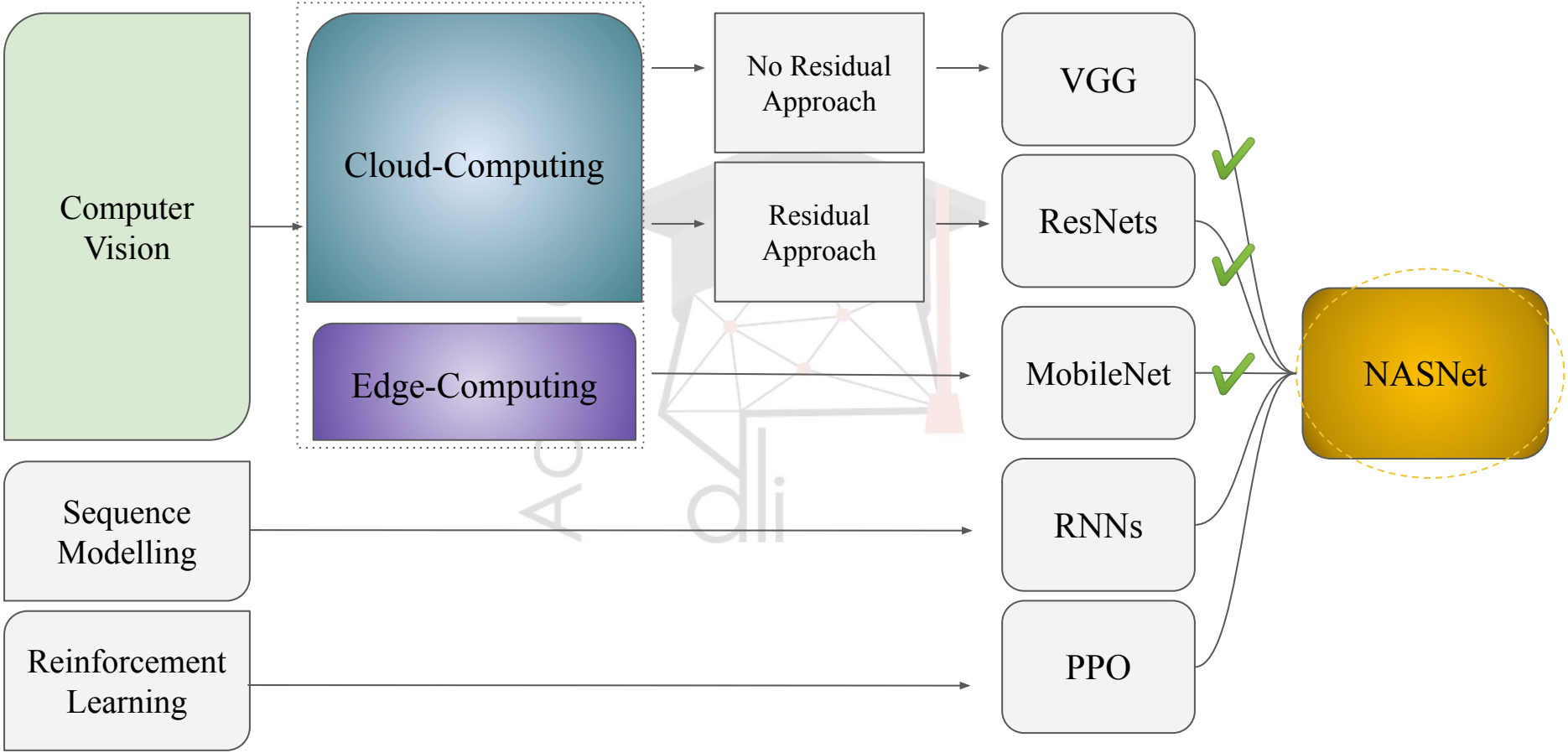
*Where ρ is typical set between 0-1*

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 |

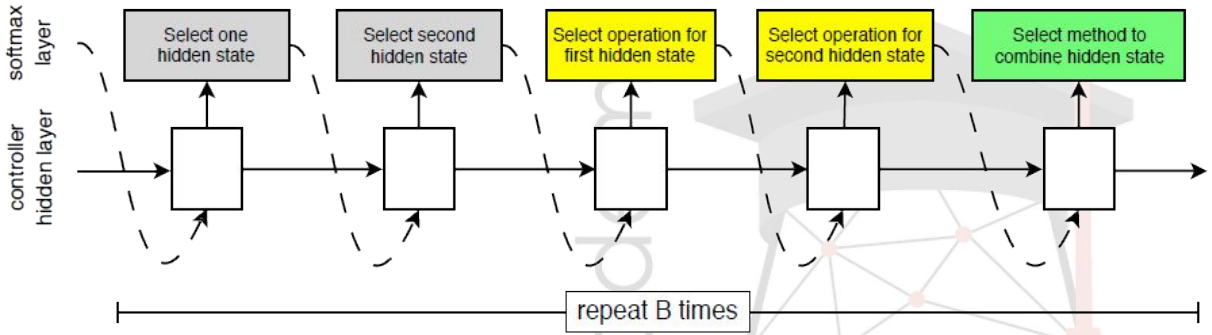
MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam

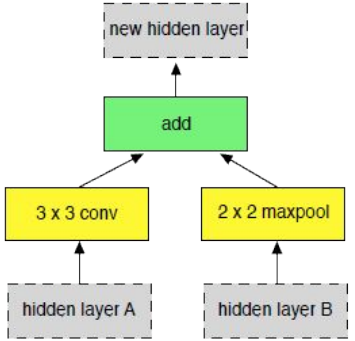
# *Are we ready for NASNet?*



# *Brief Overview :*

# Neural Architecture Search Network [NASNet]




*State-of-the-art with smaller models size and complexity, 2018 CVPR*

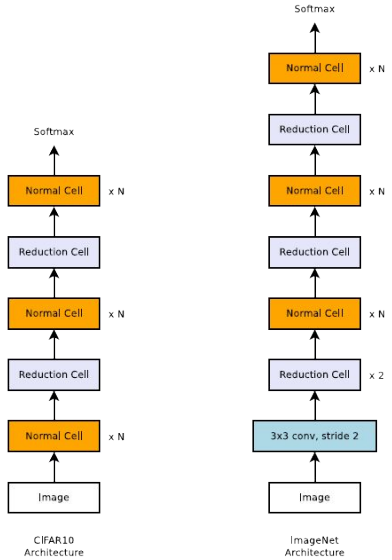
➔ Automate architectural search *(blocks)* on a small dataset *(RNNs + RL)*
➔ Transfer the to a larger dataset

