Lecture 5: Intro to Transfer Learning: Basic Transfer Learning and SOTA Models

AC295

Advanced Practical Data Science
Pavlos Protopapas

Quote of the Day

"I am still learning"

- Michelangelo -

Outline

1: Communications

2: Recap

3: Motivation

3: The Basics idea for Transfer Learning

4: Representation Learning

5: Transfer Learning Strategies

6: Transfer Learning for Deep Learning

7: SOTA Deep Models

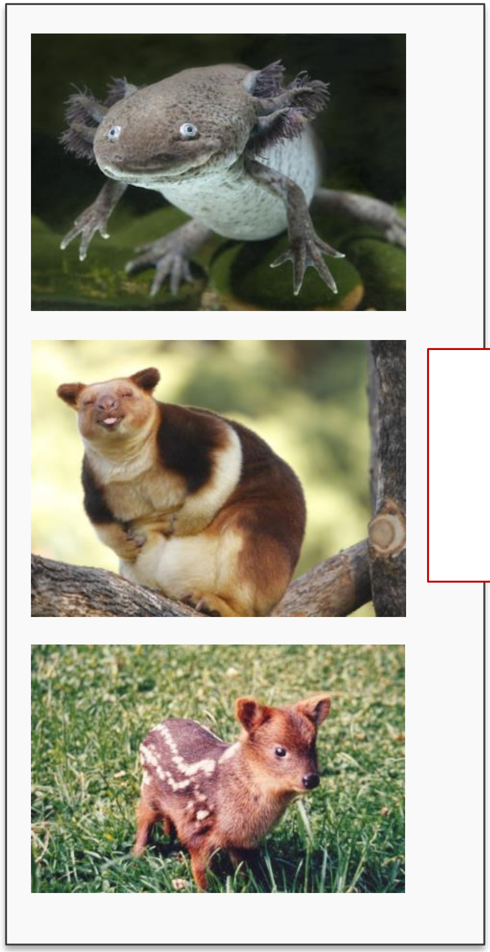
Communications

Feedback from week Practicum 1

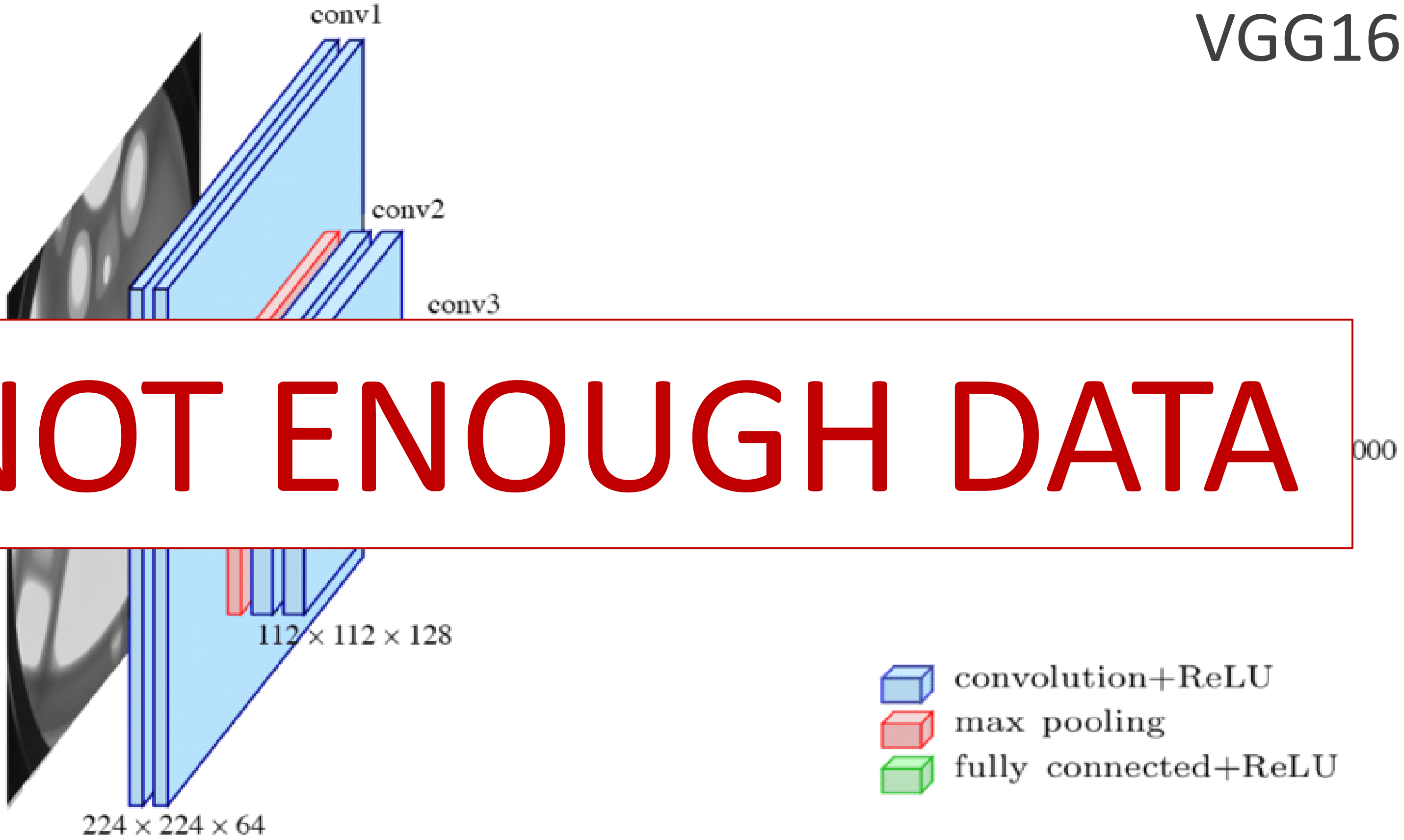
A. More

B. Difficulty

Classify Rarest Animals



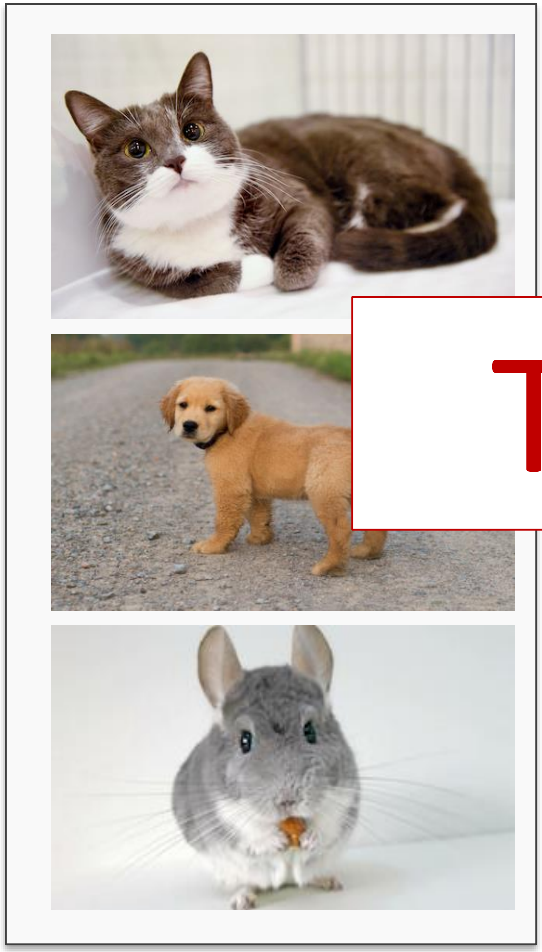
VGG16



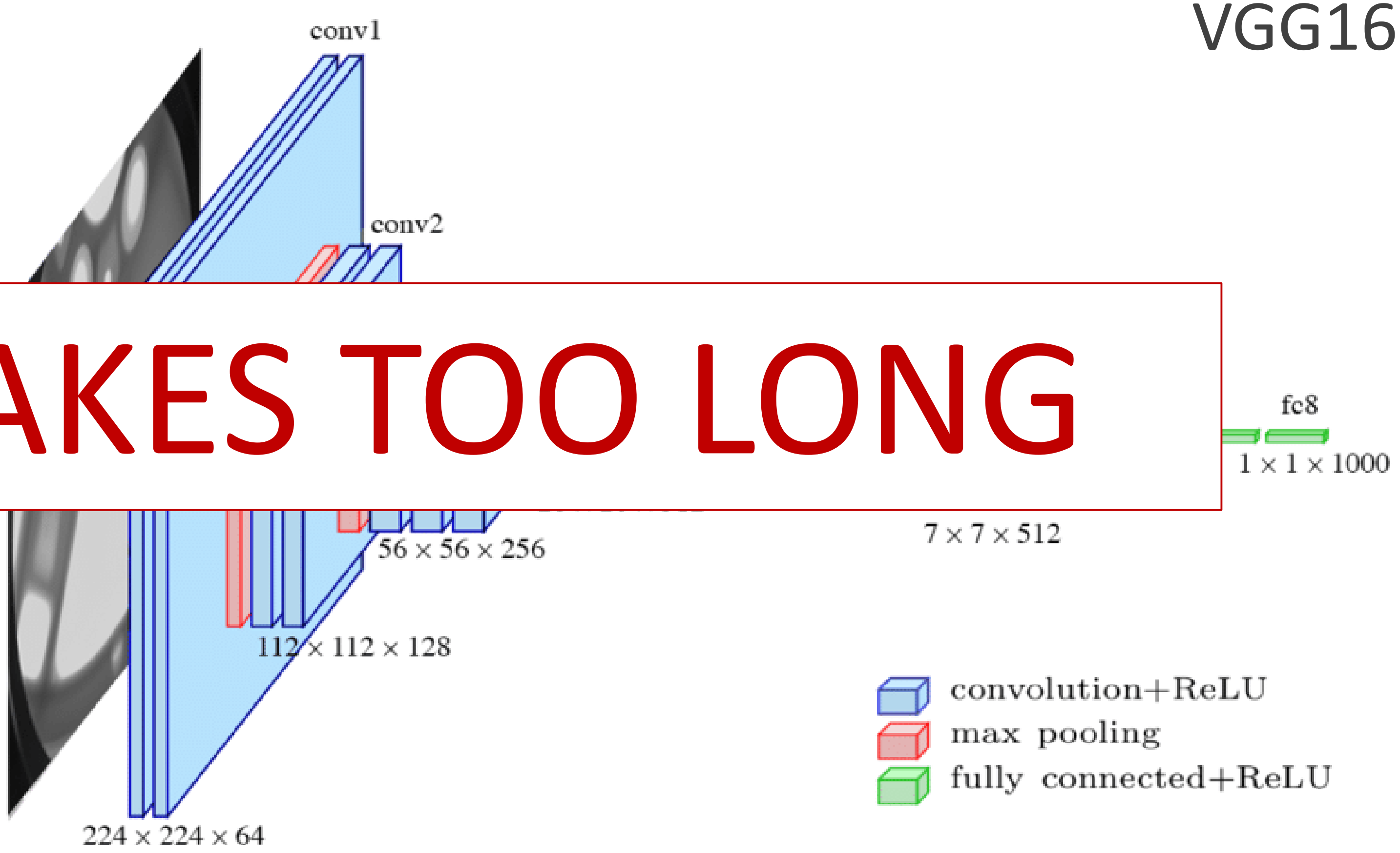
NOT ENOUGH DATA

Number of parameters: 134,268,737
Data Set: Few hundred images

Classify Cats, Dogs, Chinchillas etc



VGG16



TAKES TOO LONG

Number of parameters: 134,268,737
Enough training data. ImageNet approximate 1.2M

Transfer Learning To The Rescue

How do you build an image classifier that can be trained in a few minutes on a CPU with very little data?

