Computer Vision and Deep Learning

Lecture 7

Last time

- CNN architectures
  - Alexnet
  - VGG
  - GoogLe Net
  - ResNet
  - Mobilenets

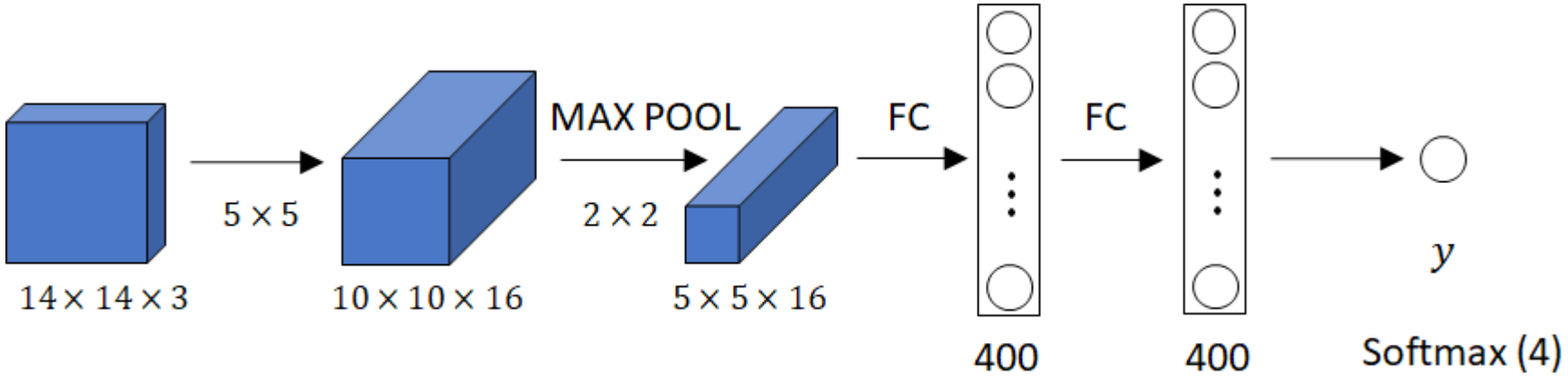
Yann LeCun

April 6, 2015 ·

In Convolutional Nets, there is no such thing as "fully-connected layers". There are only convolution layers with 1x1 convolution kernels and a full connection table.

It's a too-rarely-understood fact that ConvNets don't need to have a fixed-size input. You can train them on inputs that happen to produce a single output vector (with no spatial extent), and then apply them to larger images. Instead of a single output vector, you then get a spatial map of output vectors. Each vector sees input windows at different locations on the input.

In that scenario, the "fully connected layers" really act as 1x1 convolutions.



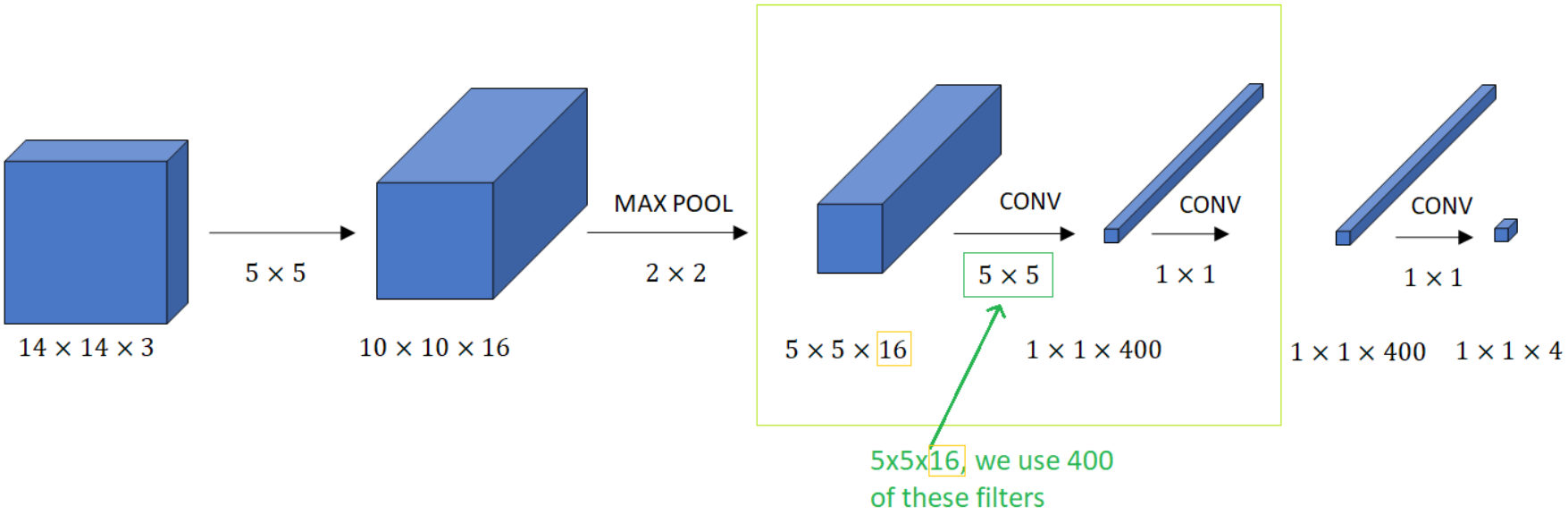
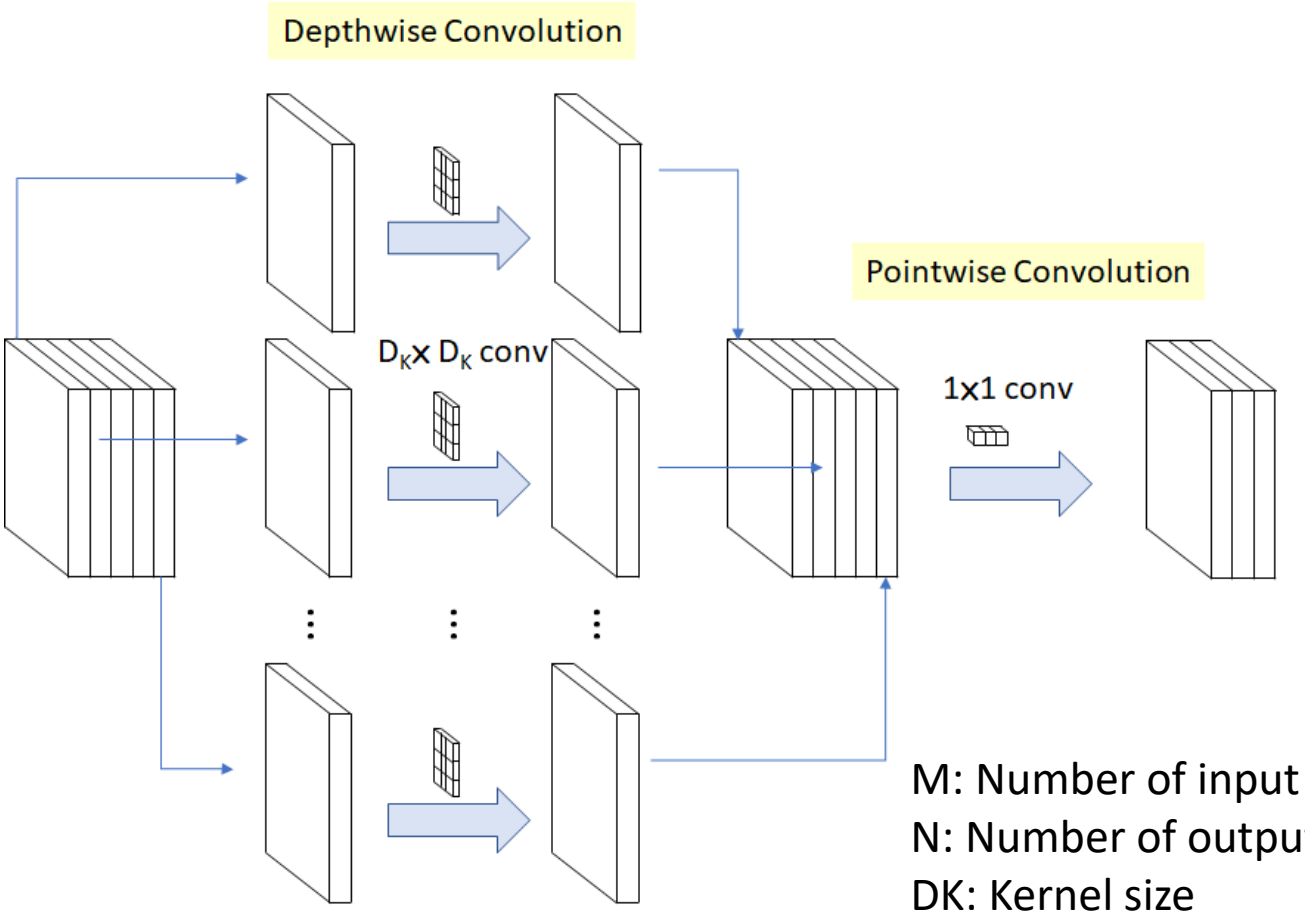