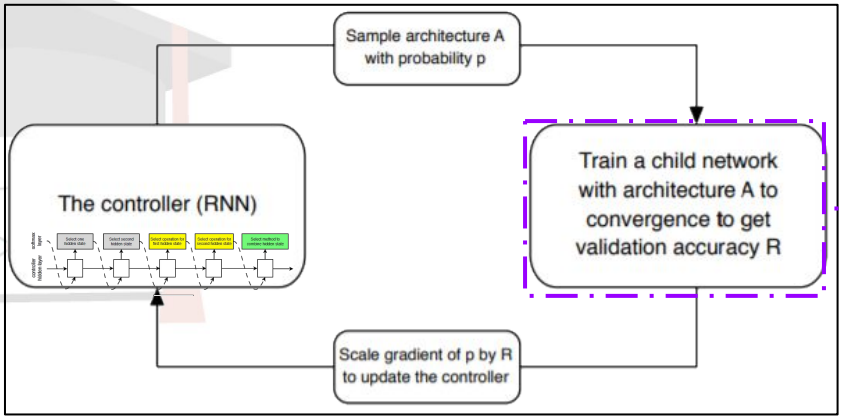
Learning Transferable Architectures for Scalable Image Recognition, Barret Zoph, Vijay Vasudevan, Jonathon Shlens, Quoc V. Le, Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger

# NASNet : *Computer Vision*

An example NASNet

Models as a service




*<u>Architecture is not defined</u>*

*We abstract the concept of layer with <u>Cells</u>*

- *Normal Cell*
- *Reduction Cell*

*Cells are grouped inside <u>Blocks</u>*

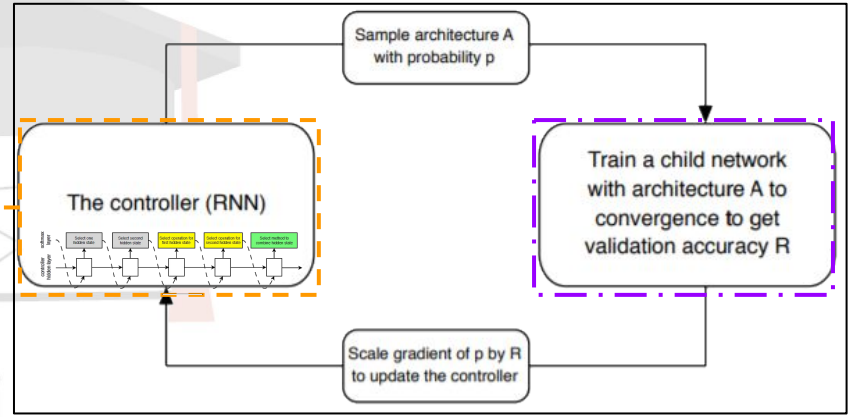
# NASNet : *Sequence Modelling*



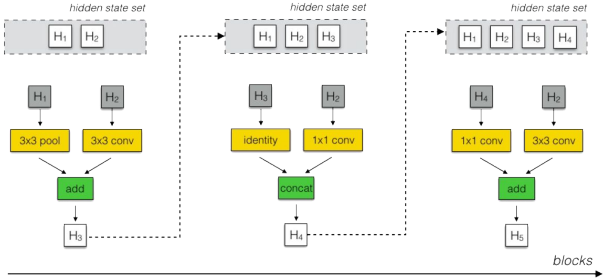
*Cell's structures and composition are searched by RNNs controller (LSTMs with 100 hidden units x layer)*

*For <u>One Block</u> :*

- *Given two initial hidden states (**hs**)*
- *Select two **hs***
- *Select operations for each **hs***
- *<u>Select method to combine **hs**</u>*



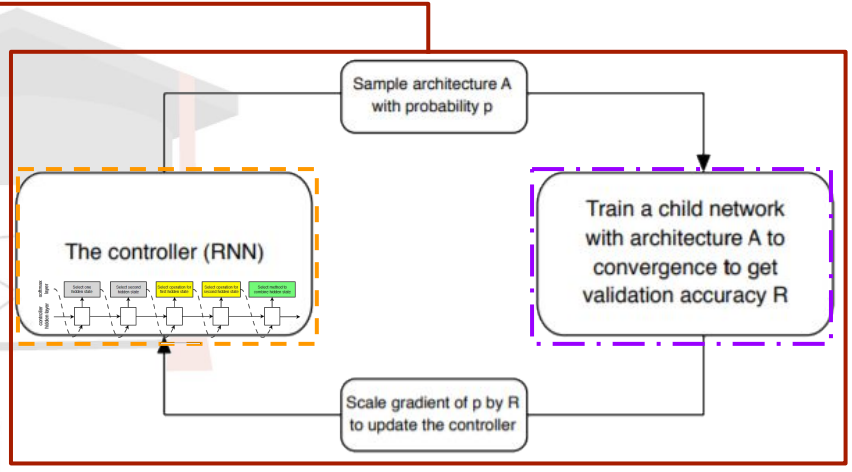

- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise-seperable conv



# NASNet : *Reinforcement Learning & Results*

Proximal Policy Optimization (PPO)