Basic idea of Transfer Learning

Train a ML model M for a task T using a dataset D_s

Use M on a new dataset D_T for the same task T

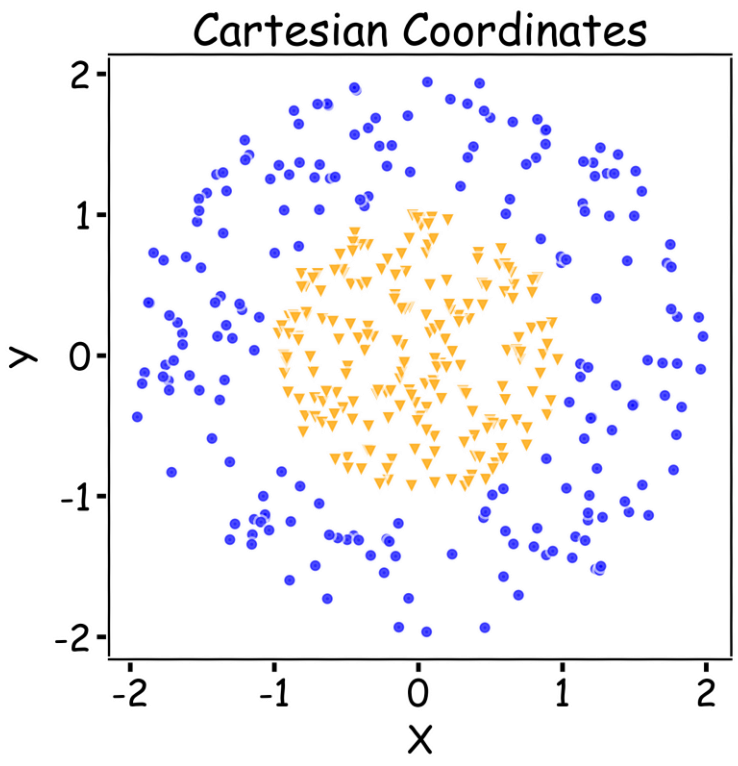
Use part of M on original dataset D_s for a new task T_n

Use part of M on a new dataset D_T for a new task T_n

Wikipedia:
Transfer learning (TL) is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem.[1]

Key Idea: Representation Learning

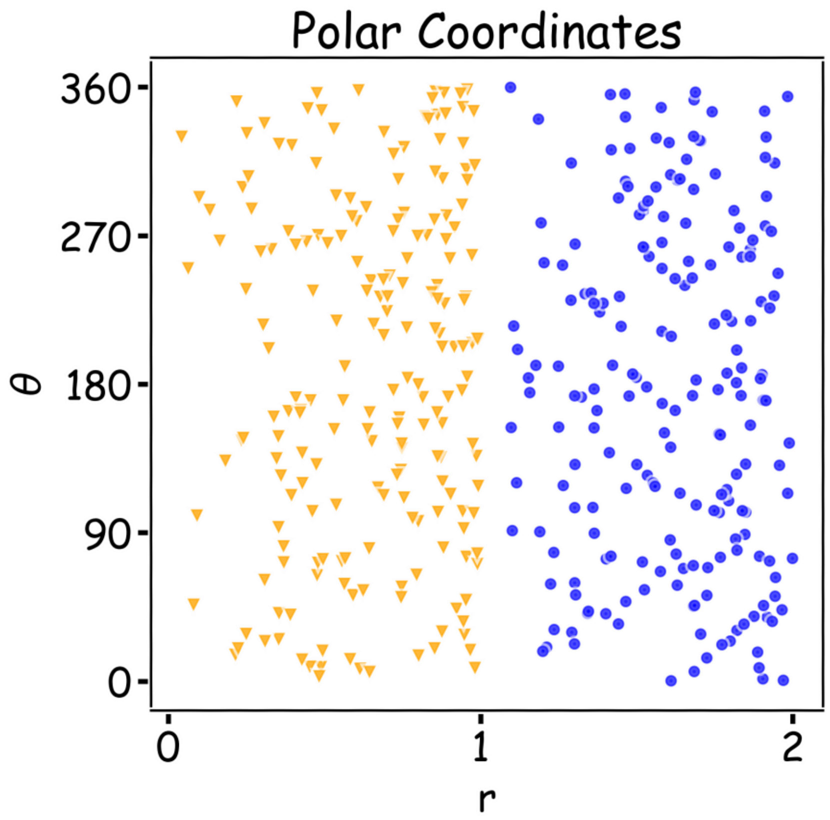
Relatively difficult task



Tranform:

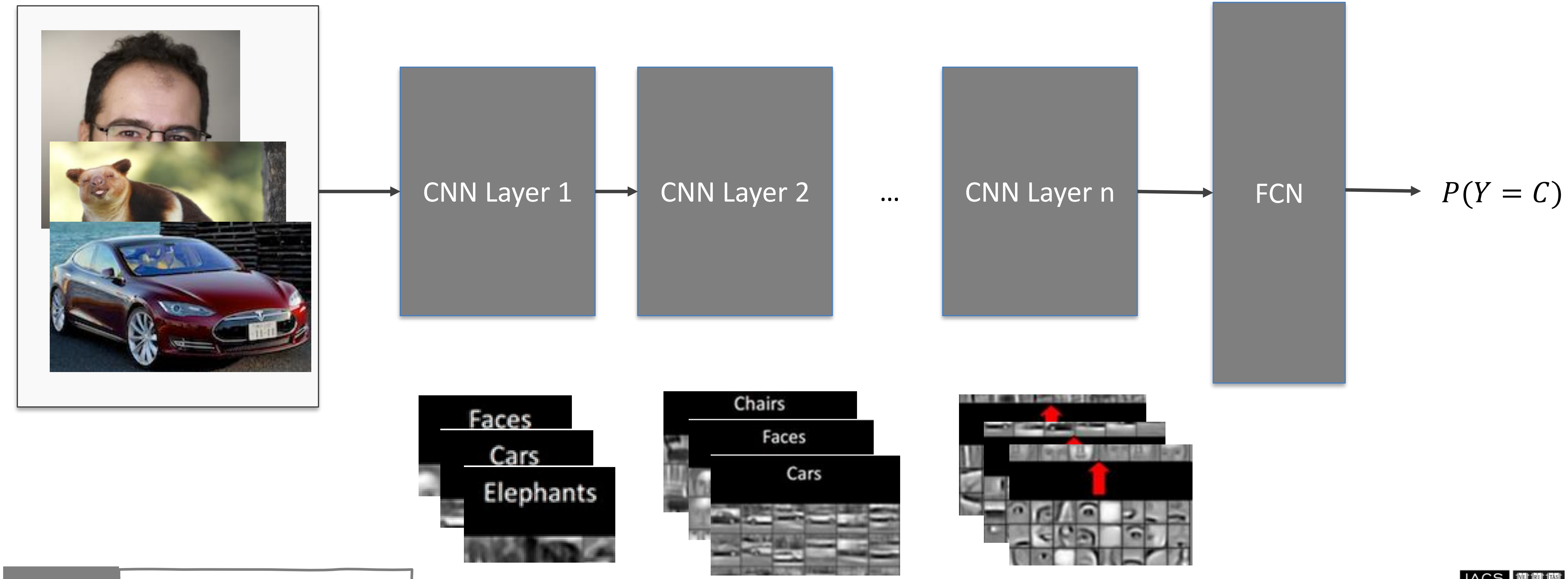
$(X, Y) \rightarrow (r, \theta)$

Easier task



Representation Learning

Task: classify cars, people, animals and objects



Transfer Learning To The Rescue

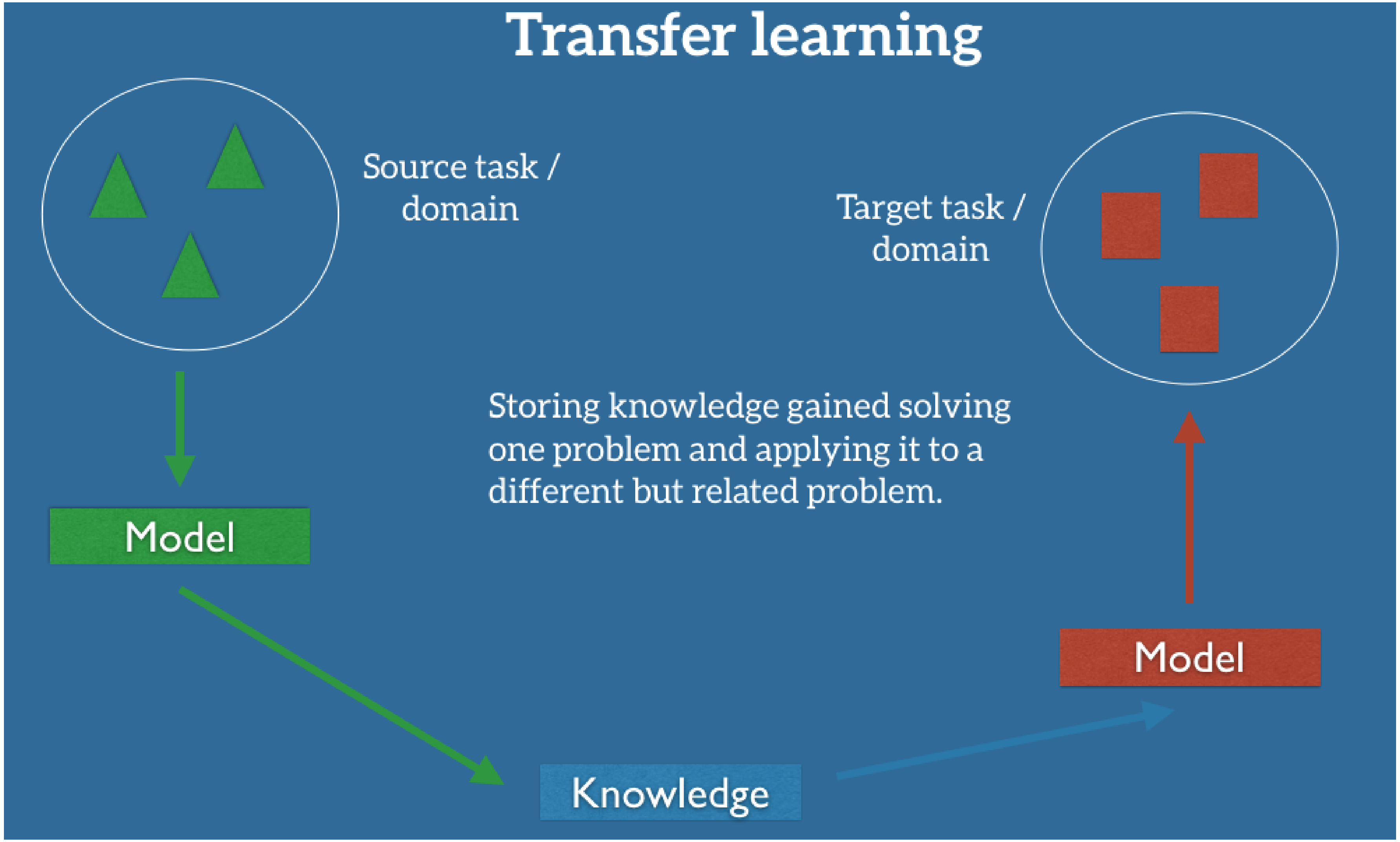
How do you make **an image classifier** that can be **trained in** a few minutes on a CPU with very little data?

Use pre-trained models, i.e., models with known weights.

Main Idea: earlier layers of a network learn low level features, which can be adapted to new domains by changing weights at later and fully-connected layers.

Example: use ImageNet trained with any sophisticated huge network. Then retrain it on a few images

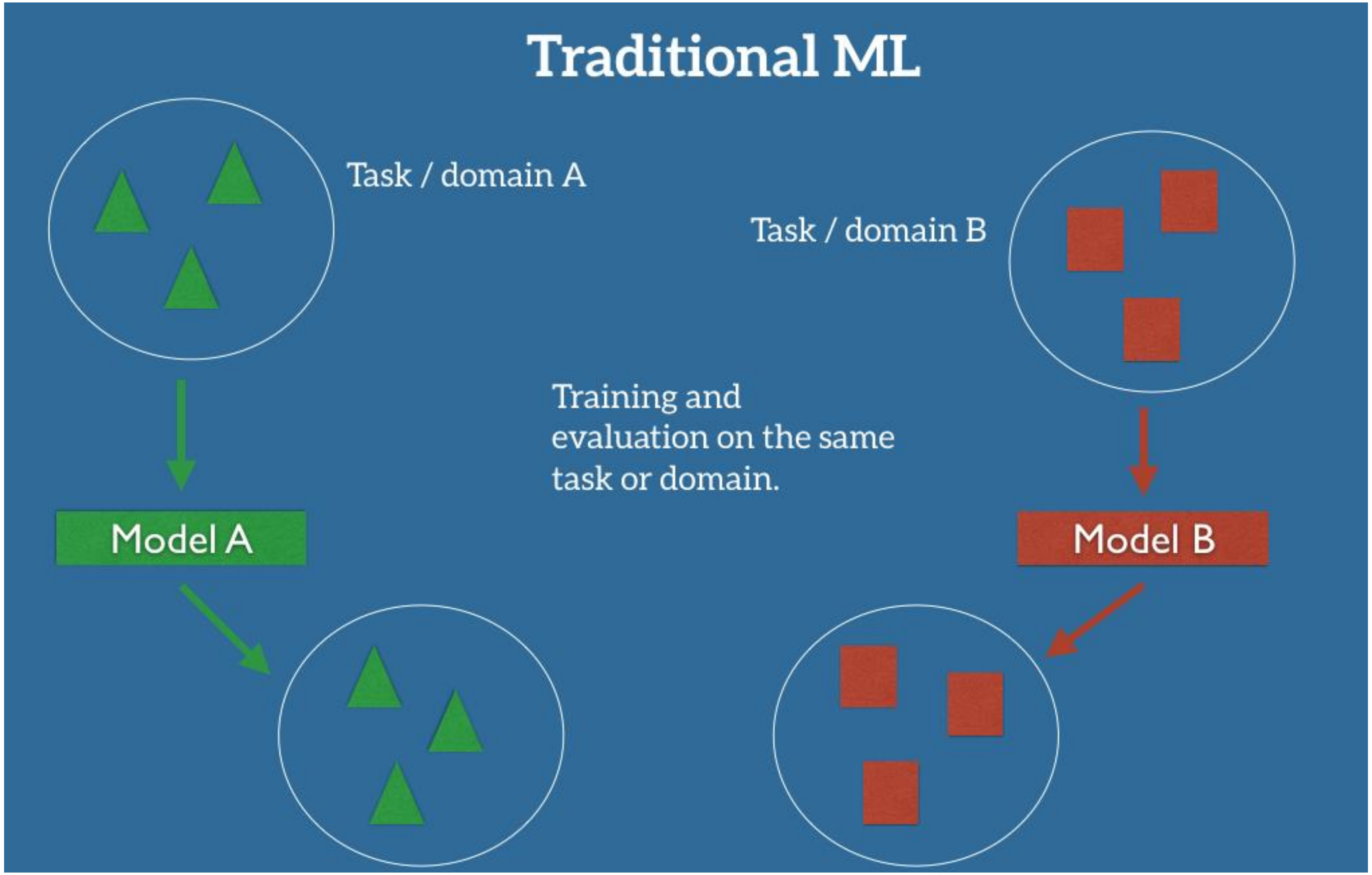
Machine Learning Setup



Transfer Learning To The Rescue

- train on a big "**source**" data set, with a big model, on one particular downstream tasks (say classification). Do it once and save the parameters. This is called a **pre-trained model**.
- use these parameters for other smaller "**target** " datasets, say, for classification on new images (possibly different **domain**, or training distribution), or for image segmentation on old images(new **task**), or new images (new task and new domain).
- less helpful if you have a large target dataset with many labels.
- will fail if source domain (where you trained big model) has nothing in common with target domain (that you want to train on smaller data set).