Image credit: Andrew Ng

Depth-wise separable convolutions



M: Number of input channels,
N: Number of output channels,
DK: Kernel size
DF: Feature map size

Operation cost:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F$$

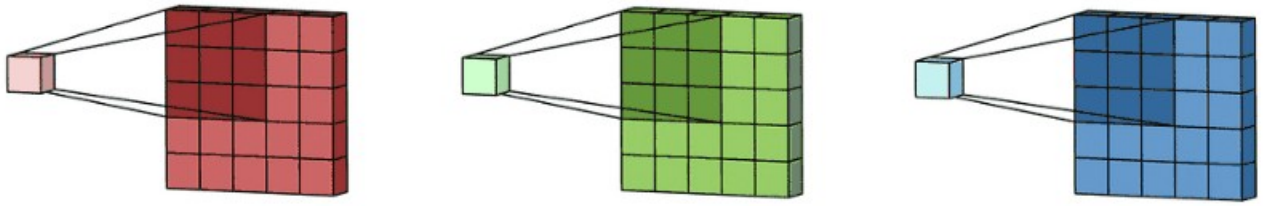
Depthwise convolution

vs.

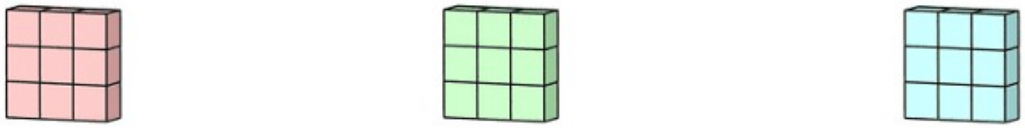
$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$$

Classical convolution

Depth-wise separable convolutions



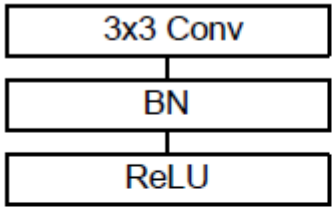
Depth-wise separable convolutions

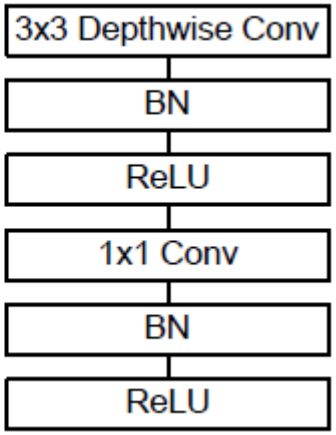


Mobilenets

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	$1 \times 1 \times 1000$

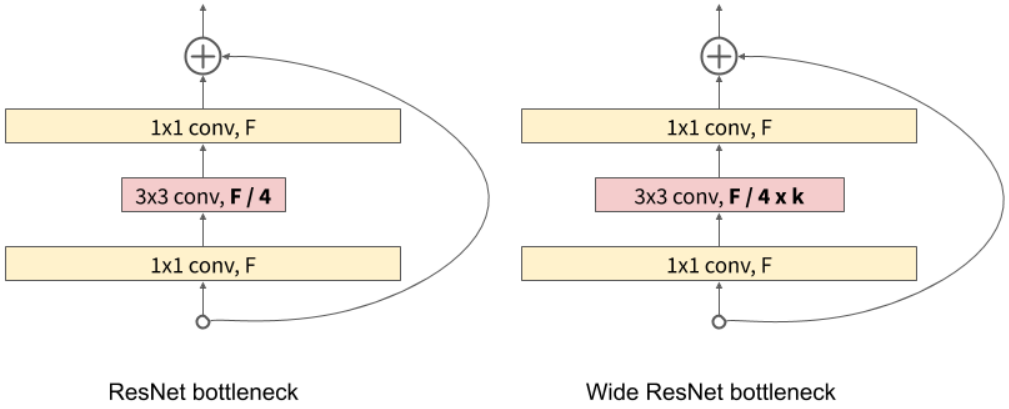




Other architectures

Wide RESNET

- Use wider residual blocks (larger number of filters)
- 50 layer wide resnet is better than 101 layer classical resnet



https://arxiv.org/pdf/1605.07146.pdf

ResNext

- Again increase the width of the networks, but this time through parallel branches in a block

	setting	top-1 err (%)	top-5 err (%)
1× complexity references:			
ResNet-101	1 × 64d	22.0	6.0
ResNeXt-101	32 × 4d	21.2	5.6
2× complexity models follow:			
ResNet-**200** [15]	1 × 64d	21.7	5.8
ResNet-101, wider	1 × **100**d	21.3	5.7
ResNeXt-101	**2** × 64d	20.7	5.5
ResNeXt-101	**64** × 4d	**20.4**	**5.3**

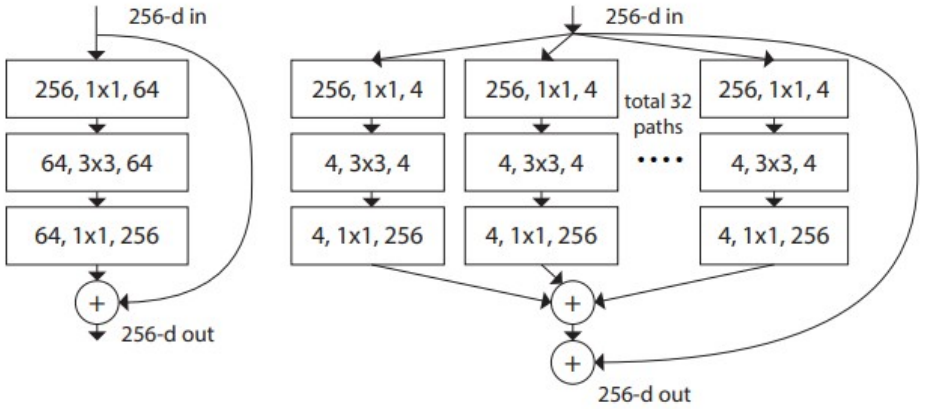
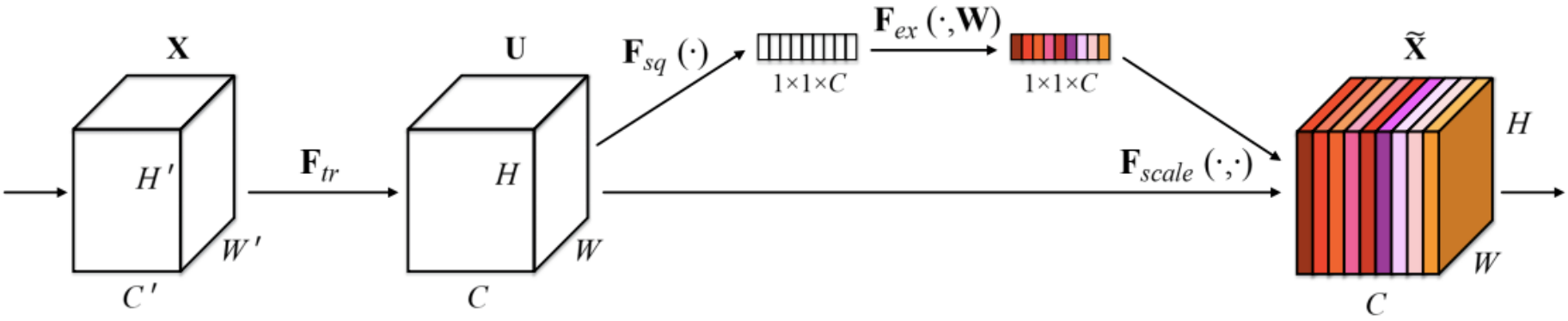