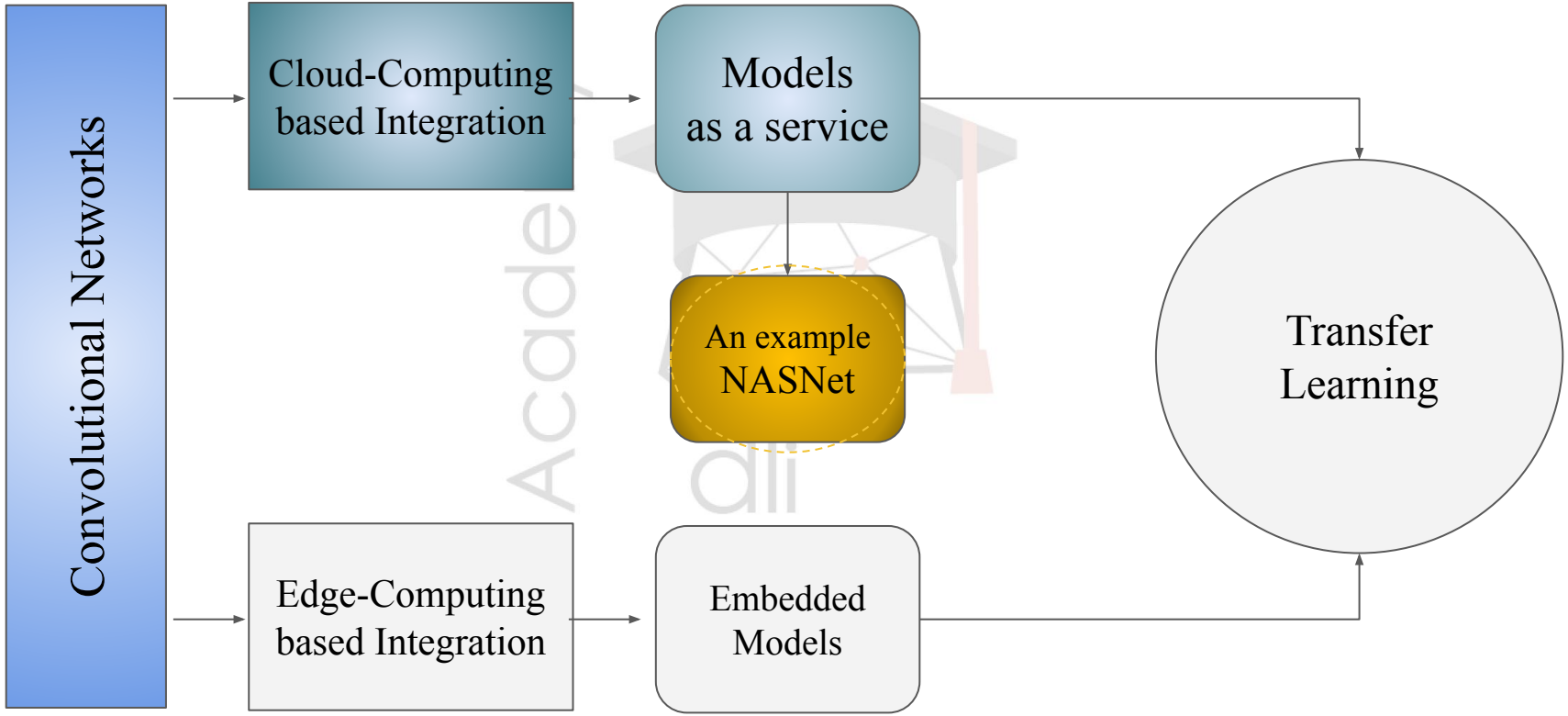
- *RL algorithm developed by OpenAI*
- *High performance but very simple to implement and tune*