Transfer Learning



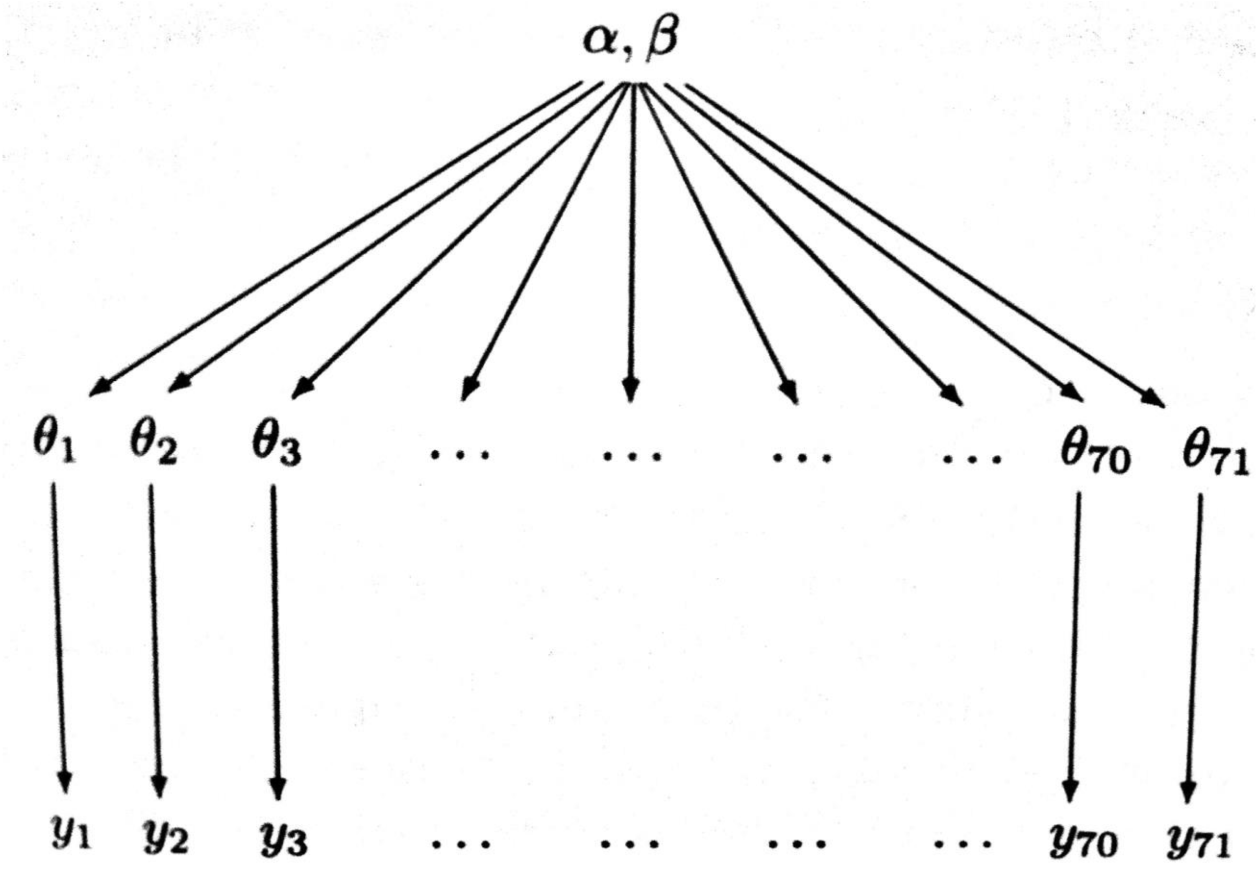
Transfer Learning

Not a new idea!

It has been there in the ML and stats literature for a while.

• an exemplar is hierarchical **glm models** in stats, where information flows from higher data units to lower data units to the lower data.

• neural networks **learn hierarchical representations** and thus are particularly suited to this kind of learning. Furthermore, since we learn representations, we can deal with domain adaptation/covariate shift.



Transfer Learning

Application

- learning from simulations (self driving cars)
- domain adaptation: bikes -> bikes with backgrounds, bikes at night, etc
- speech recognition for immigrants and minorities
- cross-lingual adaptation for few shot learning of resource poor languages (english->nepali for example)

Transfer Learning: using a pre-trained net

- create a classifier to distinguish dogs and cats

- use a convnet previously trained (expensive for you to learn)
    - e.g. Imagenet (1.4 M images and 1000 classes) >> more later

- in NLP, you might use a language model trained on Wikipedia and reddit, and then look at legal documents

A Formal Definition

A **Domain** consists of two components: $D = \{\chi, P(X)\}$

- Feature space: χ
- Marginal Distribution: $P(X); X = \{x_1 \ \ldots, x_n\}, x_i \in \chi$

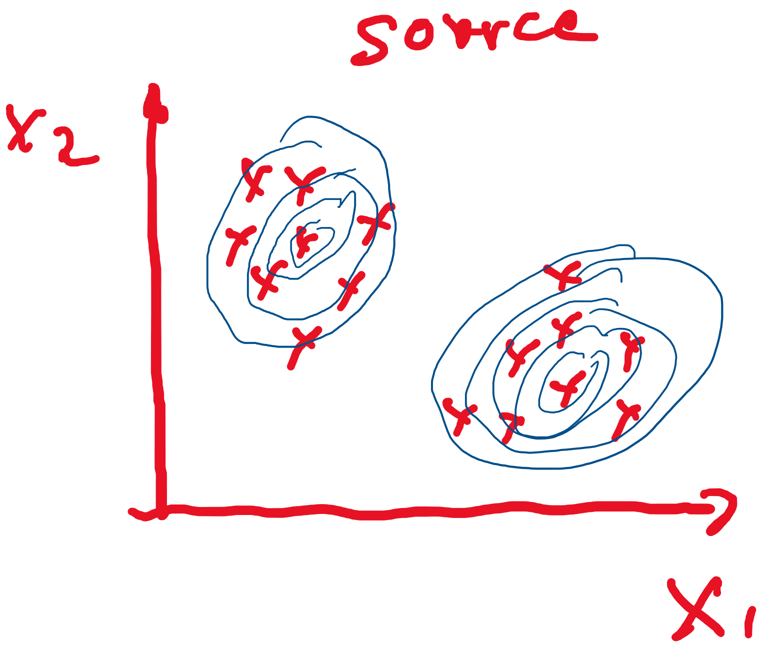
For a given Domain, a **Task** is defined by two components:

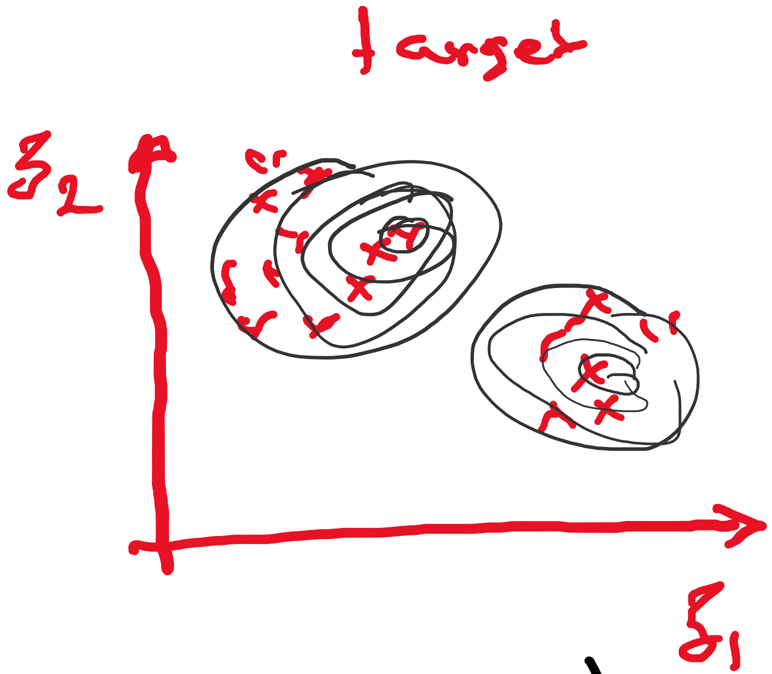
$$T = \{\mathcal{Y}, P(Y|X)\} = \{\mathcal{Y}, \eta); Y = \{y_1, \ldots, y_n\}, y_i \in \mathcal{Y}$$

- Label space: $\mathcal{Y}$
- A predictive function η, learned from feature vector/label pairs $(x_i, y_i), x_i \in \chi, y_i \in \mathcal{Y}$.
- For each feature vector in the domain, η predicts its corresponding label $\eta(x_1) = y_1$.

Pan and Yang, A Survey on Transfer Learning

Domain





$P_s(x_1, x_2)$

$x \subset \mathcal{X}$

$P_t(\xi_1, \xi_2)$

$\xi \subset \Xi$

if $\mathcal{X} = \Xi$ and $P_s(X) \neq P_t(X)$

Covariate shift or sample selection bias

Scenarios

Different features spaces among source and target

$$\chi_{source} \neq \chi_{target}$$

Scenario: document A – the source - is written in one language while document B – the target - is written in a different language

Task: can we use the weights learned training a model that distinguish phonemes on the source to distinguish those in the target written in another language? In NLP this method is called cross lingual adaptation.

Scenarios

Different marginal probabilities among source and target

$$P_s(x) \neq P_t(Xt)$$

Scenario: Consider two different telescopes, in which one is equipped with a sensor with higher sensitivity than the other.

Task: can we use the weights learned training a model learned training a model on the source data that distinguish topics in another target document?

Scenarios

Different labels among source and target

$$\mathcal{Y}_{source} \neq \mathcal{Y}_{target}$$

Scenario: farm animals versus wild forest animals or different level of classification, e.g. {dogs, cats} different breeds {Retriever, Bulldog, …, Persian, Siamese}.

Scenarios

Different conditional probabilities distribution among source and target task

$$P_s(Y|X) \neq P_t(Y|X)$$

Scenario: source and target are documents are unbalanced regarding their labels. Common scenario in practice, approachable with sampling techniques (e.g. under, over).

Probability Shift: $P_s(Y) \neq P_{\mathrm{t}}(\mathrm{Y})$ not the same class distribution
Conditional Shift: $P_s(X|Y) \neq P_t(X|Y)$

Key Takeaways

During the process of transfer learning, the following three important questions must be answered:

- **What to transfer:** identify which portion of knowledge is source-specific and what is common between the source and the target.
- **When to transfer:** aim at utilizing transfer learning to improve target task performance/results and not degrade them. We need to be careful about when to transfer and when not to.
- **How to transfer:** changes to existing algorithms and different techniques, which we will cover in later sections of this article.