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

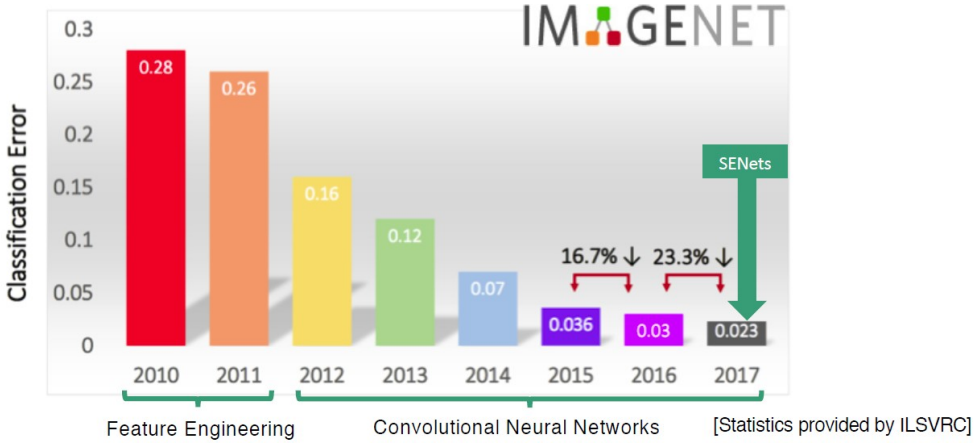
https://arxiv.org/pdf/1611.05431.pdf

SENet



ImageNet 2017 winner

2.3% top-5 accuracy



Compute weight for each feature map:

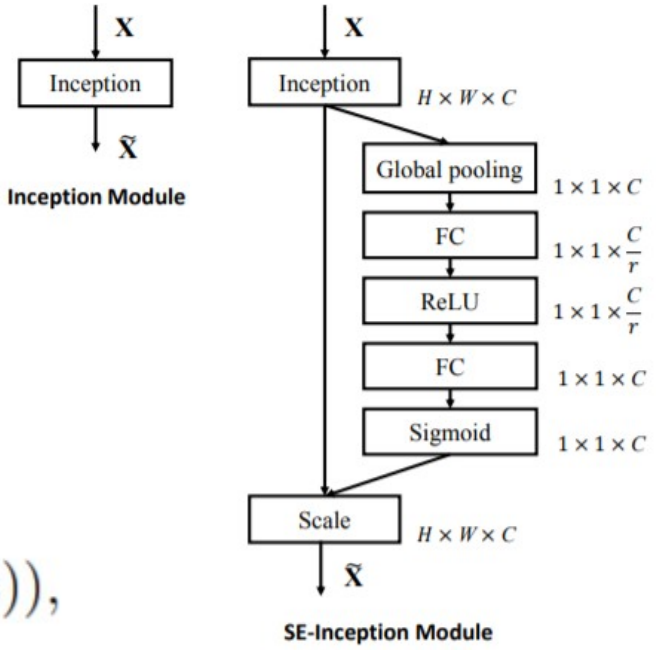
- Global average pooling
- 2FC layers

Squeeze-and-Excitation Networks, 2018: https://arxiv.org/pdf/1709.01507.pdf

SENet

Squeeze (**Global Information Embedding**) squeeze global spatial information into a channel descriptor

$$z_c = \mathbf{F}_{sq}(\mathbf{u}_c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} u_c(i, j).$$



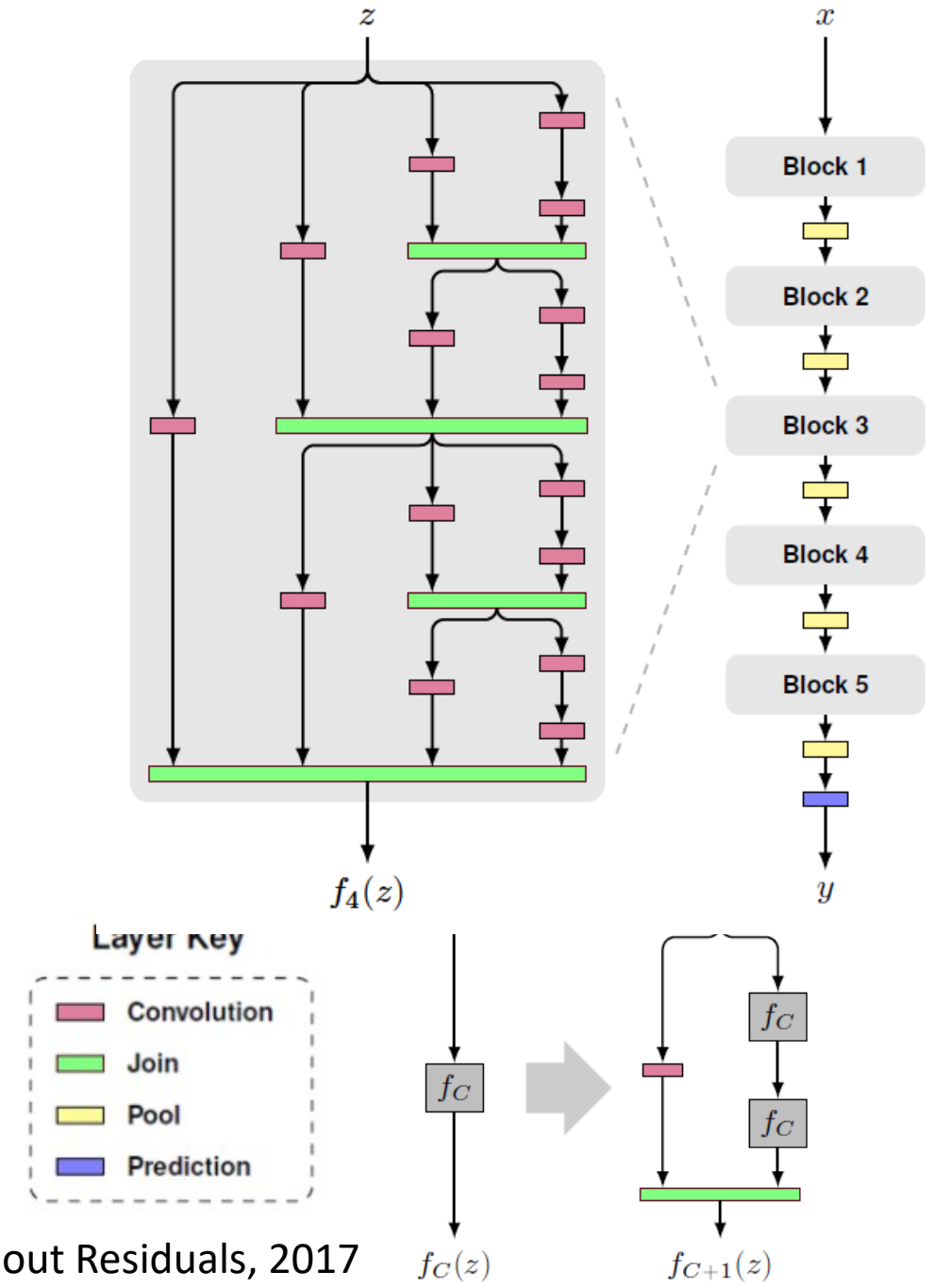
Excitation (**Adaptive Recalibration**)

- capture channel-wise dependencies
- dynamics conditioned on the input

$$\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(g(\mathbf{z}, \mathbf{W})) = \sigma(\mathbf{W}_2 \delta(\mathbf{W}_1 \mathbf{z})),$$

Fractal Nets

- non-residual-network approach
- neural network macro-architecture based on self-similarity
  - both shallow and deep paths to output
- the key may be the ability to transition, during training, from effectively shallow to deep