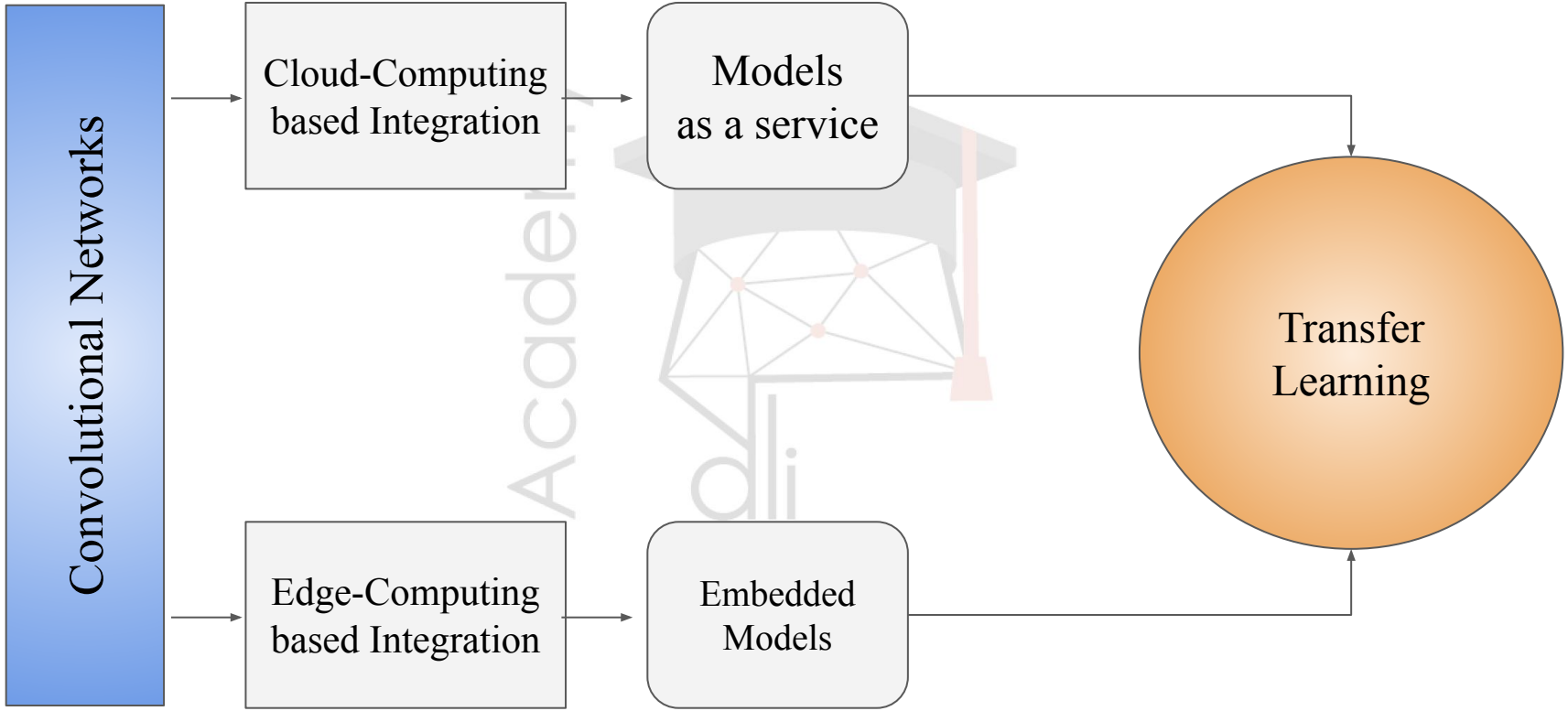

*Achieved after 2000 GPU-hours (500 GPUs)*

| model | depth | # params | error rate (%) |
|---|---|---|---|
| DenseNet ($L = 40, k = 12$) [26] | 40 | 1.0M | 5.24 |
| DenseNet($L = 100, k = 12$) [26] | 100 | 7.0M | 4.10 |
| DenseNet ($L = 100, k = 24$) [26] | 100 | 27.2M | 3.74 |
| DenseNet-BC ($L = 100, k = 40$) [26] | 190 | 25.6M | 3.46 |
| Shake-Shake 26 2x32d [18] | 26 | 2.9M | 3.55 |
| Shake-Shake 26 2x96d [18] | 26 | 26.2M | 2.86 |
| Shake-Shake 26 2x96d + cutout [12] | 26 | 26.2M | 2.56 |
| NAS v3 [71] | 39 | 7.1M | 4.47 |
| NAS v3 [71] | 39 | 37.4M | 3.65 |
| NASNet-A (6 @ 768) | - | 3.3M | 3.41 |
| NASNet-A (6 @ 768) + cutout | - | 3.3M | 2.65 |
| NASNet-A (7 @ 2304) | - | 27.6M | 2.97 |
| NASNet-A (7 @ 2304) + cutout | - | 27.6M | 2.40 |
| NASNet-B (4 @ 1152) | - | 2.6M | 3.73 |
| NASNet-C (4 @ 640) | - | 3.1M | 3.59 |

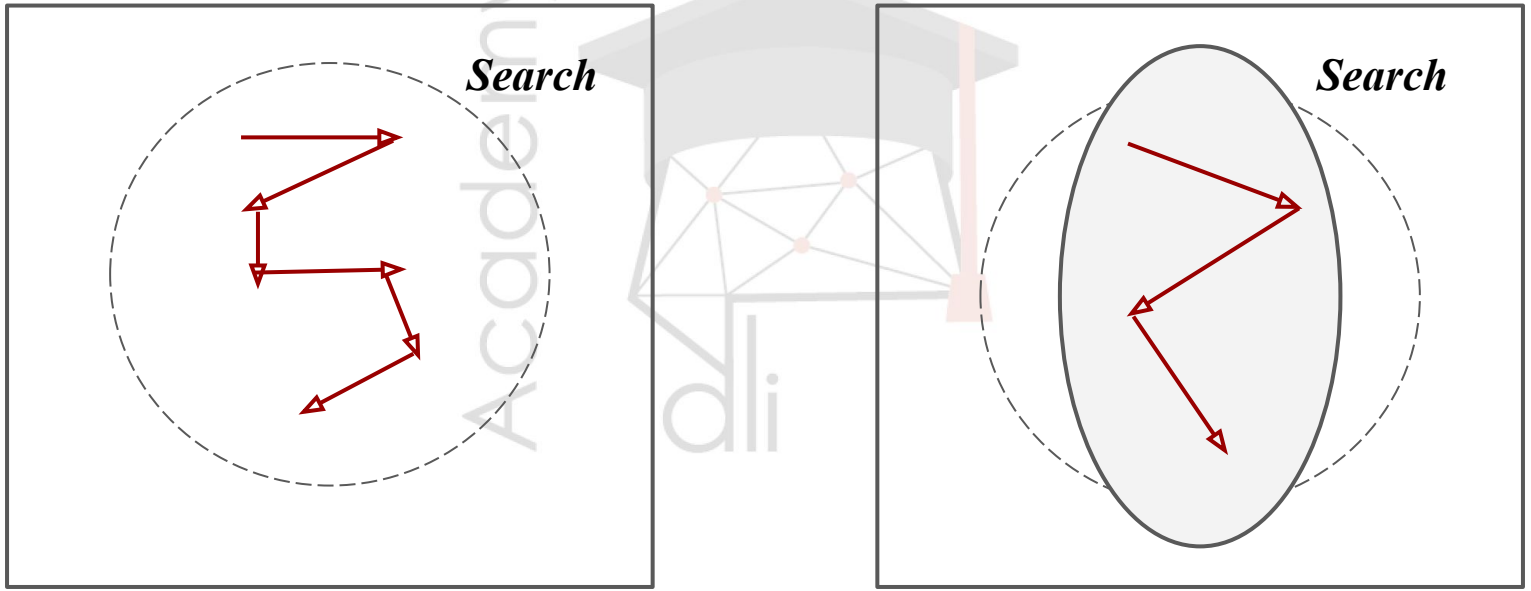
# Where research in *industries* is going?