Key Takeaways

During the process of transfer learning, the following three important questions must be answered:

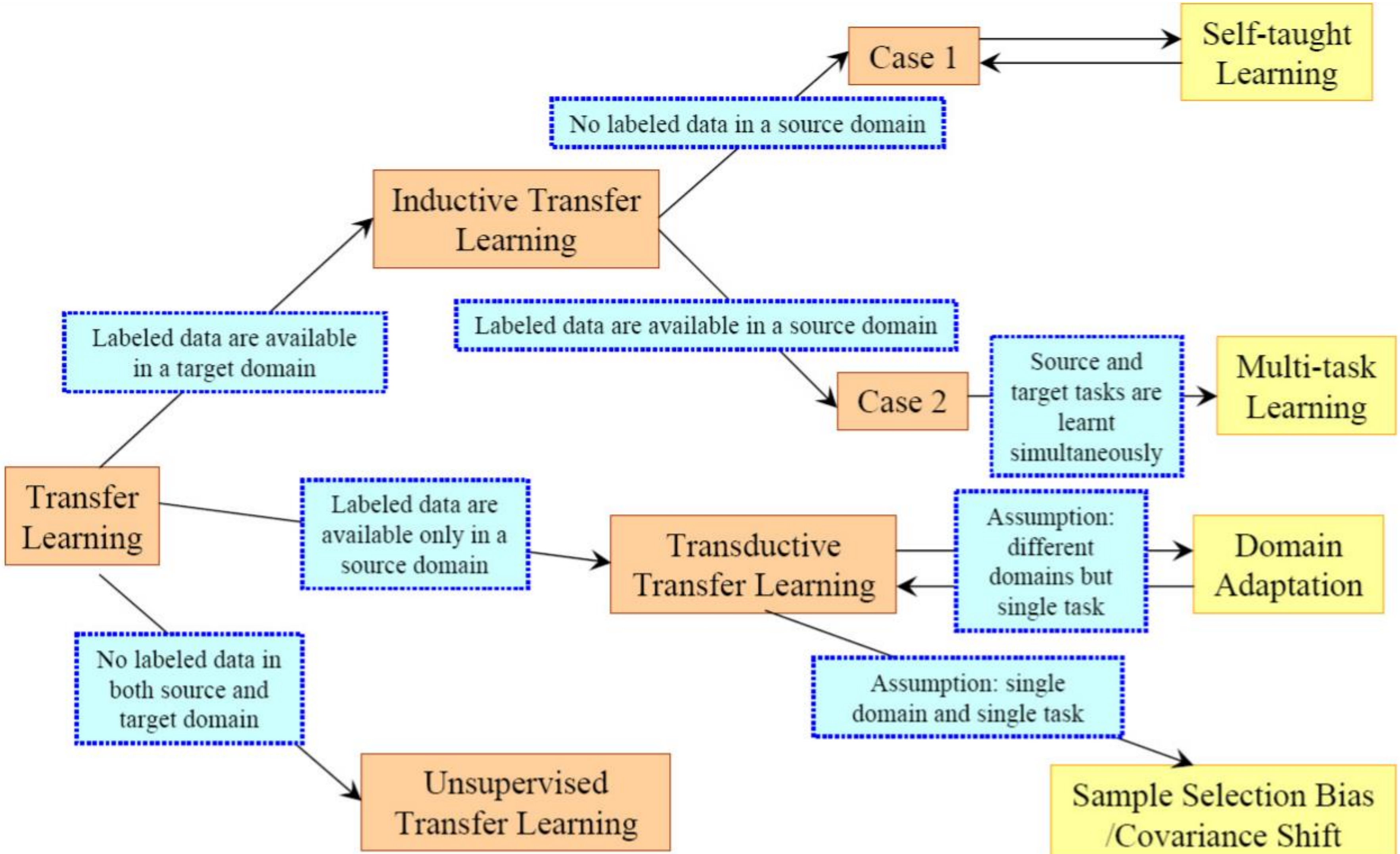
- **What to transfer:** identify which portion of knowledge is source-specific and what is common between the source and the target.
- **When to transfer:** we need to be careful about when to transfer and when not to. aim at utilizing transfer learning to improve target task performance/results and not degrade them (negative transfer).
- **How to transfer: I**dentify ways of transferring the knowledge across domains/tasks (more later).

Transfer Learning Strategies

There are different transfer learning strategies and techniques, which can be applied **based on the domain, task** at hand, **and the availability of data:**

- **Inductive Transfer learning**: the source and target have **same domains**, yet the they have **different tasks** (e.g. documents written in the same language, but unbalanced labels). The algorithms utilize the inductive biases of the source domain to help improve the target task.
- **Unsupervised Transfer Learning:** the source and target have **same domains**, with a focus on **unsupervised tasks in the target** domain. The source and target domains are similar, but the tasks are different. In this scenario, labeled data is unavailable in either of the domains.
- **Transductive Transfer Learning:** In this scenario, there are similarities between the source and target tasks, but the corresponding domains are different. In this setting, the source domain has a lot of labeled data, while the target domain has none.

Transfer Learning Strategies



Pan and Yang, A Survey on Transfer Learning

Transfer Learning Strategies

The approaches for Transfer Learning can be defined in few categories:

- **Instance transfer:** Reusing knowledge from the source domain to the target task (ideal scenario). In most cases, the source domain data cannot be reused directly.
- **Feature-representation transfer:** This approach aims to minimize domain divergence and reduce error rates by identifying good feature representations that can be utilized from the source to target domains.
- **Parameter transfer:** This approach works on the assumption that the models for related tasks share some parameters or prior distribution of hyperparameters.
- **Relational-knowledge transfer** attempts to handle non-IID data, such as data that is not independent and identically distributed.

Transfer Learning for Deep Learning

What people thinks

- you can't do deep learning unless you have a million labeled examples.

What people can do, instead

- You can learn representations from unlabeled data
- You can train on a nearby objective for which is easy to generate labels (imageNet).
- You can transfer learned representations from a relate task.

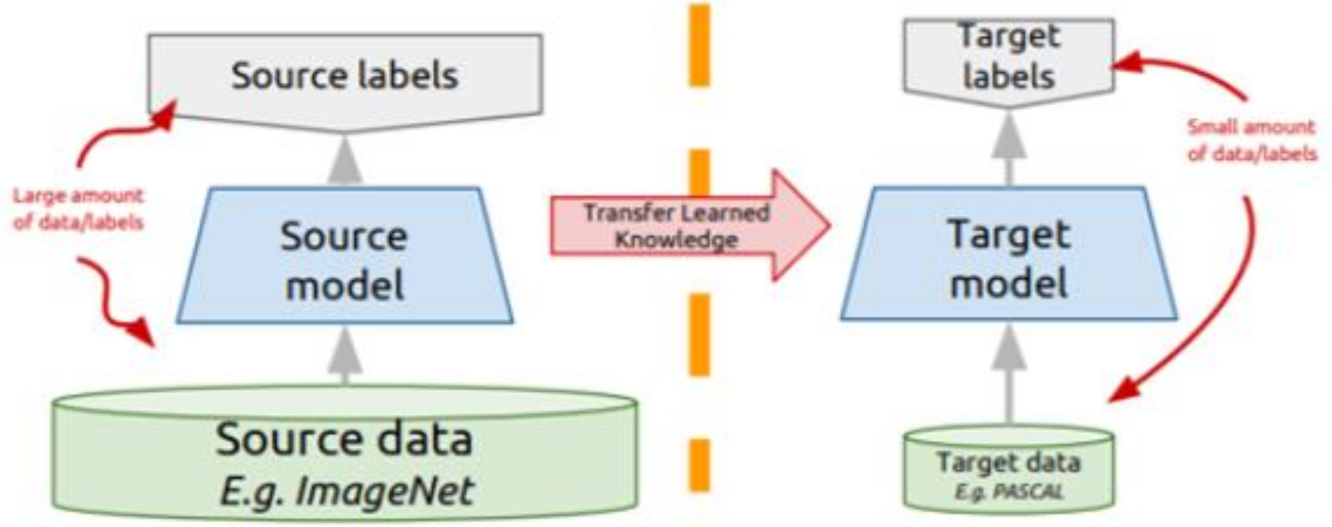
Transfer Learning for Deep Learning

Instead of training a network from scratch:

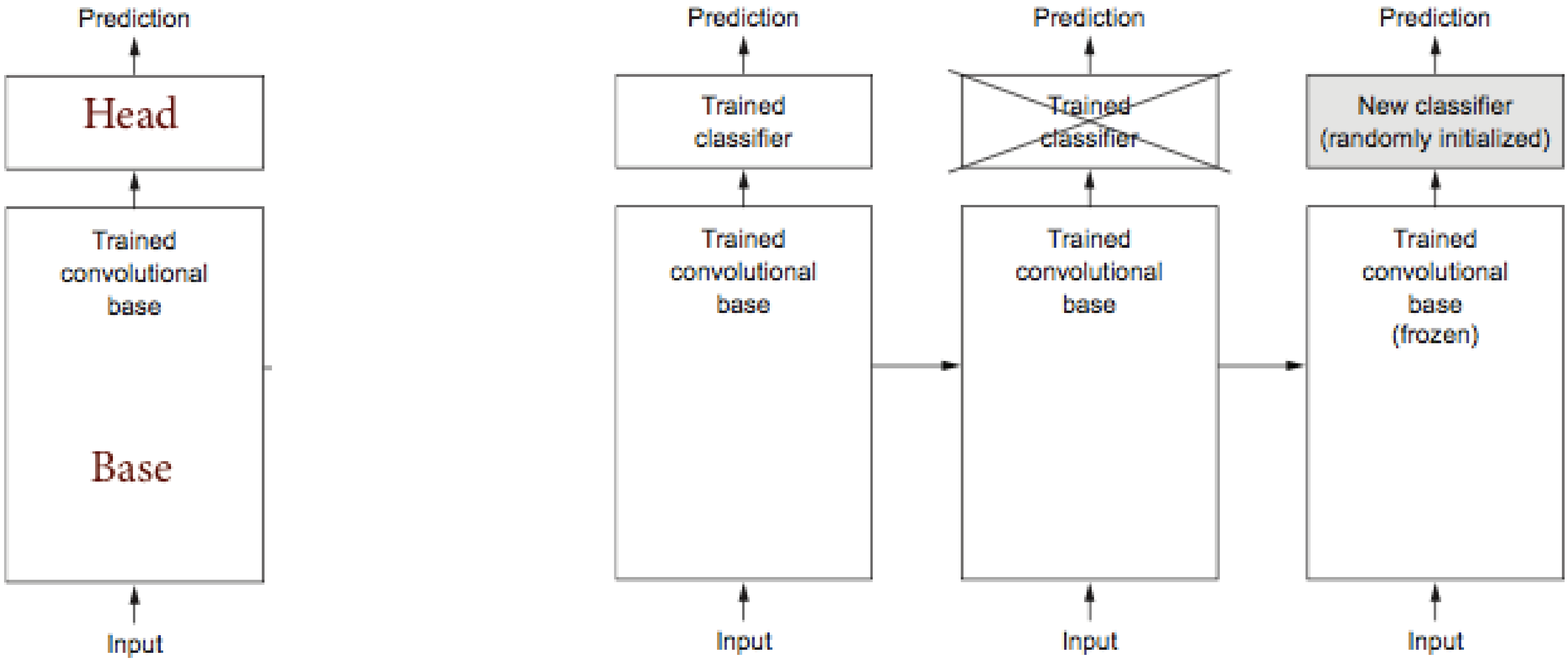
- Take a network trained on a different domain for a different **source task**
- Adapt it for your domain and your **target task**

Variations

- Same domain, different task.
- Different domain, same task.



Representation Extraction



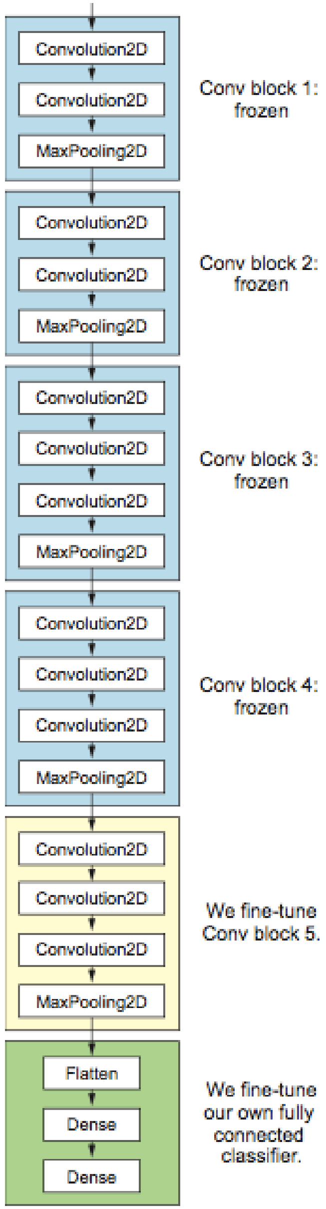
Representation Extraction

Use representations learned by big net to extract features from new samples, which are then fed to a new classifier:

• keep (frozen) convolutional **base** from big model
• generally throw away **head** FC layers since these have no notion of space, and convolutional base is more generic
• since there are both dogs and cats in ImageNet you could get **away** with using the head FC layers as well
• but by throwing it away you can learn more from other dog/cat images

Fine-tuning

• up to now we have frozen the entire convolutional base.
• remember that earlier layers learn highly generic feature maps (edges, colors, textures).
• later layers learn abstract concepts (dog's ear).
• to particularize the model to our task, its often worth tuning the later layers as well.
• but we must be very careful not to have big gradient updates.



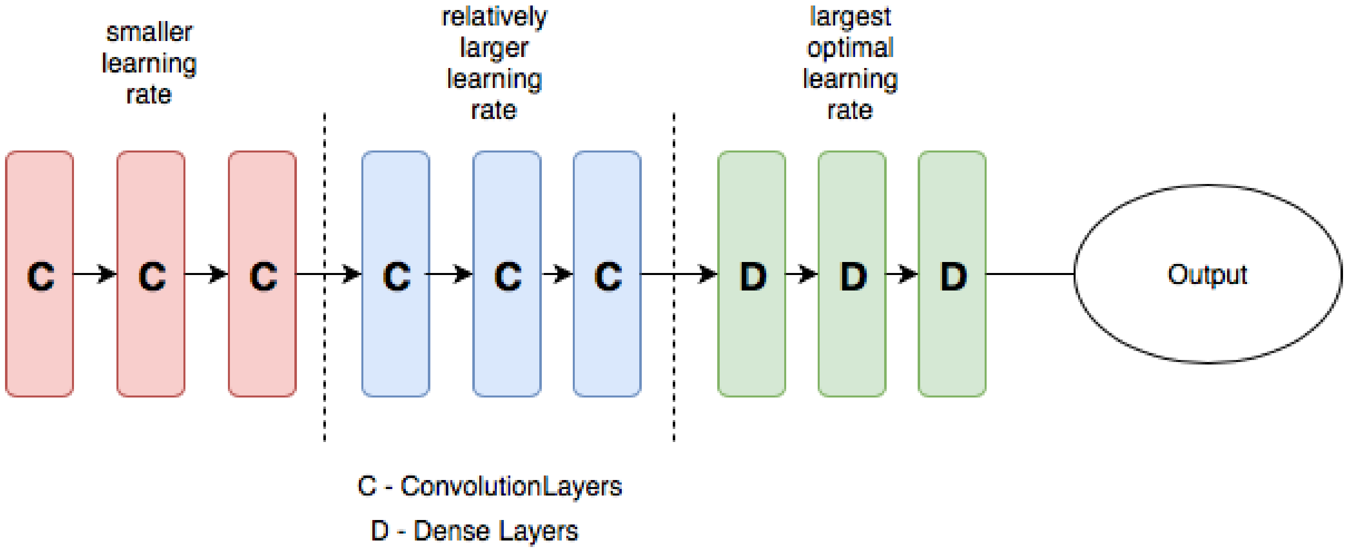
Procedure for Fine-tuning

1. freeze the convolutional base
2. first train the fully connected head you added, keeping the convolutional base fixed. This will get their parameters away from random and in a
3. regime of smaller gradients
4. unfreeze some "later" layers in the base net and
5. now train the base net and FC net together

Since you are now in a better part of the loss surface already, gradients won't be terribly high, but we still need to be careful. Thus use a **very low learning rate**.