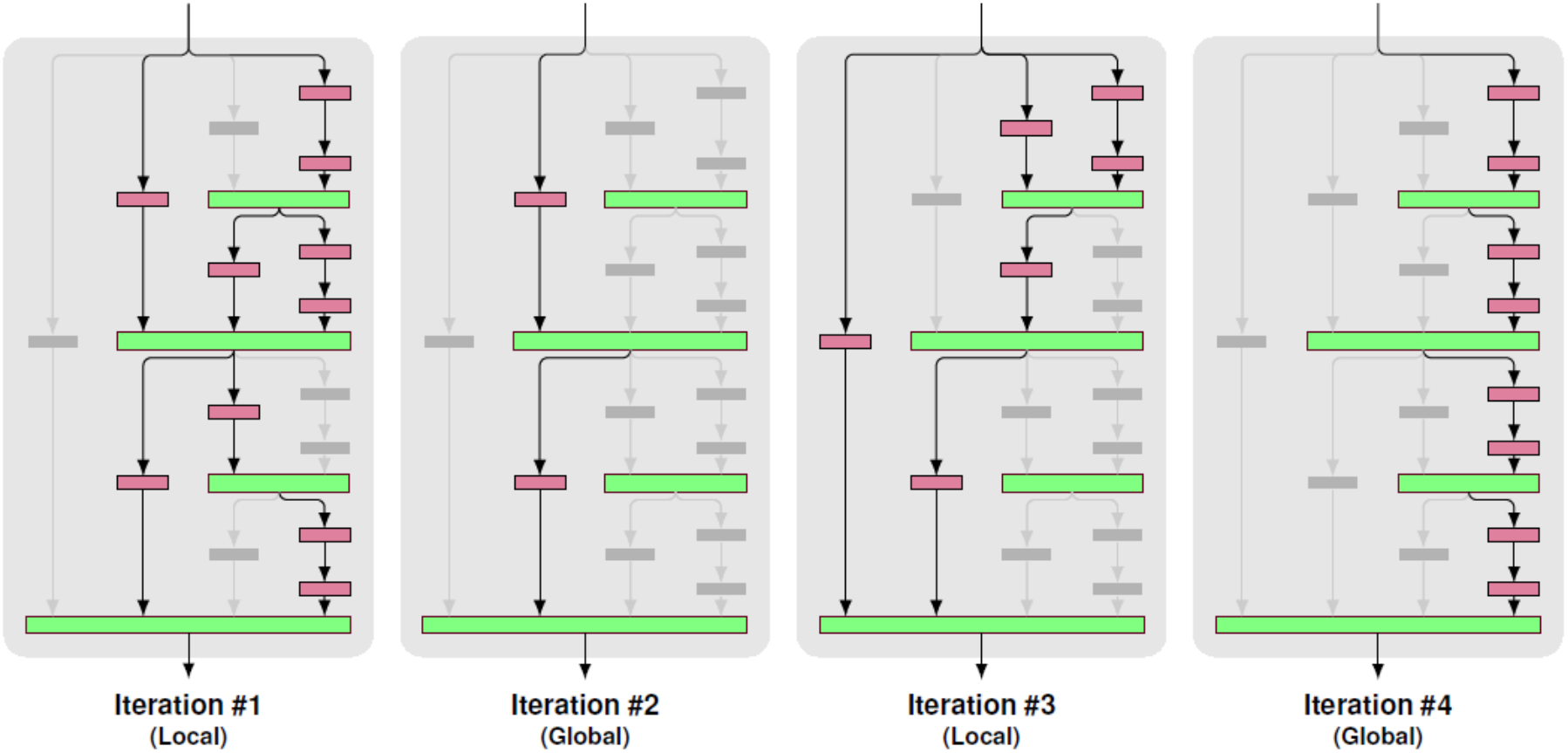


FractalNet: Ultra-Deep Neural Networks without Residuals, 2017

Fractal nets

Drop path - prevents co-adaptation of parallel paths by randomly dropping operands of the join layers

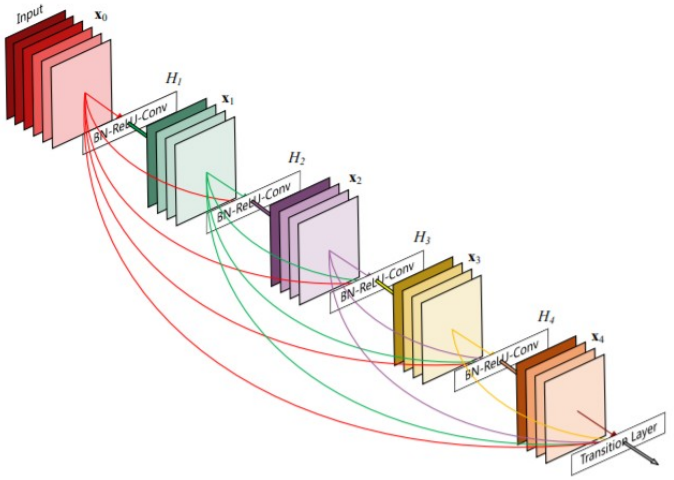
- Global: a single path is selected for the entire network
- Local: drops each input with fixed probability



Densely connected networks

Dense block: each layer is connected to every other layer in feed-forward fashion

- strengthens feature propagation,
- encourages feature reuse
- helps with the vanishing gradients problem



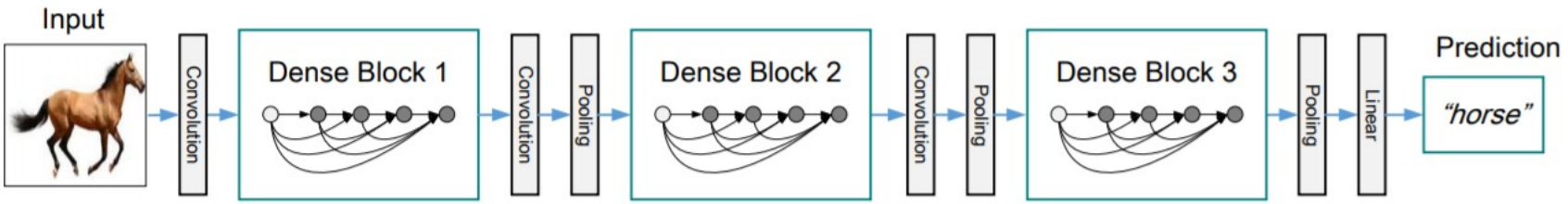
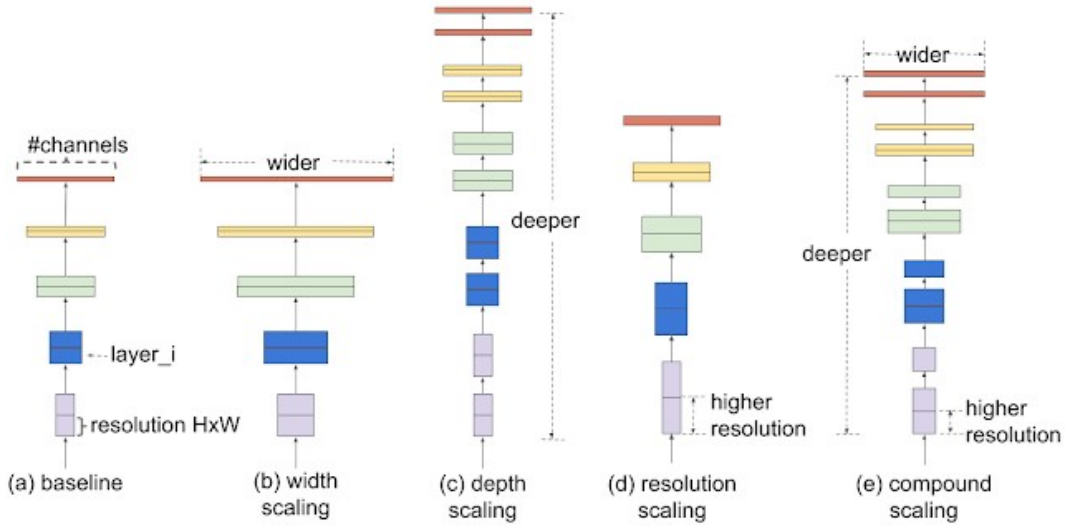
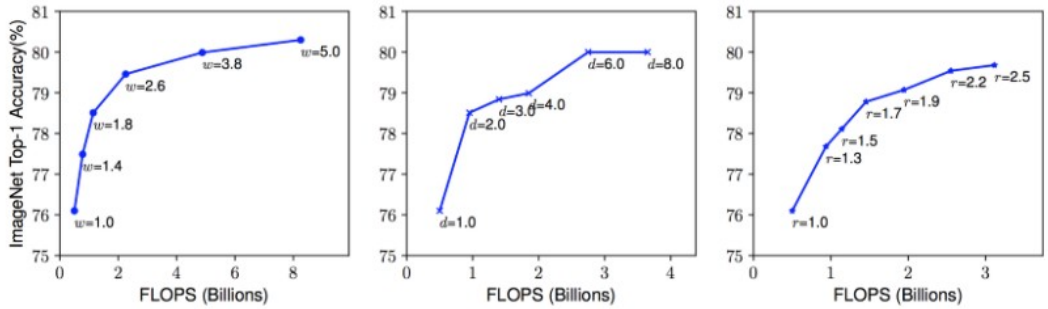


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

Densely Connected Convolutional Networks, 2017

Efficient nets





Scaling Up a Baseline Model with Different Network Width (w), Depth (d), and Resolution (r) Coefficients. Bigger networks with larger width, depth, or resolution tend to achieve higher accuracy, but the accuracy gain quickly saturate after reaching 80%, demonstrating the limitation of single dimension scaling.