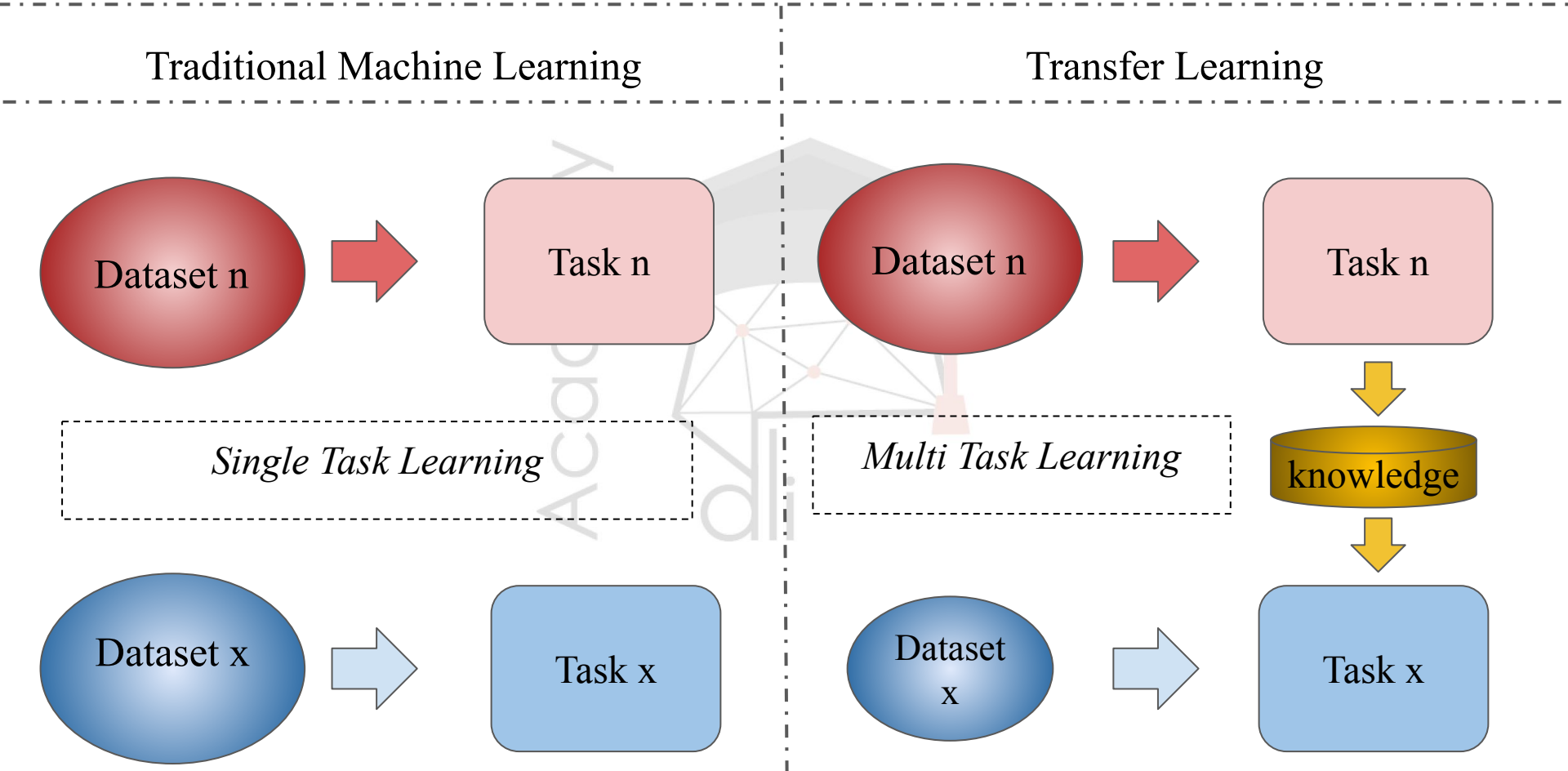
# Where research in *industries* is going?