Transfer Learning for Deep Learning: Differential Learning Rates

- A low learning rate can take a lot of time to train on the "later" layers. Since we trained the FC head earlier, we could probably retrain them at a higher learning rate.
- General Idea: Train different layers at different rates.
- Each "earlier" layer or layer group (the color-coded layers in the image) can be trained at 3x-10x smaller learning rate than the next "later" one.
- One could even train the entire network again this way until we overfit and then step back some epochs.



SOTA Deep Models: Pre-trained Models

Let us have a quick look at some of the best performing and popular state of the art deep image classification architecture:

AlexNet: credited for opening the floodgates. Designed by Geoffrey Hinton, this network reduced the top-five error rate to 15.3%. Was also one of the first using GPUs to speed up computing.
VGGs (16-19): network from Oxford's Visual Geometry is one of the best performing architectures, widely used to for benchmarking other designs. VGG-16 uses simple 3x3 convolutional layers stacked one on the other (16 channels), followed by one maxpooling.
Inception (AKA Google-Net): introduced for ImageNet Large Scale Visual Recognition Challenge (ILSVCR), it was one of the first to achieve near human performances (top-five error rate of 6.67%). The innovation was to concatenate different kernel size at the same level.

SOTA Deep Models: Pre-trained Models

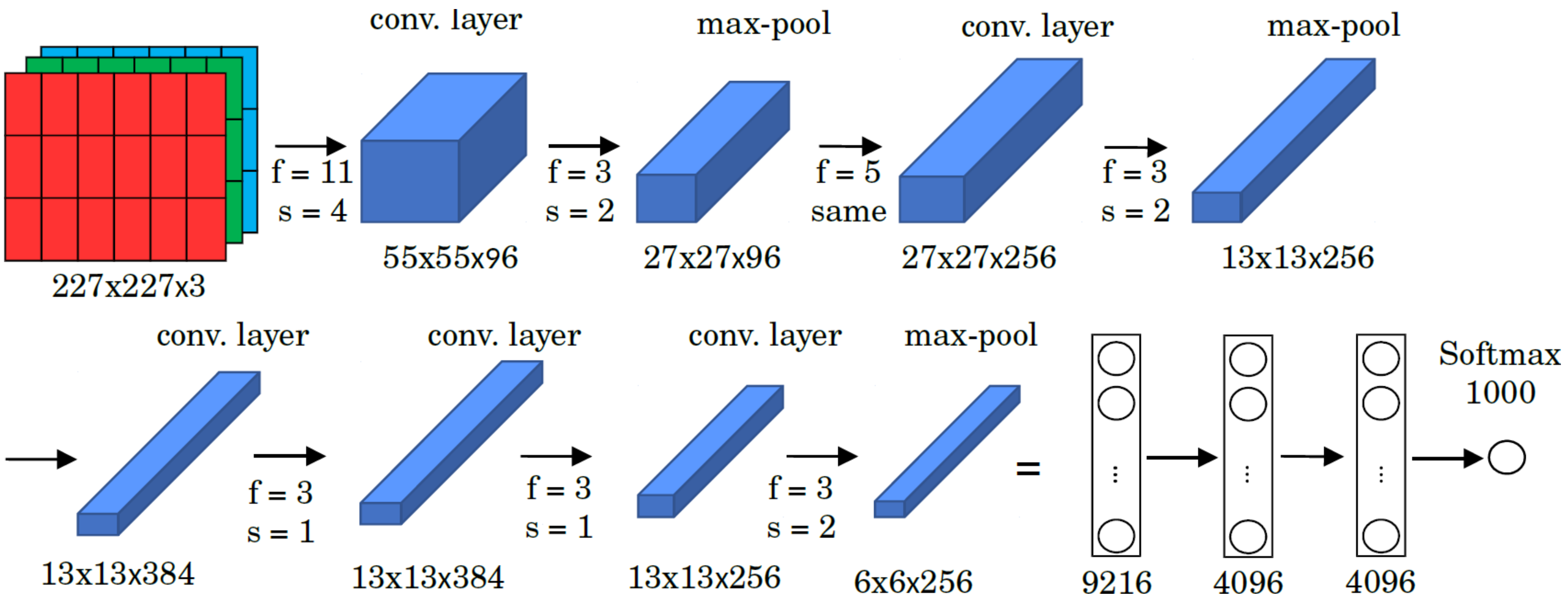
ResNets: introduced by Microsoft Research Asia, the residual network (ResNet) was a novel architecture using batch normalization and skipping connections (top-five error rate of 3.57%). With its 152 layers, It is way deeper than VGG.

MobileNet: designed to be suitable for mobile and embedded system. This network utilizes a novel idea of using depth-wise separable convolutions to reduce the overall number of parameters required to train the network.

DenseNets:

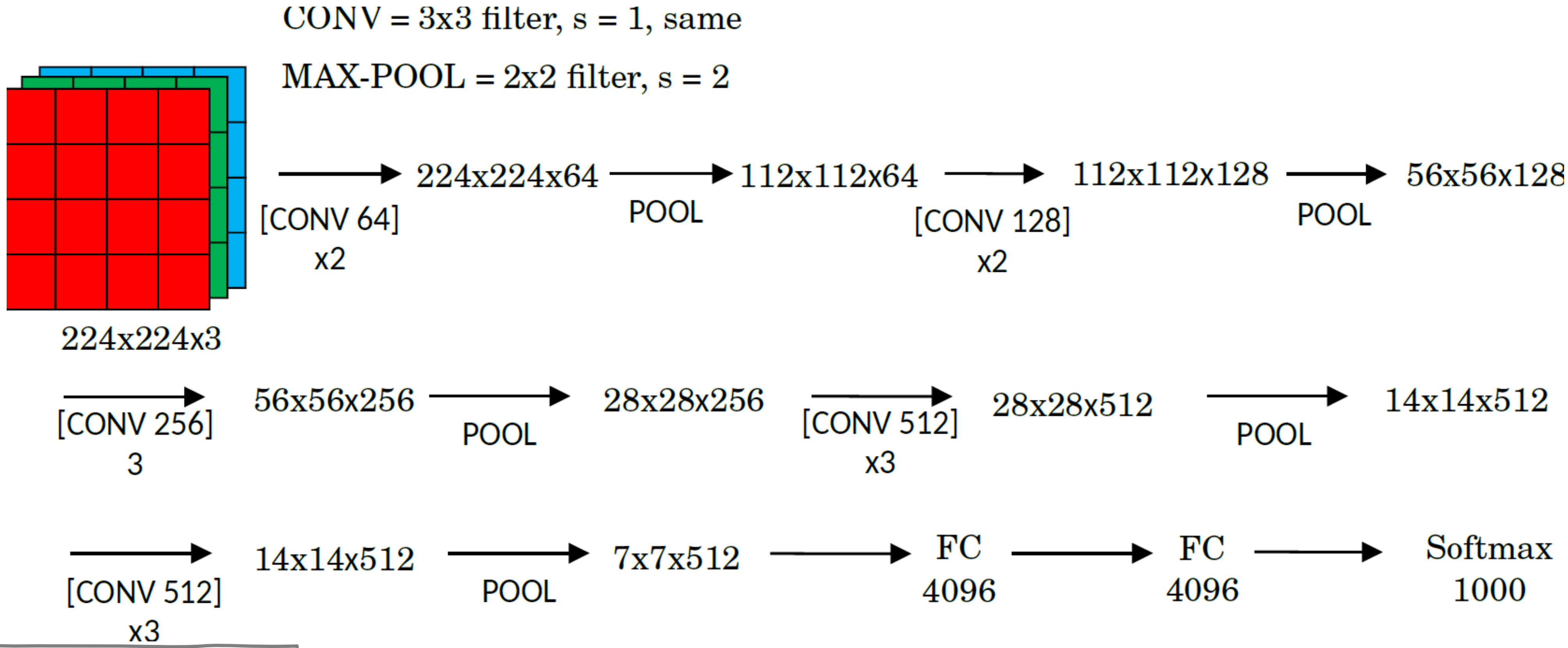
SOTA Deep Models: AlexNet

- 1.2 million high-resolution (227x227x3) images in the ImageNet 2010 contest;
- 1000 different classes, NN with 60 million parameters to optimize (~ 255 MB);
- Uses ReLu activation functions; GPUs for training, 12 layers.



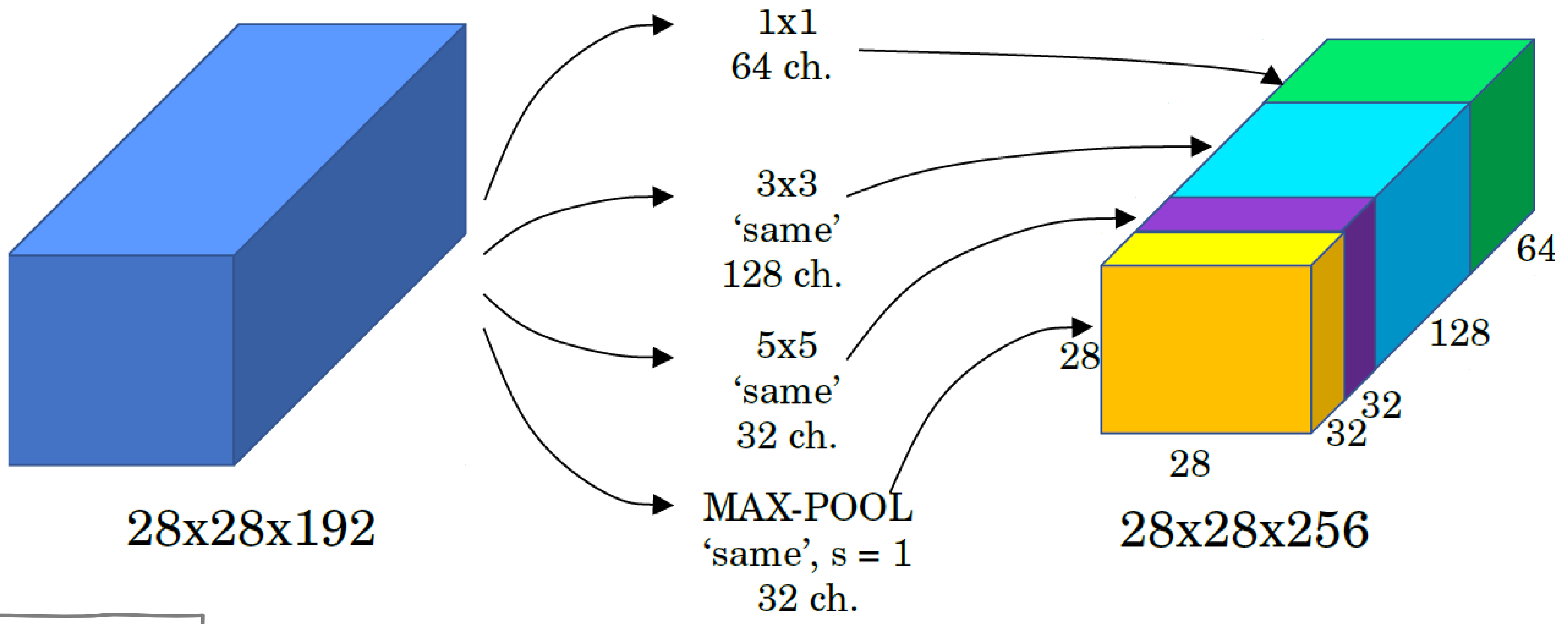
SOTA Deep Models: VGG 16-19

- ImageNet Challenge 2014; 16 or 19 layers; 138 million parameters (~ 522 MB).
- Convolutional layers use ‘same’ padding and stride s=1.
- Max-pooling layers use a filter size f=2 and strie s=2.