Efficient nets

- carefully balancing network depth, width, and resolution can lead to better performance
- design a new baseline network and scale it up to obtain a family of models, called *EfficientNets*

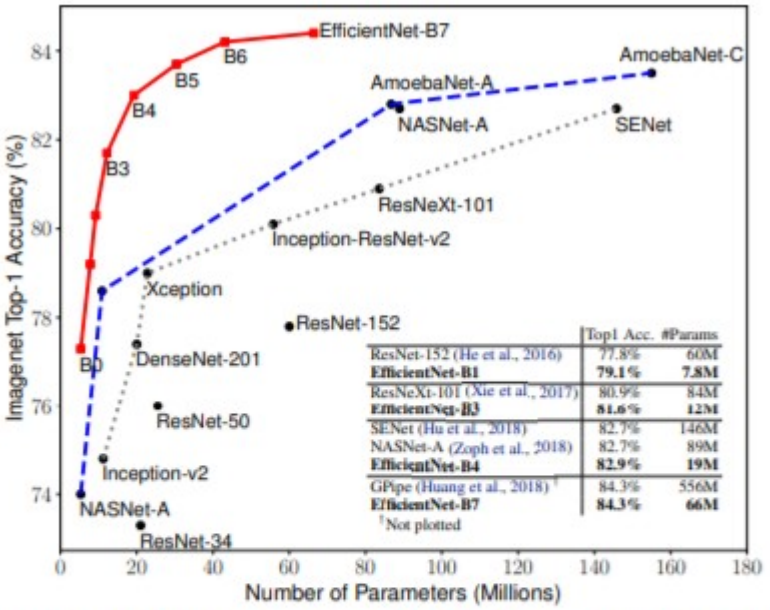


Figure 1. **Model Size vs. ImageNet Accuracy.** All numbers are for single-crop, single-model. Our EfficientNets significantly outperform other ConvNets. In particular, EfficientNet-B7 achieves new state-of-the-art 84.3% top-1 accuracy but being 8.4x smaller and 6.1x faster than GPipe. EfficientNet-B1 is 7.6x smaller and 5.7x faster than ResNet-152. Details are in Table 2 and 4.

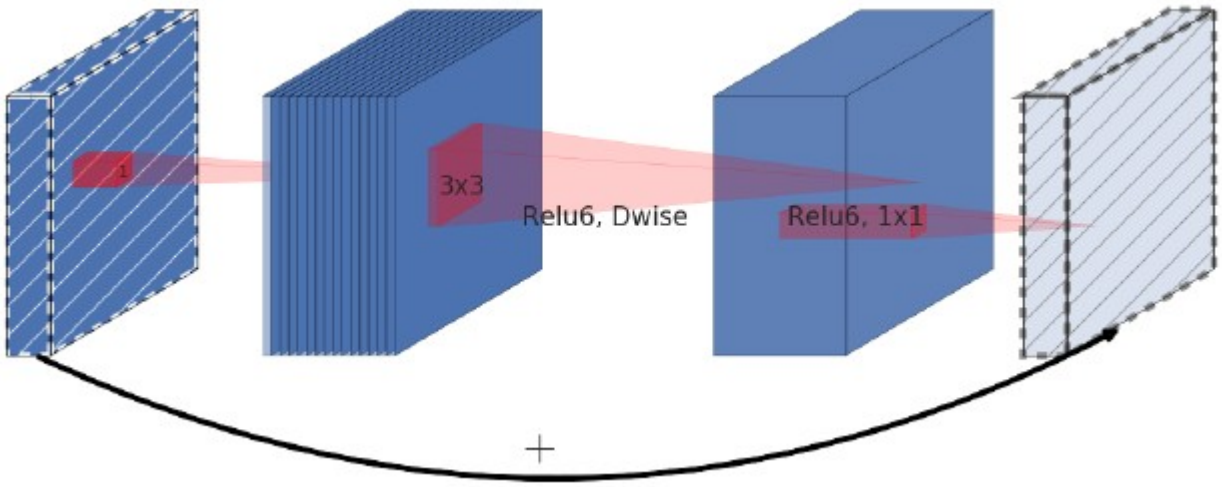
Efficient nets

- $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$ such that for any new ϕ, the total FLOPS will approximately increase by 2^{ϕ}

Combined scaling:

$$\begin{aligned} \text{depth: } & d = \alpha^{\phi} \\ \text{width: } & w = \beta^{\phi} \\ \text{resolution: } & r = \gamma^{\phi} \\ \text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned}$$

MBConv block



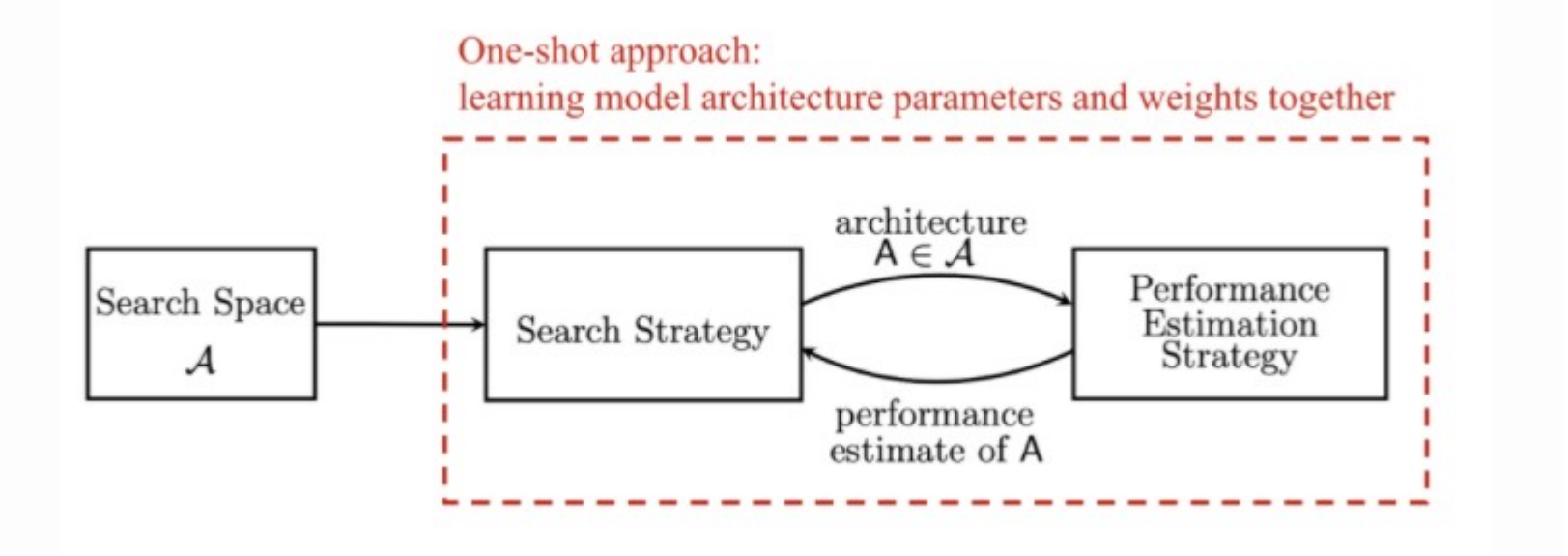

```
from keras.layers import Conv2D, DepthwiseConv2D, Add
def inverted_residual_block(x, expand=64, squeeze=16):
    block = Conv2D(expand, (1,1), activation='relu')(x)
    block = DepthwiseConv2D((3,3), activation='relu')(block)
    block = Conv2D(squeeze, (1,1), activation='relu')(block)
    return Add()([block, x])
```

- Random grid search to determine α = 1.2, β = 1.1, γ = 1.1
- Fix α, β, γ as constants and scale up baseline network with different ϕ (EfficientNet-B1 to B7)

Table 1. **EfficientNet-B0 baseline network** – Each row describes a stage i with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$. Notations are adopted from equation 2.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Neural network architectural search



Search space: define a set of operations and how they can be connected.

Search strategy: samples a population of network architecture candidates: child model performance metrics as rewards and optimizes to generate high-performance architecture candidates.