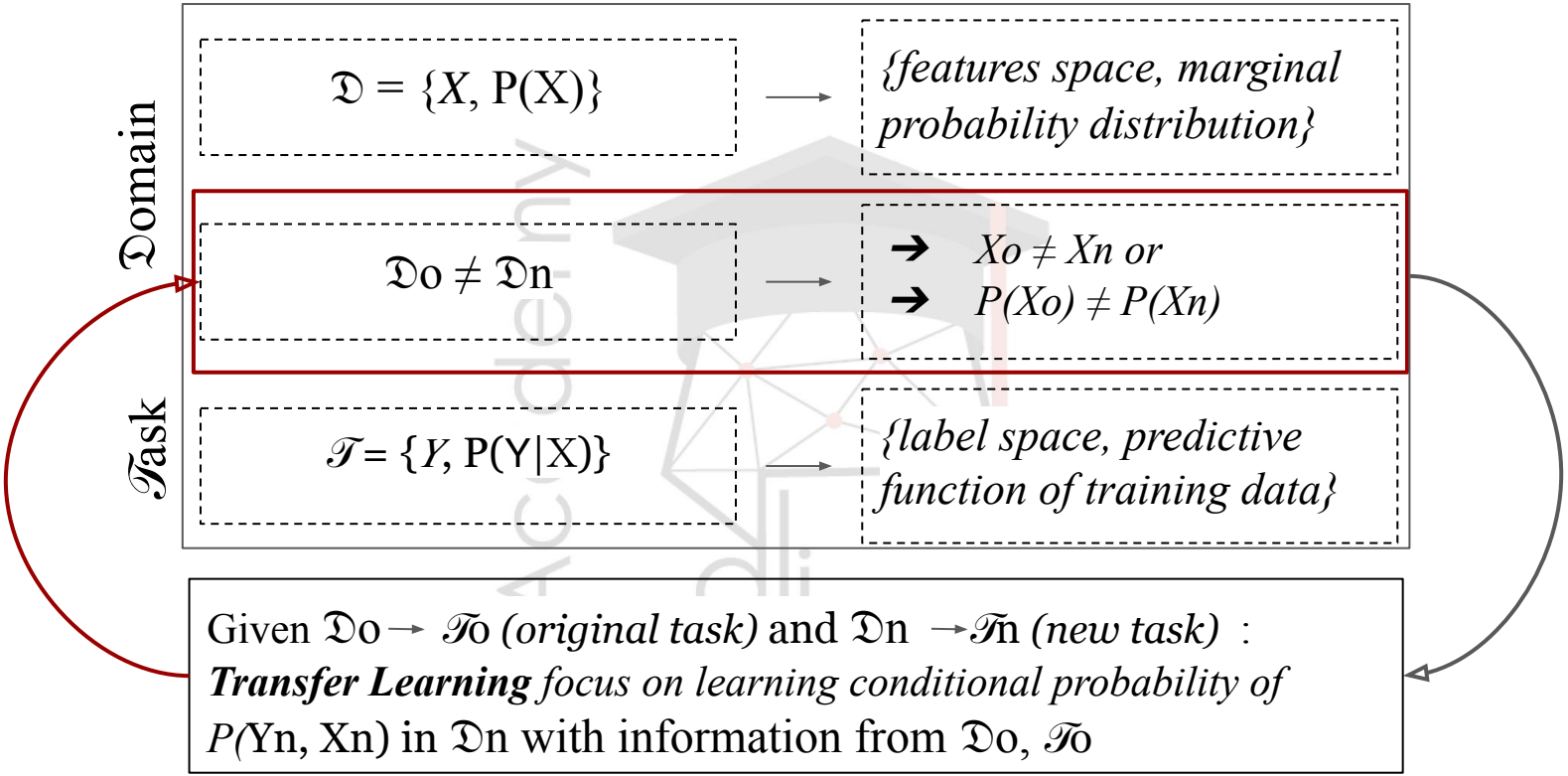
# A long peek into Transfer Learning



# Introduction to Transfer Learning



# Formal definition of Transfer Learning



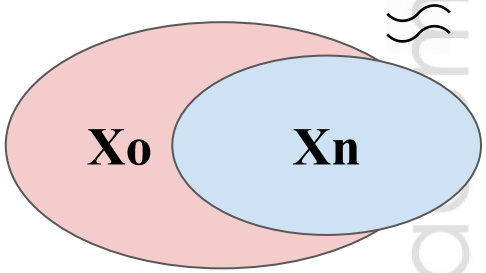
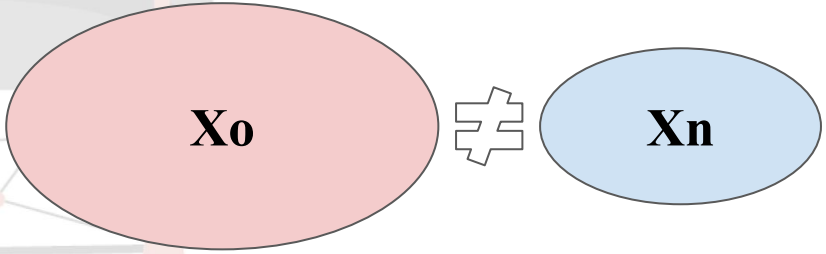
# Formal definition of Transfer Learning

*Examples*

| ***Xo ≠ Xn*** | ***P(Xo) ≠ P(Xn)*** |
| --- | --- |
| We have very different ***features space*** :<br><br>➔ Xo : road signs images<br>➔ Xn : medical images | We have similar ***features space*** :<br>➔ Xo : road signs images ***(day)***<br>➔ Xn : road signs images ***(night)***<br><br>*This is commonly defined as* ***Domain Adaptation*** |
| ***Yo ≠ Yn*** | ***P(Yo \| Xo) ≠ P(Yn \| Xn)*** |
| We have very different ***object classes*** :<br><br>➔ Xo : road signs images<br>➔ Xn : cars, motorcycle, . . . | When we haven't segmented pictures<br><br>➔ Xo : stylized man inside a colored circle :<br>◆ *road signs*<br>➔ Xn : stylized man :<br>◆ drawing of a child |

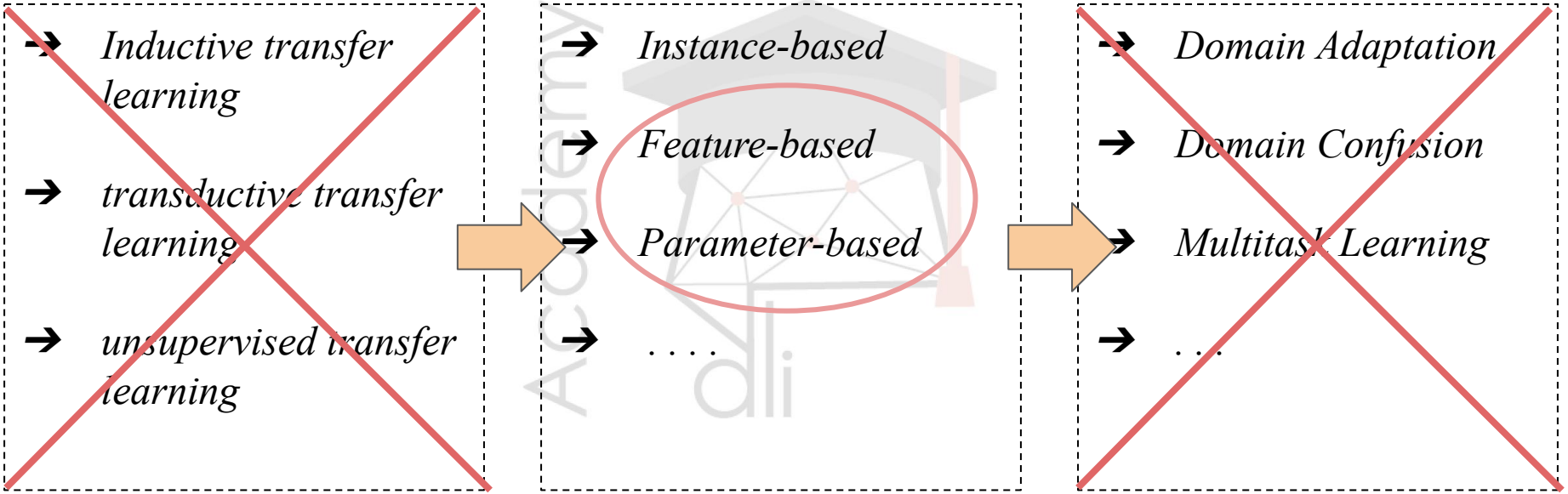
# Formal definition of Transfer Learning