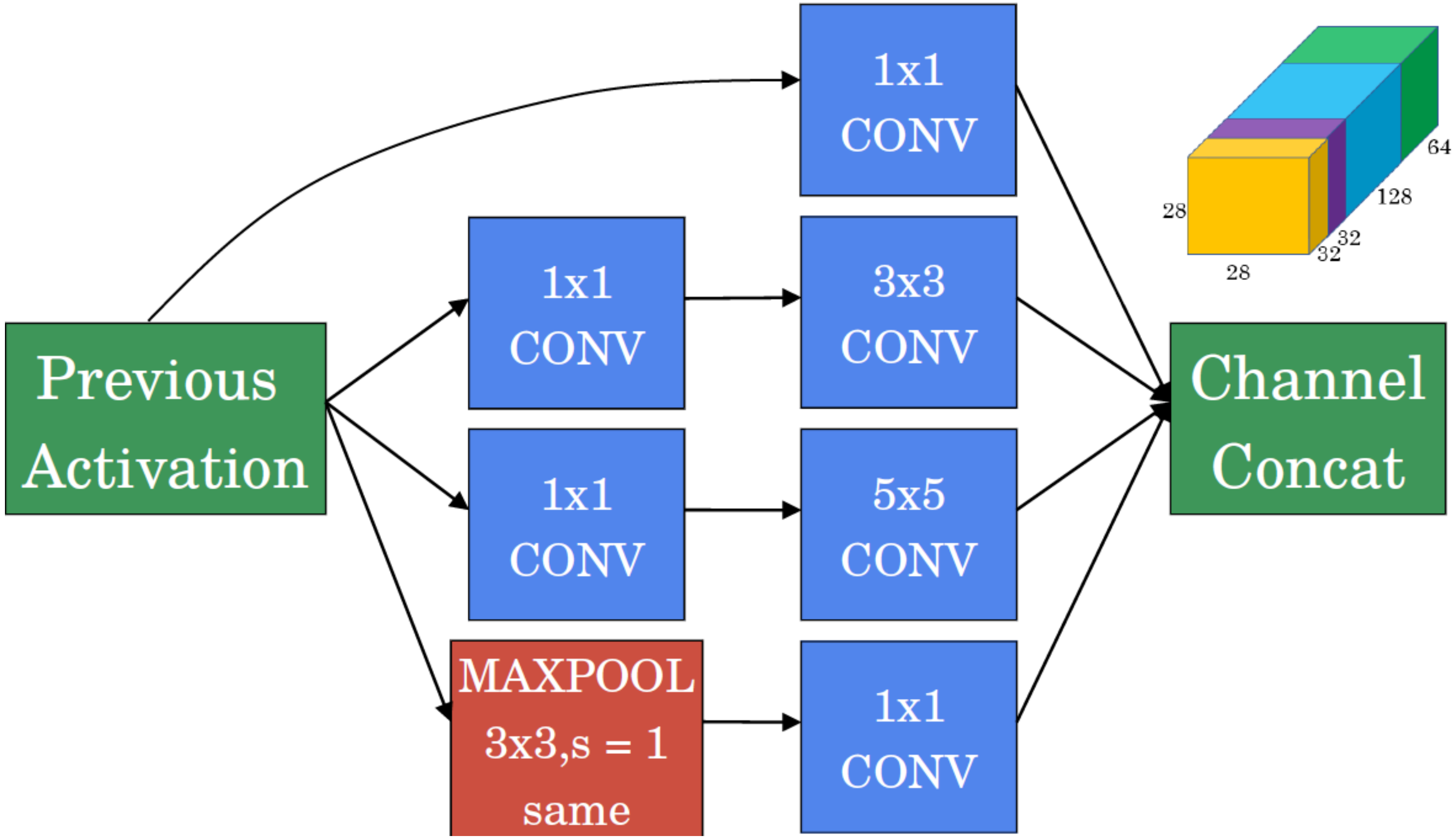
SOTA Deep Models: Inception (GoogLeNet)

- The motivation behind inception networks is to use more than a singe type of convolution layer at each layer.
- Use 1 x 1,3 x 3,5 x convolutional layers, and max-pooling layers in parallel.
- All modules use same convolution.
- Basic implementation:



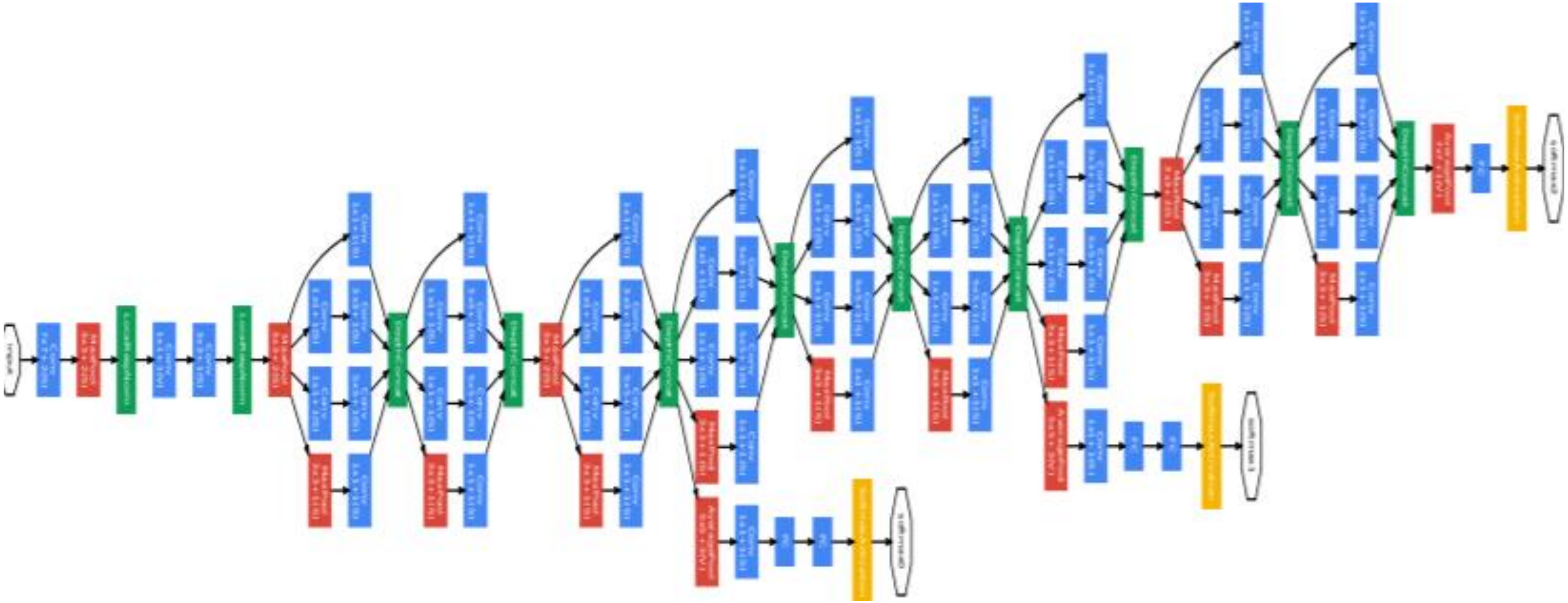
SOTA Deep Models: Inception (GoogLeNet)

- Use 1 x 1 convolutions that reduce the size of the channel dimension.
  - The number of channels can vary from the input to the output..



SOTA Deep Models: Inception (GoogLeNet)

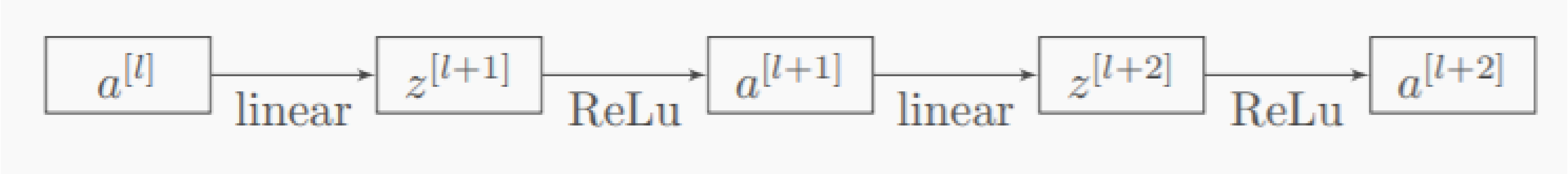
- The inception network is formed by concatenating other inception modules.
- It includes several softmax output units to enforce regularization.



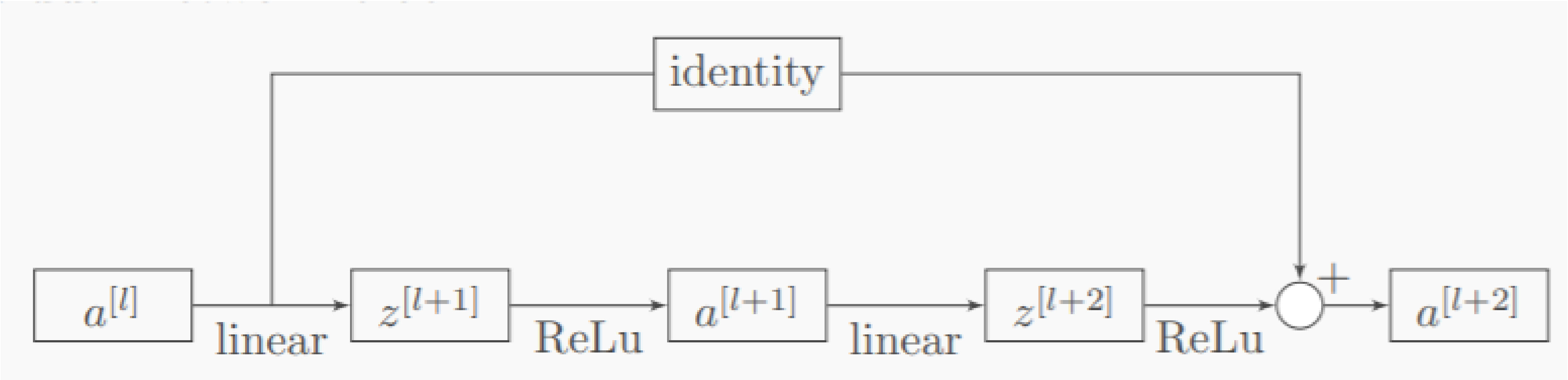

WE NEED TO GO
DEEPER

SOTA Deep Models: ResNets

- Residual nets appeared in 2016 to train very deep NN (100 or more layers)
- Their architecture uses ‘residual blocks’.
- Plain network structure:

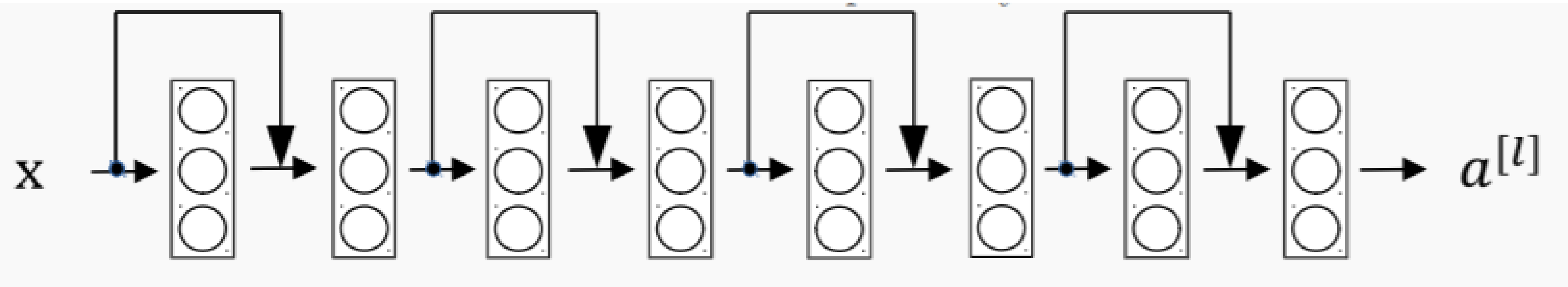


- **Residual network block:**

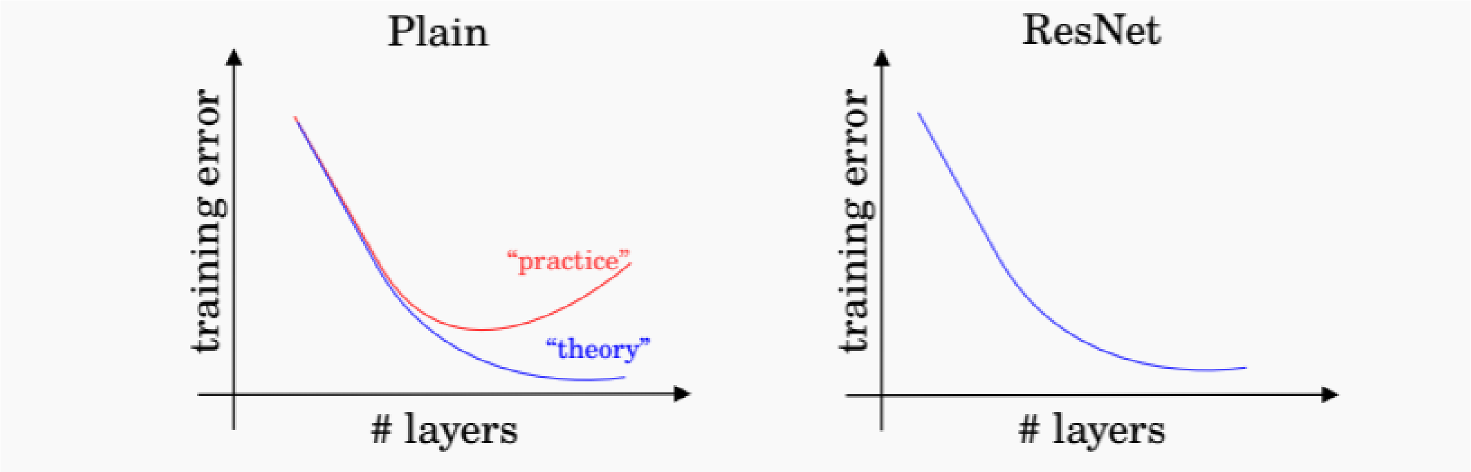


SOTA Deep Models: ResNets

- A residual network stacks residual blocks sequentially.