Performance estimation strategy: measure, estimate, or predict the performance of the proposed child models in order to obtain feedback for the search algorithm to learn

https://lilianweng.github.io/lil-log/2020/08/06/neural-architecture-search.html

Summary – CNN architectures

- Networks are getting more and more deep
- The trend is to not use fully connected layers anymore
- Pre-trained models on large datasets are available
- ResNet and SENet currently good defaults to use
- Convolutional networks design is still a flourishing research area
  - aspects of network architectures are continuously investigated and improved

Summary – CNN architectures

Alexnet

VGG

Inception

ResNet

SeNet

Fractal nets

Densely connected nets

Mobile nets

Efficient nets

Transfer learning

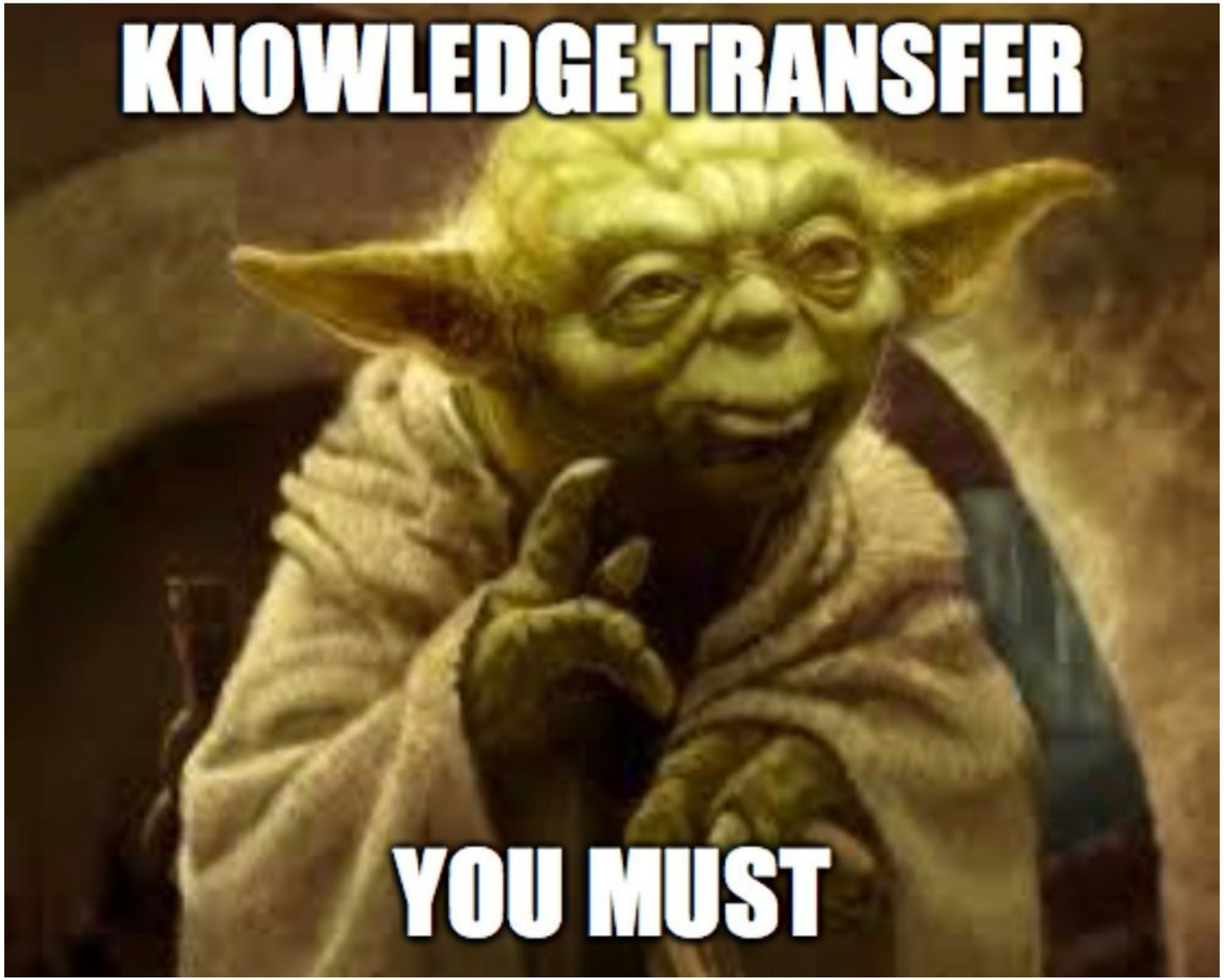
Transfer learning

Have some dataset for a problem you want to solve but it has < ~1M images?

1. Find a very large dataset that has similar data, train a big ConvNet there
2. Transfer learn to your dataset

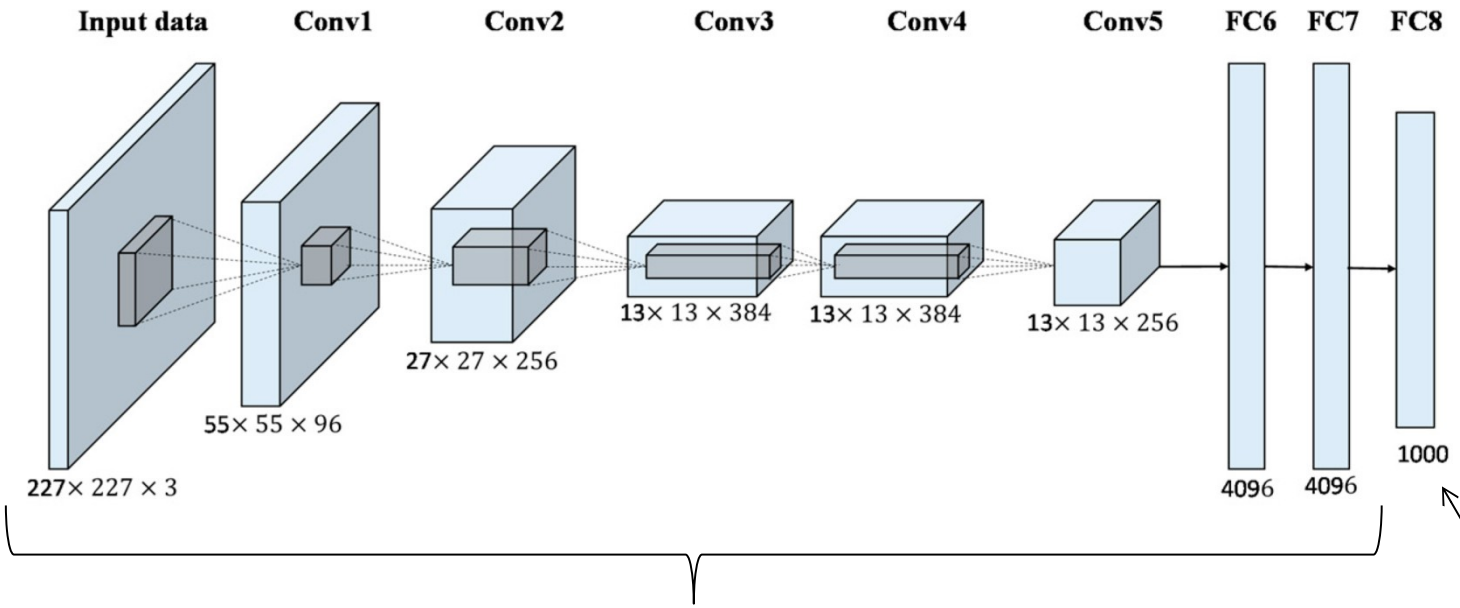
It's the norm, not the exception!