*Categorization & Taxonomy*

| **Homogeneous TL** | **Heterogeneous TL** |
|---|---|
|  |  |
| Xo = Xn ***AND*** Yo = Yn<br>*P(Xo) ≠ P(Xn)* ***OR*** *P(Yo \| Xo) ≠ P(Yn \| Xn)* | Xo ≠ Xn ***AND/OR*** Yo ≠ Yn<br>*No features / labels sharing* |
| *Solutions :*<br>➔ correct *P(Xo) ≠ P(Xn)*<br>➔ *correct P(Yo \| Xo) ≠ P(Yn \| Xn)*<br>➔ *both* | *Solutions :*<br>➔ Try to reduce the gap between an Heterogeneous TL to Homogeneous TL |

# Formal definition of Transfer Learning

*Categorization & Taxonomy*



# Formal definition of Transfer Learning

*Categorization & Taxonomy*

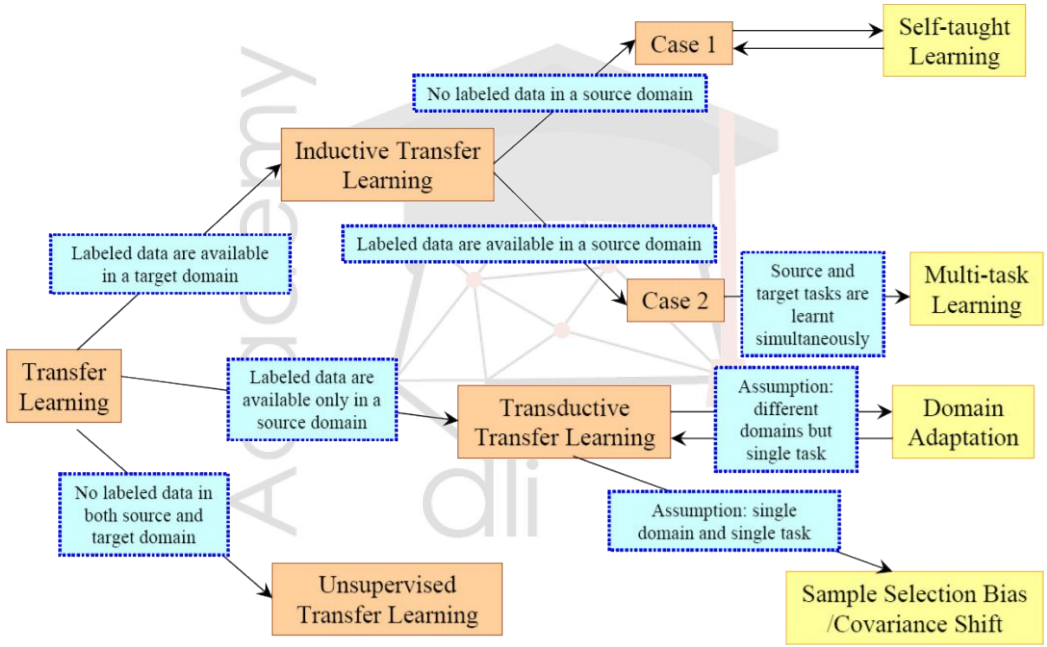



Figure from : [A Survey on Transfer Learning Sinno Jialin Pan and Qiang Yang Fellow, IEEE]

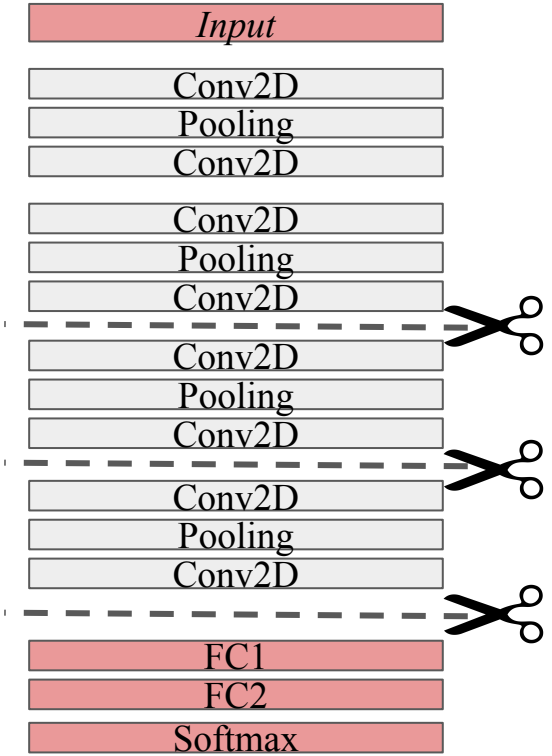
# Transfer Learning for Computer Vision

## *Strategies*

Consider a pretrained networks (like VGG or ResNet)