- The idea is to allow the network to become deeper without increasing the training complexity.

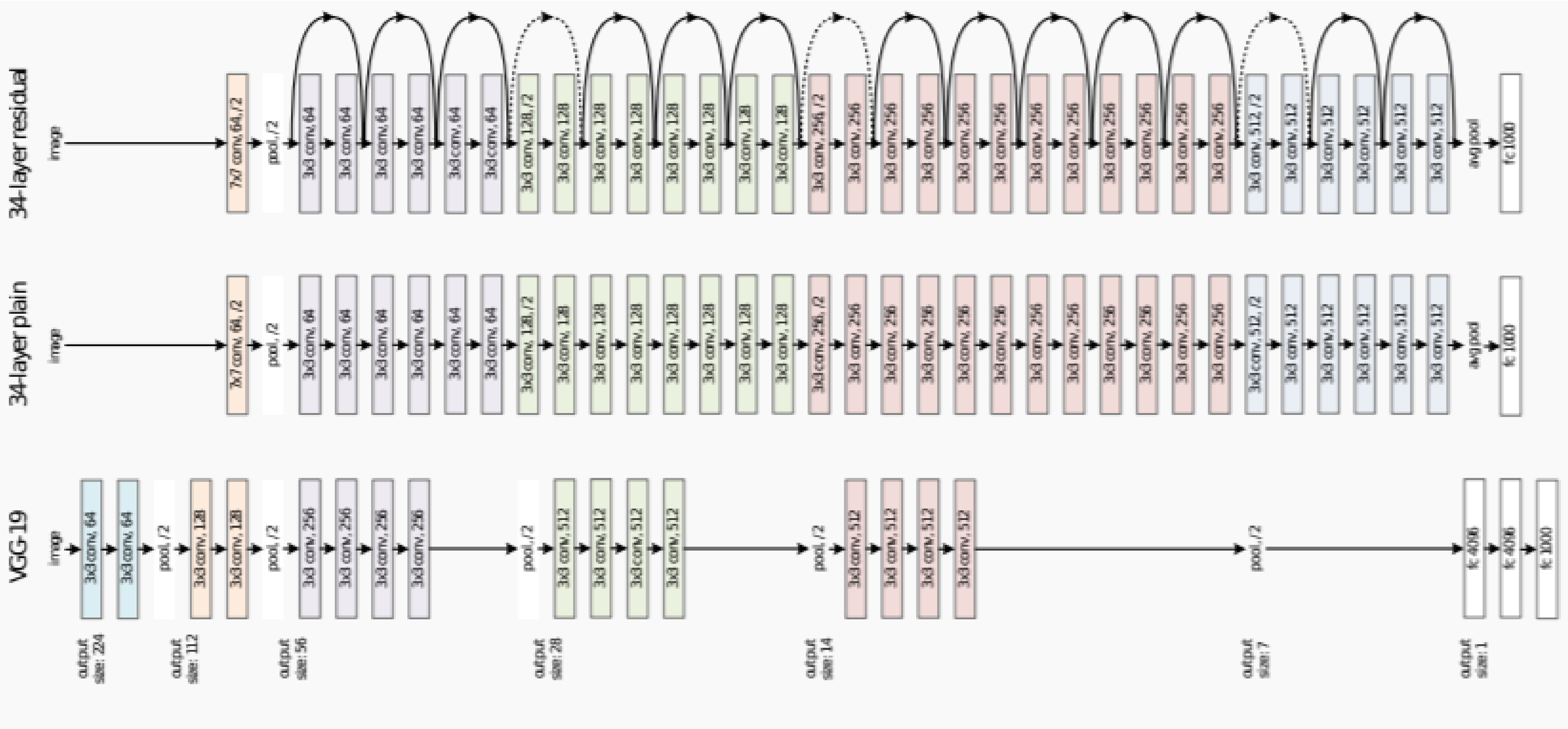


SOTA Deep Models: ResNets

- Residual nets implement blocks with convolutional layers that use 'same' padding option (even when max-pooling).
  - This allows the block to learn the identity function.
- The designer may want to reduce the size of features and use 'valid' padding.
  - In such case, the shortcut path can implement a new set of convolutional layers that reduces the size appropriately.

Number of Layers	Number of Parameters
ResNet 18	11.174M
ResNet 34	21.282M
ResNet 50	23.521M
ResNet 101	42.513M
ResNet 152	58.157M

SOTA Deep Models: ResNets



Source: He2016

SOTA Deep Models: MobileNet, a lightweight model

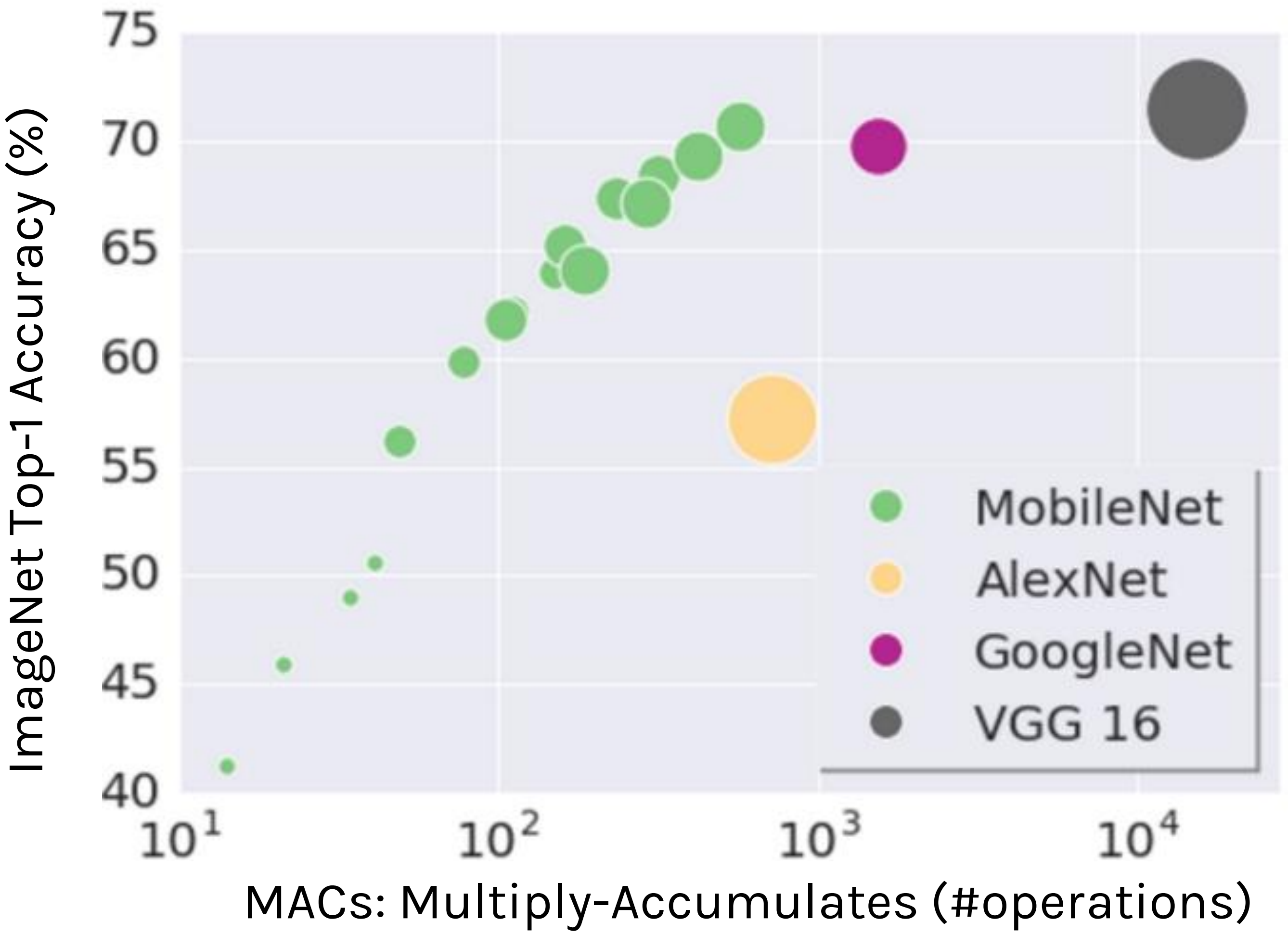


Table 1. MobileNet Body Architecture

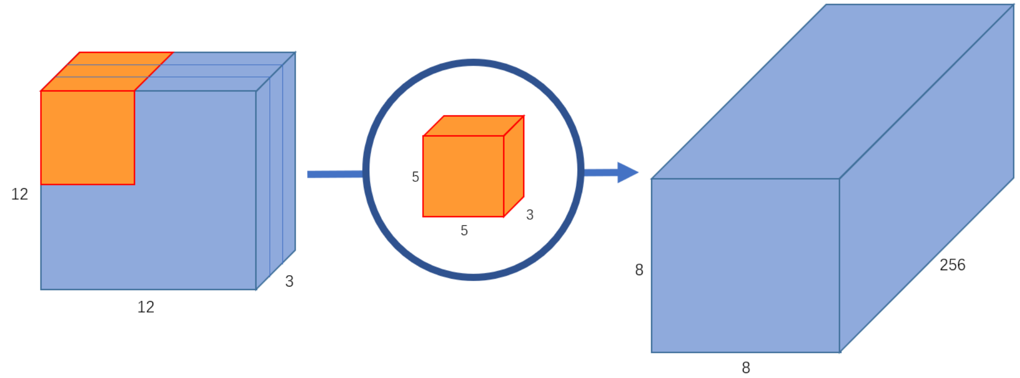
Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
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 Conv / s1	$3 \times 3 \times 512$ dw $1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$ $14 \times 14 \times 512$
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	$1 \times 1 \times 1000$

MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications (arXiv.1704.04861)

SOTA Deep Models: MobileNet <cont>

Standard Convolution

Filters and combines inputs into a new set of outputs in one step



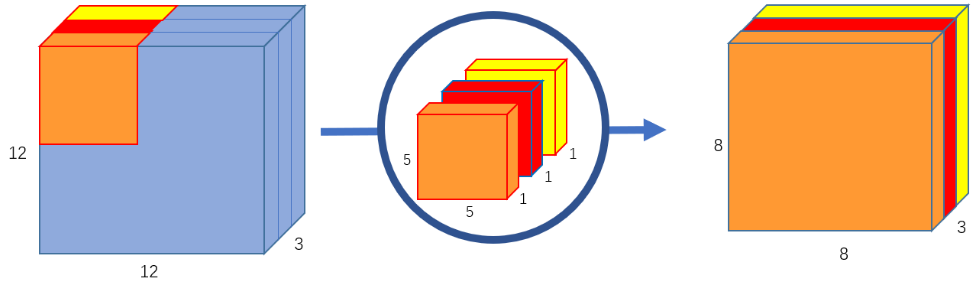
Input: 12x12x3
Filter: 5x5x3x256

Output: 8x8x256
(no padding)

MACs: (5x5)x3x256x(12x12) ~ **2.8M**
Parameters: (5x5x3)x256 + 256 ~ **20K**

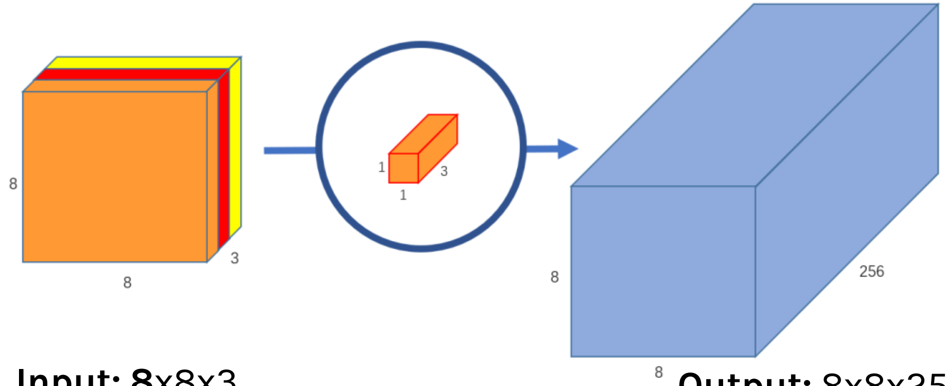
Depth-Wise Separable Convolution (DW)

It combines a depth wise convolution and a pointwise convolution



Input: 12x12x3
Filter: 5x5x3

Output: 8x8x3
(no padding)