Transfer learning



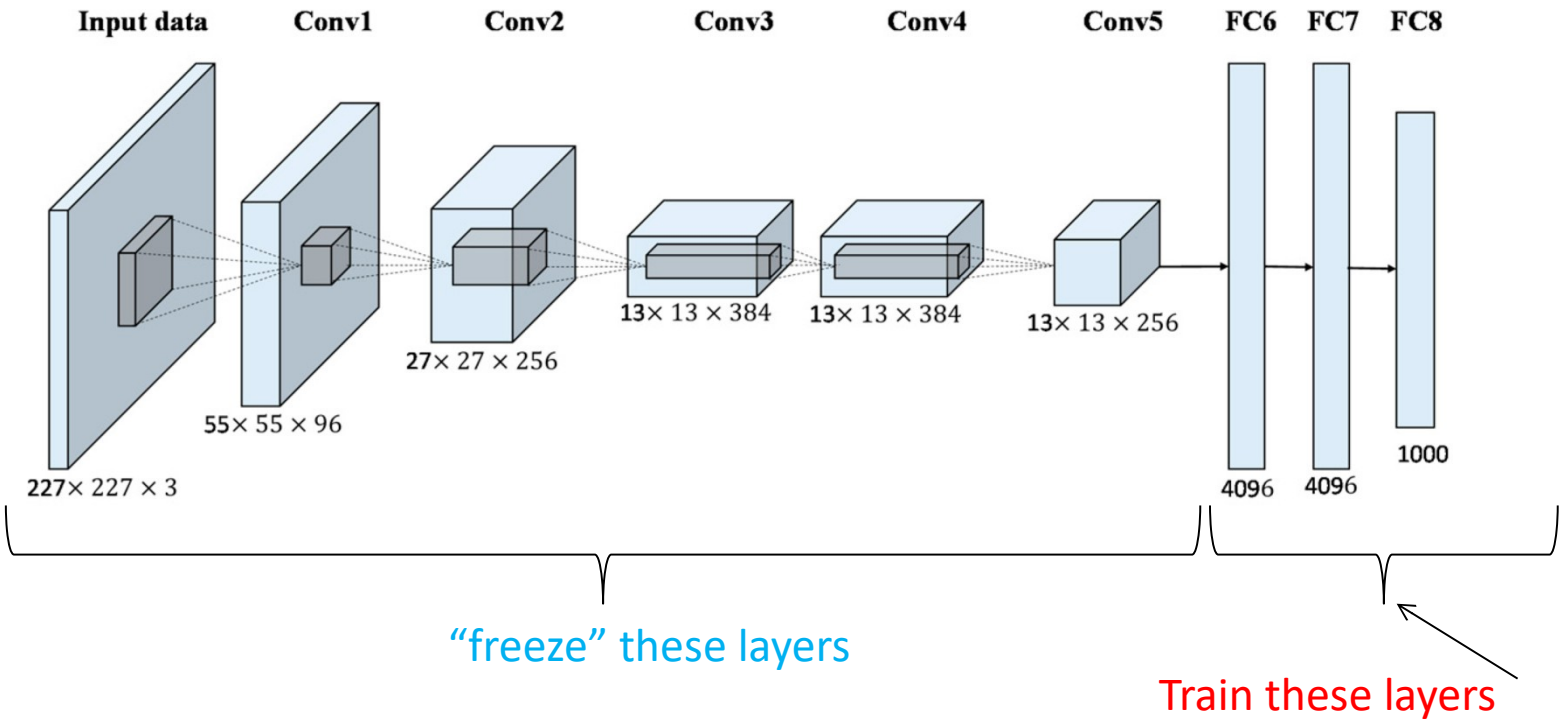
Transfer learning – small database

Train on a similar large dataset: ImageNet, COCO etc.



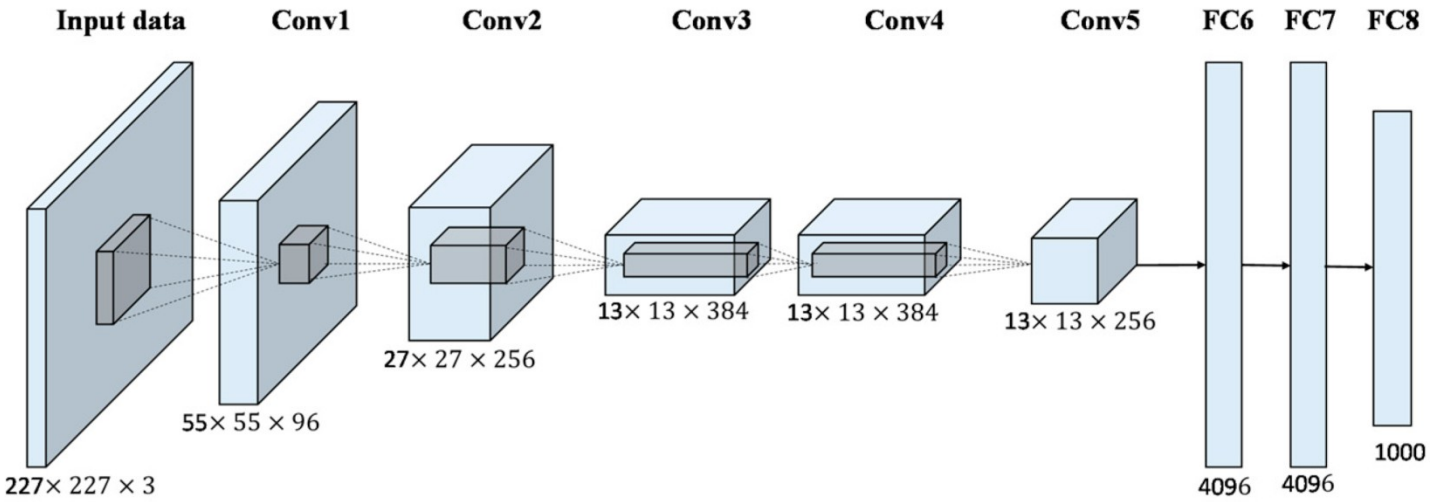
Transfer learning – larger database

Train on a similar large dataset: ImageNet, COCO etc.



Use a lower learning rate when finetuning! 1/10 of the original learning rate is a good starting point

Transfer learning



More generic features

More specific features

Transfer learning

	Similar large dataset	**Different large dataset**
Few training data	Change and train the last layer	☹ linear classifier from different layers in the net ?
Quite a lot of training data	Fine-tune a few layers	Fine-tune a larger number of layers

Transfer learning

- TensorFlow: https://github.com/tensorflow/models
- Keras: https://keras.io/api/applications/
- PyTorch: https://github.com/pytorch/vision

https://modelzoo.co/

Multi-task learning

Case study

An All-In-One Convolutional Neural Network for Face Analysis (2017)

- https://arxiv.org/pdf/1611.00851.pdf
- Key ideas that we should take from this paper:
  - End to end deep learning
  - Multitask learning
  - Designing new loss functions

Fig. 1: The proposed method can simultaneously detect faces, predict their landmarks locations, pose angles, smile expression, gender, age as well as the identity from any unconstrained face image.

Multi-task learning

- When?
  - Training on a set of tasks that could benefit from having shared lower-level features
  - Amount of data you have for each task is quite similar
  - Can train a big enough network to do well on all tasks

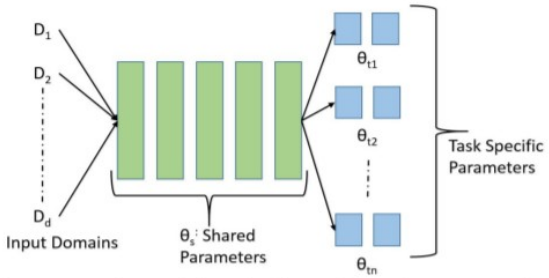


Fig. 2: A general multitask learning framework for deep CNN architecture. The lower layers are shared among all the tasks and input domains.

https://www.youtube.com/watch?v=UaXTsAr4GJw

An All-In-One Convolutional Neural Network for Face Analysis (2017)

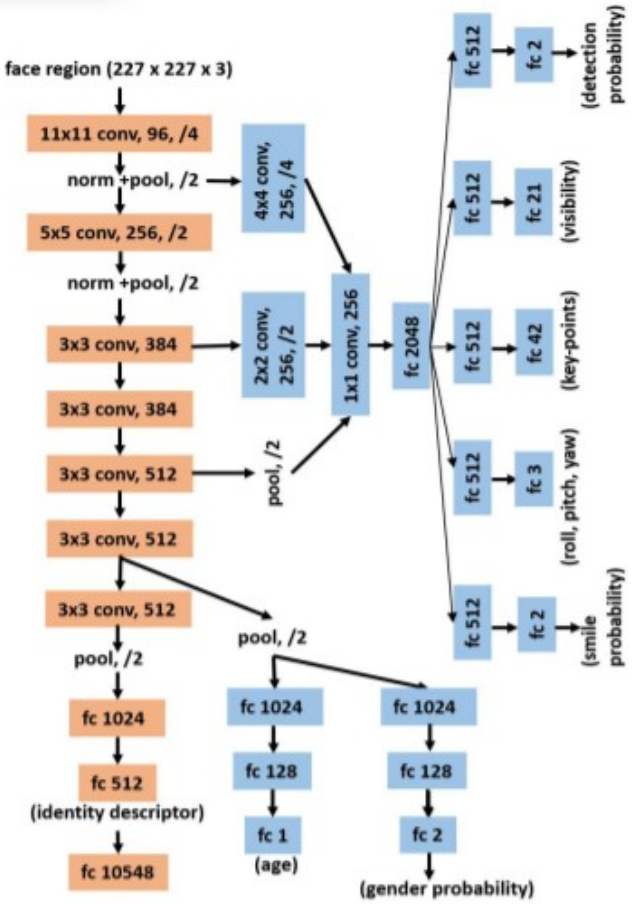


Fig. 3: CNN Architecture for the proposed method. Each layer is represented by filter kernel size, type of layer, number of feature maps and the filter stride. Orange represents the pre-trained network from Sankaranarayanan et al. [41], while blue represents added layers for MTL.