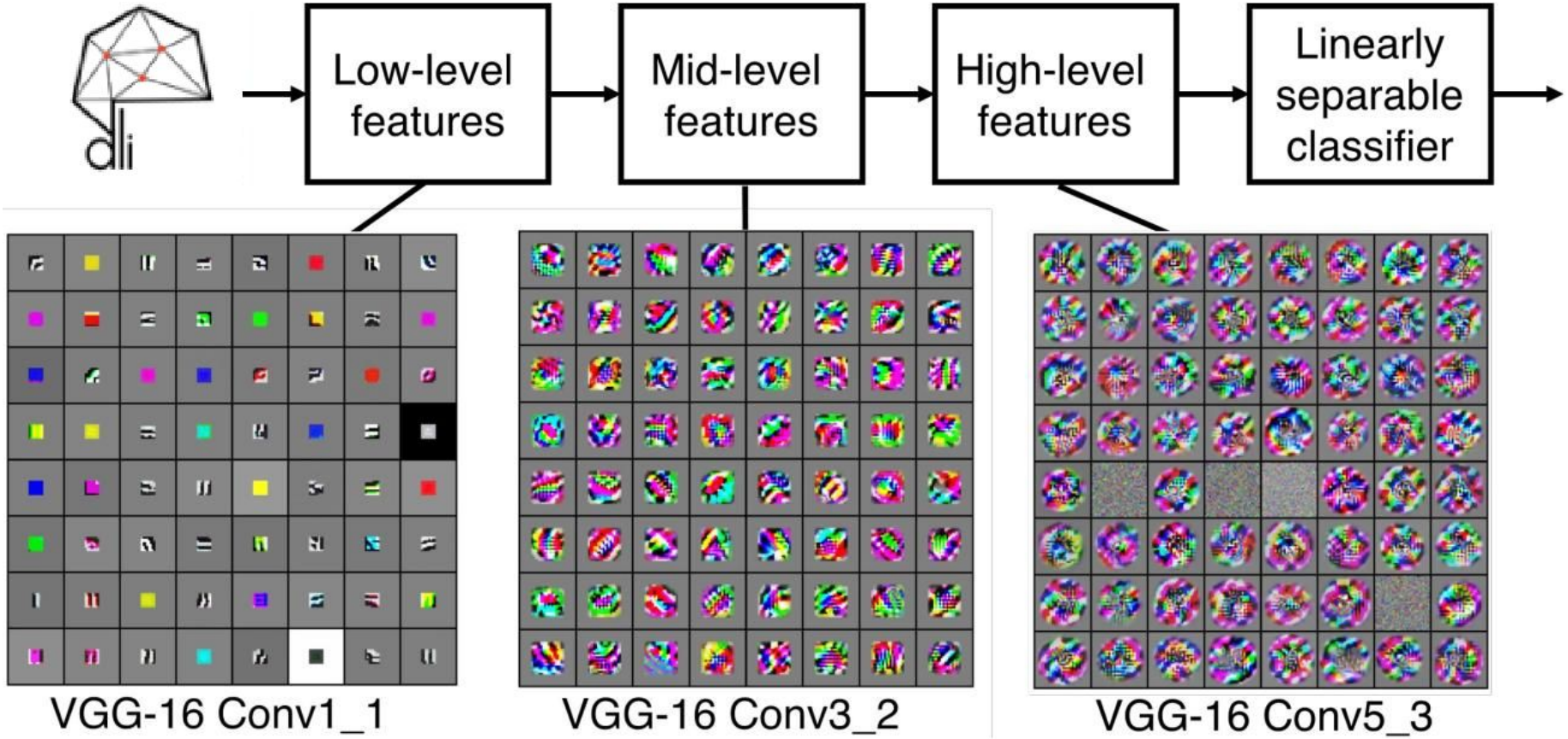
- ➔ Each Layer of our network contains trained weights useful for another task
- ➔ We can see them as an "*optimized" initialization of weights*
- ➔ In computer vision domain each convolutional layer works as "features extractor"

*Which features can be useful for our task?*

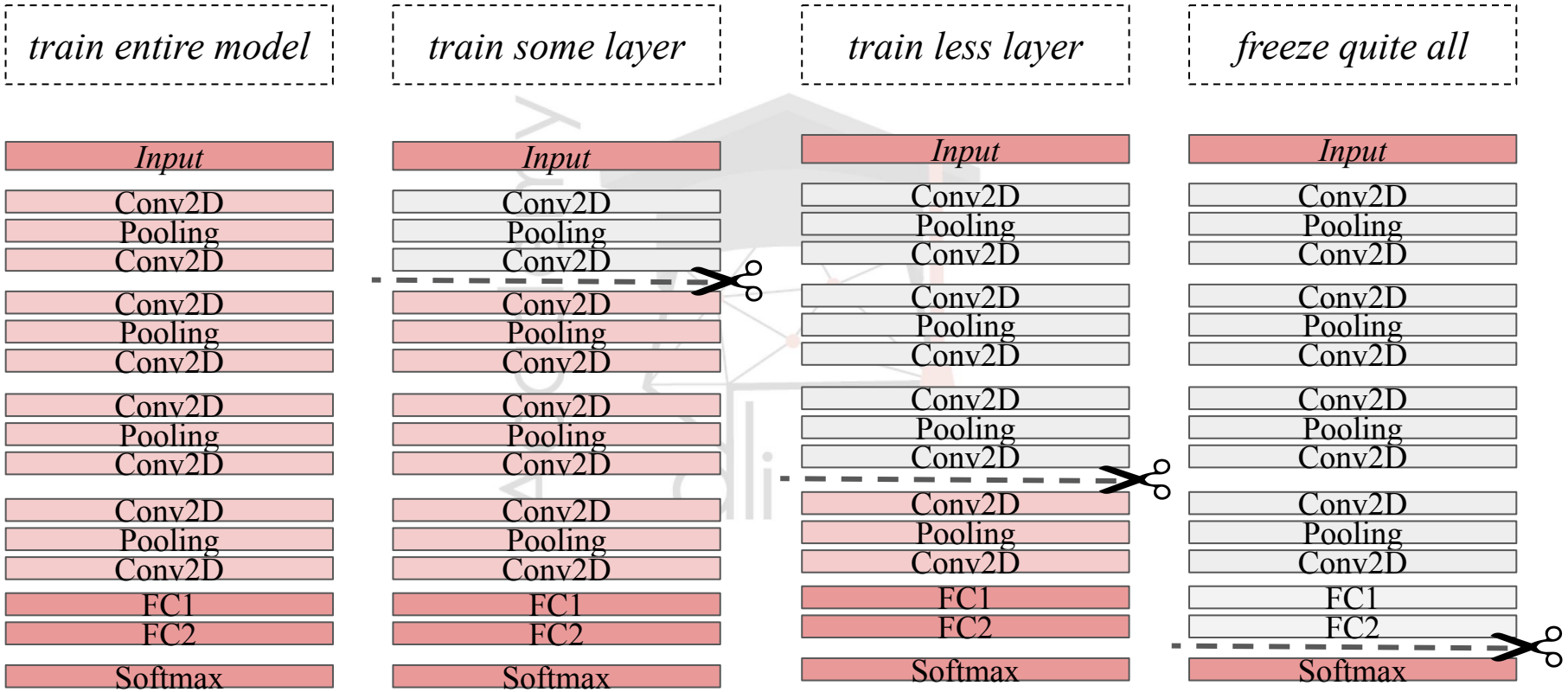



[How transferable are features in deep neural networks? Jason Yosinski, Jeff Clune, Yoshua Bengio, Hod Lipson]

# Pre Trained Models and Feature Extractors



# How Should I choose the right cut?



# How Should I choose the right cut?