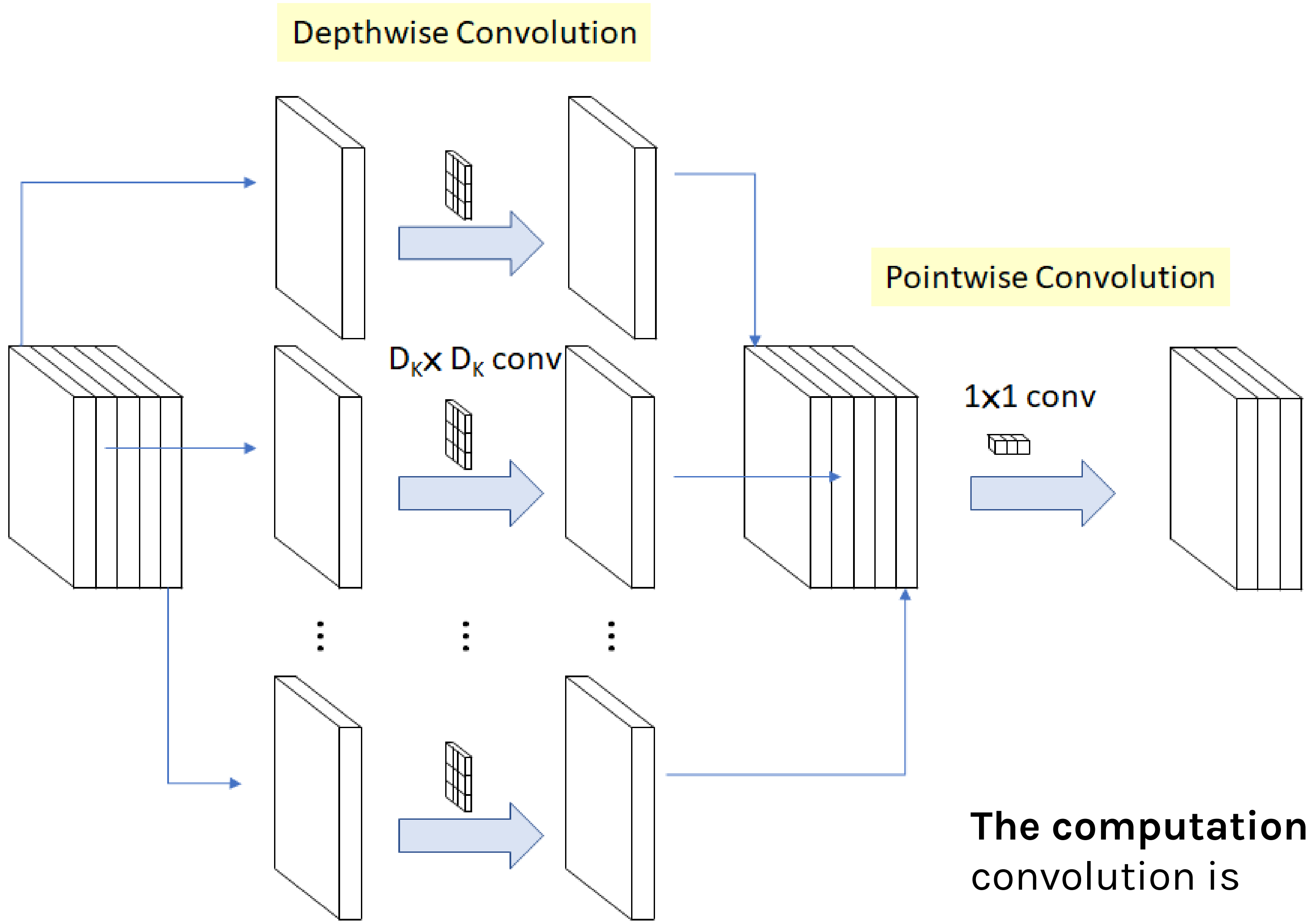


Input: 8x8x3
Filter: 1x1x3x256

Output: 8x8x256
(no padding)

MACs: (5x5)x3x(12x12) + 3x256x(8x8) ~ **60K**
Parameters: (5x5x3 + 3) + (1x1x3x256+256) ~ **1K**

SOTA Deep Models: MobileNet <cont>



Computation Reduction

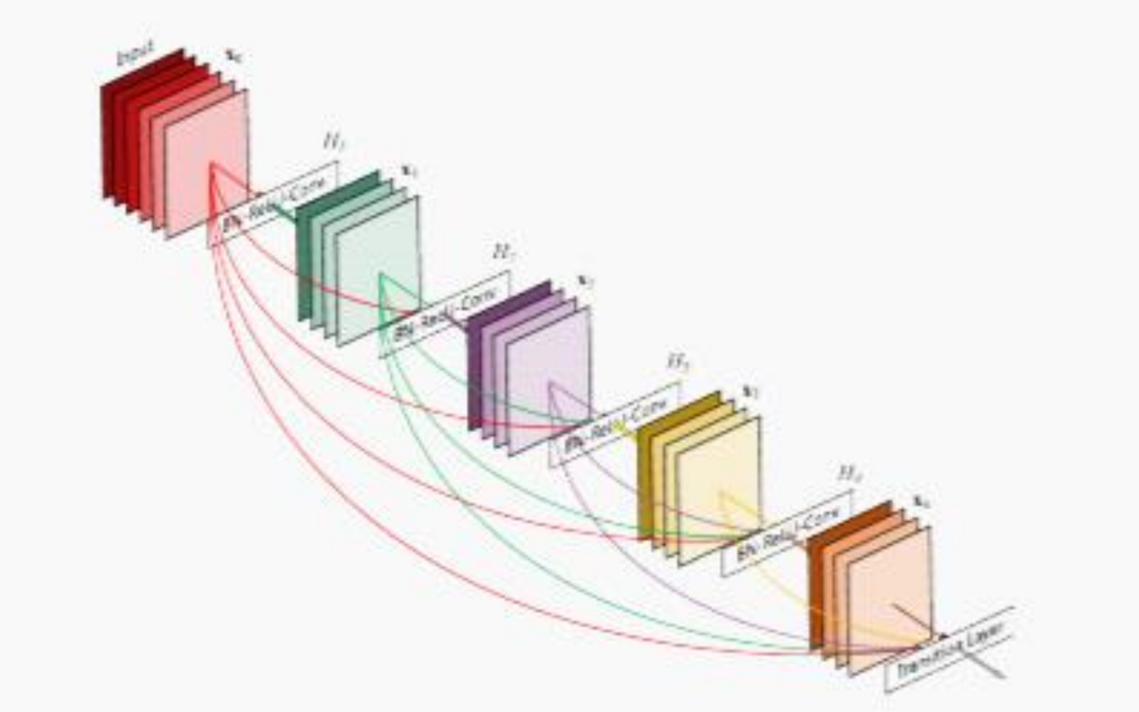
M input channels
N output channels.
DK the filter (kernel) size
DF the feature map size

$$\frac{1}{N} + \frac{1}{D_K^2}$$

The computation Reduction comparing to standard convolution is

SOTA Deep Models: DenseNets

- **Goal:** allow maximum information (and gradient) flow >> connect every layer directly with each other.
- DenseNets exploit the potential of the network through feature reuse >> no need to learn redundant feature maps.
- DenseNets layers are very narrow (e.g. 12 filters), and they just add a small set of new feature-maps.



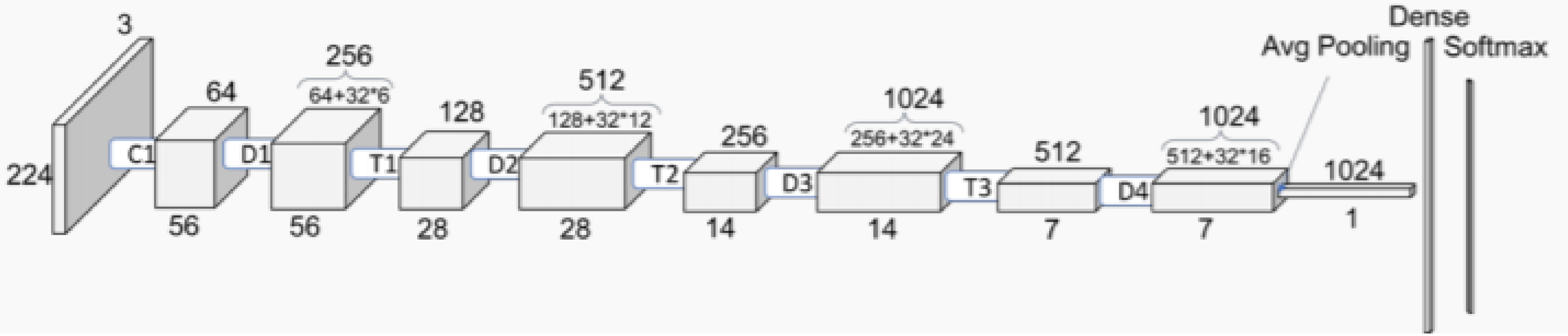
SOTA Deep Models: DenseNets

- DenseNets do not sum the output feature maps of the layer with the incoming feature maps but concatenate them:

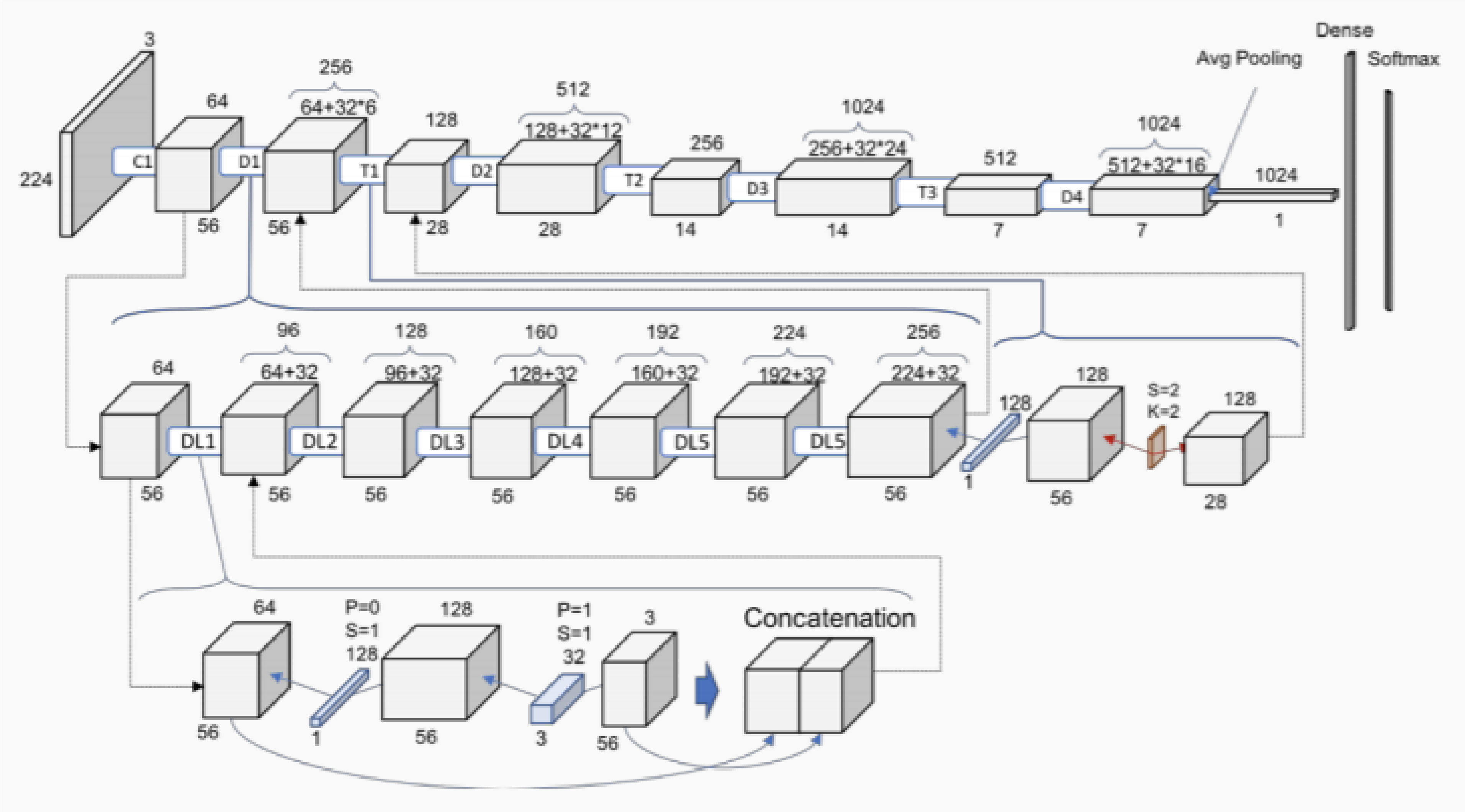
$$a^{[l]} = g([a^{[0]}, a^{[1]}, \ldots, a^{[l-1]}])$$

- Dimensions of the feature maps remains constant within a block, but the number of filters changes between them >> growth rate:

$$k^{[l]} = k^{[0]} + k \cdot (l-1)$$



SOTA Deep Models: DenseNets



SOTA Deep Models: Summary Networks

- We are now reaching top-5 error rates lower than human manual classification.

Year	CNN	Developed by	Place	Top-5 error rate	No. of parameters
1998	LeNet(8)	Yann LeCun et al			60 thousand
2012	AlexNet(7)	Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever	1st	15.3%	60 million
2013	ZFNet()	Matthew Zeiler and Rob Fergus	1st	14.8%	
2014	GoogLeNet(19)	Google	1st	6.67%	4 million
2014	VGG Net(16)	Simonyan, Zisserman	2nd	7.3%	138 million
2015	ResNet(152)	Kaiming He	1st	3.6%	

THANK YOU