$$L_G = -(1-g)\cdot\log(1-p_g) - g\cdot\log(p_g),$$

$$L_S = -(1-s)\cdot\log(1-p_s) - s\cdot\log(p_s),$$

$$L_A = (1-\lambda)\frac{1}{2}(y-a)^2 + \lambda\left(1 - exp(-\frac{(y-a)^2}{2\sigma^2})\right),$$

$$L_R = \sum_{c=0}^{10547} -y_c\cdot\log(p_c),$$

Dataset	Face Analysis Tasks	# training samples
CASIA [51]	Identification, Gender	490,356
MORPH [39]	Age, Gender	55,608
IMDB+WIKI [40]	Age, Gender	224,840
Adience [27]	Age	19,370
CelebA [31]	Smile, Gender	182,637
AFLW [24]	Detection, Pose, Fiducials	20,342
Total		**993,153**

Multi-task learning

https://www.youtube.com/watch?v=UdXfsAr4Gjw

Face recognition

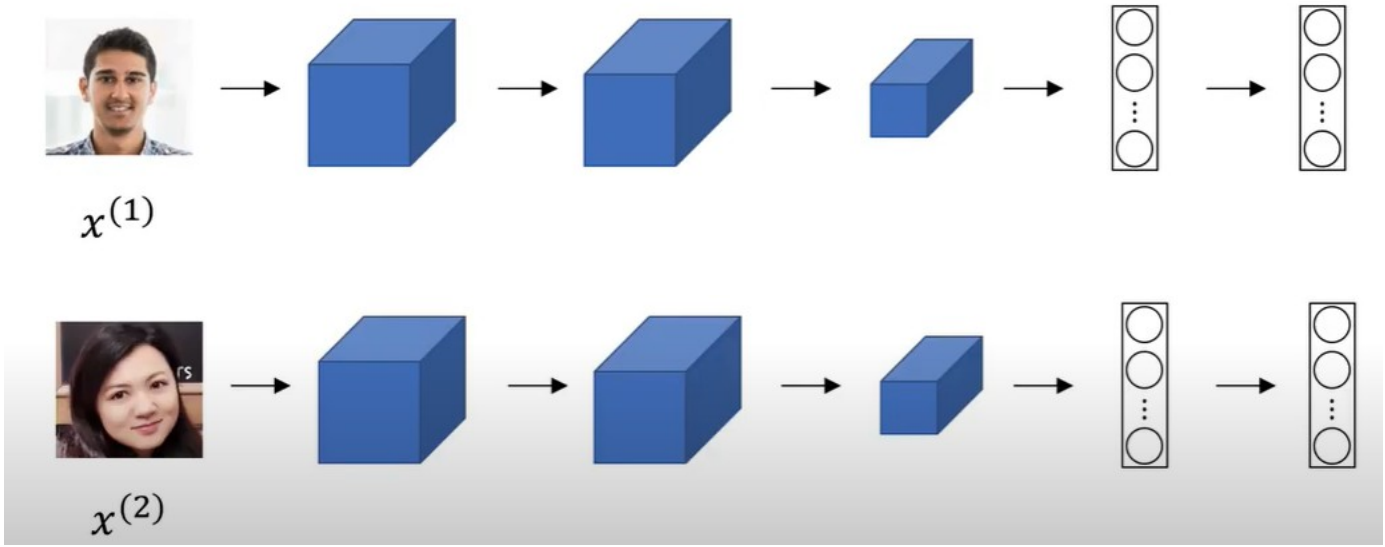
Case study

Face recognition – case study

- Face verification: validating a claimed identity based on the image of a face, and either accepting or rejecting the identity claim (*one-to-one matching*)
- Face recognition: is to identify a person based on the image of a face: the face image has to be compared with all the registered persons (*one-to-many matching*)
- One shot learning

Siamese networks

Two or more inputs are encoded and the output features are compared



If $x^{(i)}, x^{(j)}$ are the same person, $\left\|f(x^{(i)}) - f(x^{(j)})\right\|^2$ is small.

If $x^{(i)}, x^{(j)}$ are different persons, $\left\|f(x^{(i)}) - f(x^{(j)})\right\|^2$ is large.

Example credit: Andrew Ng, deeplearning.ai, Convolutional Neural Networks

```
def build_siamese_model(inputShape, embeddingDim=48):
    # specify the inputs for the feature extractor network
    inputs = Input(inputShape)
    # define the first set of CONV => RELU => POOL => DROPOUT layers
    x = Conv2D(64, (2, 2), padding="same", activation="relu")(inputs)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = Dropout(0.3)(x)
    # second set of CONV => RELU => POOL => DROPOUT layers
    x = Conv2D(64, (2, 2), padding="same", activation="relu")(x)
    x = MaxPooling2D(pool_size=2)(x)
    x = Dropout(0.3)(x)
    # prepare the final outputs
    pooledOutput = GlobalAveragePooling2D()(x)
    outputs = Dense(embeddingDim)(pooledOutput)
    # build the model
    model = Model(inputs, outputs)
    # return the model to the calling function
    return model
```

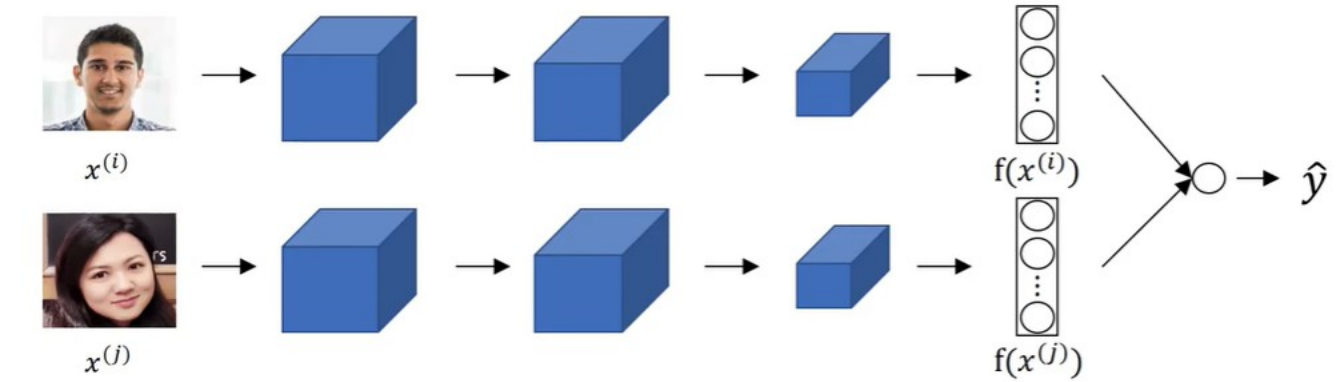
https://www.pyimagesearch.com/2020/11/30/siamese-networks-with-keras-tensorflow-and-deep-learning/

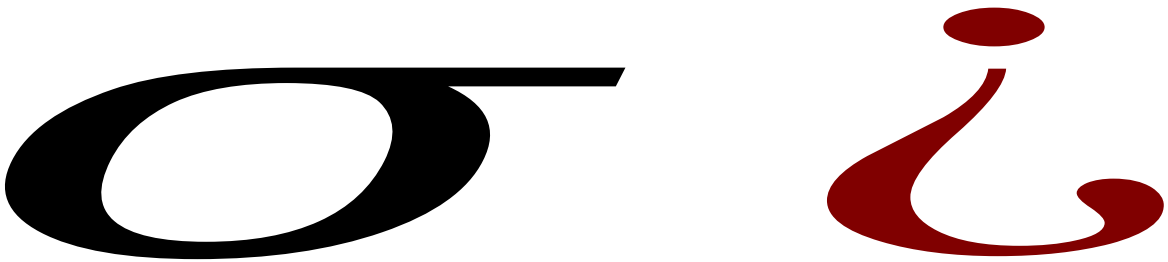
```
imgA = Input(shape=config.IMG_SHAPE)
imgB = Input(shape=config.IMG_SHAPE)
featureExtractor = build_siamese_model(config.IMG_SHAPE)
featsA = featureExtractor(imgA)
featsB = featureExtractor(imgB)
distance = Lambda(utils.euclidean_distance)([featsA, featsB])
outputs = Dense(1, activation="sigmoid")(distance)
model = Model(inputs=[imgA, imgB], outputs=outputs)

model.compile(loss="binary_crossentropy", optimizer="adam",
metrics=["accuracy"])
# train the model
print("[INFO] training model...")
history = model.fit(
[pairTrain[:, 0], pairTrain[:, 1]], labelTrain[:],
validation_data=([pairTest[:, 0], pairTest[:, 1]], labelTest[:]),
batch_size=config.BATCH_SIZE,
epochs=config.EPOCHS)
```

https://www.pyimagesearch.com/2020/11/30/siamese-networks-with-keras-tensorflow-and-deep-learning/

Face verification as a binary classification problem





Example credit: Andrew Ng, deeplearning.ai, Convolutional Neural Networks