MSE | MASTER OF SCIENCE IN ENGINEERING

Machine Learning

T-MachLe

11. Deep Learning II

Andres Perez Uribe
Jean Hennebert

Plan

11.1 Data augmentation

11.2 CNN model selection

11.3 Evolution of CNN architectures

11.4 Transfer learning & Meta learning

11.5 Troubles with Deep Networks

11.6 Visualization tools for understanding CNN models

Practical Work 11

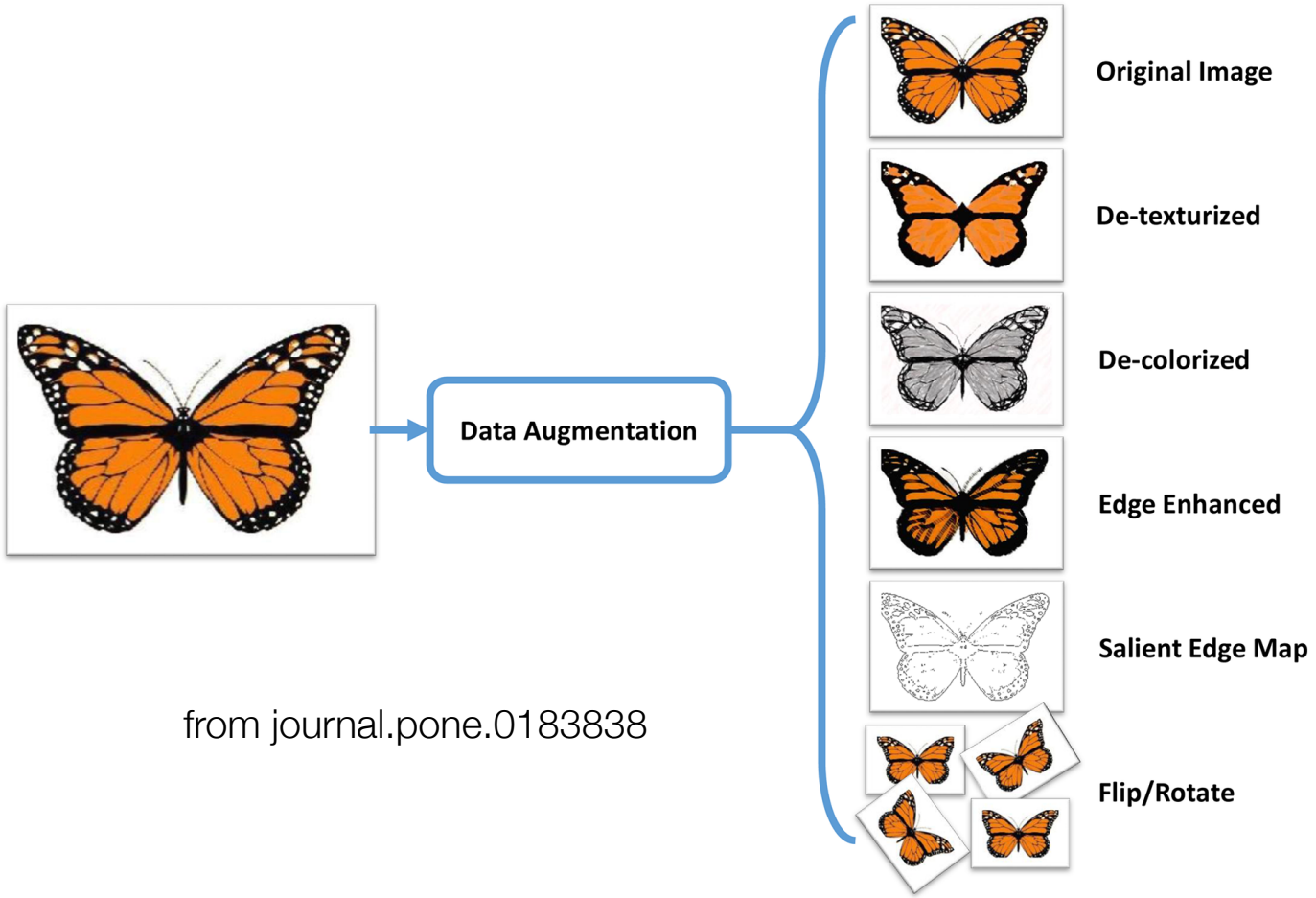
Avoiding overfitting

Steps for reducing overfitting:

- Add more data
- Use data augmentation
- Use architectures that generalize well
- Add regularization
- Add batch regularization
- Reduce architecture complexity

Data augmentation

- rotation, translation, zoom /crop, flips, color perturbation



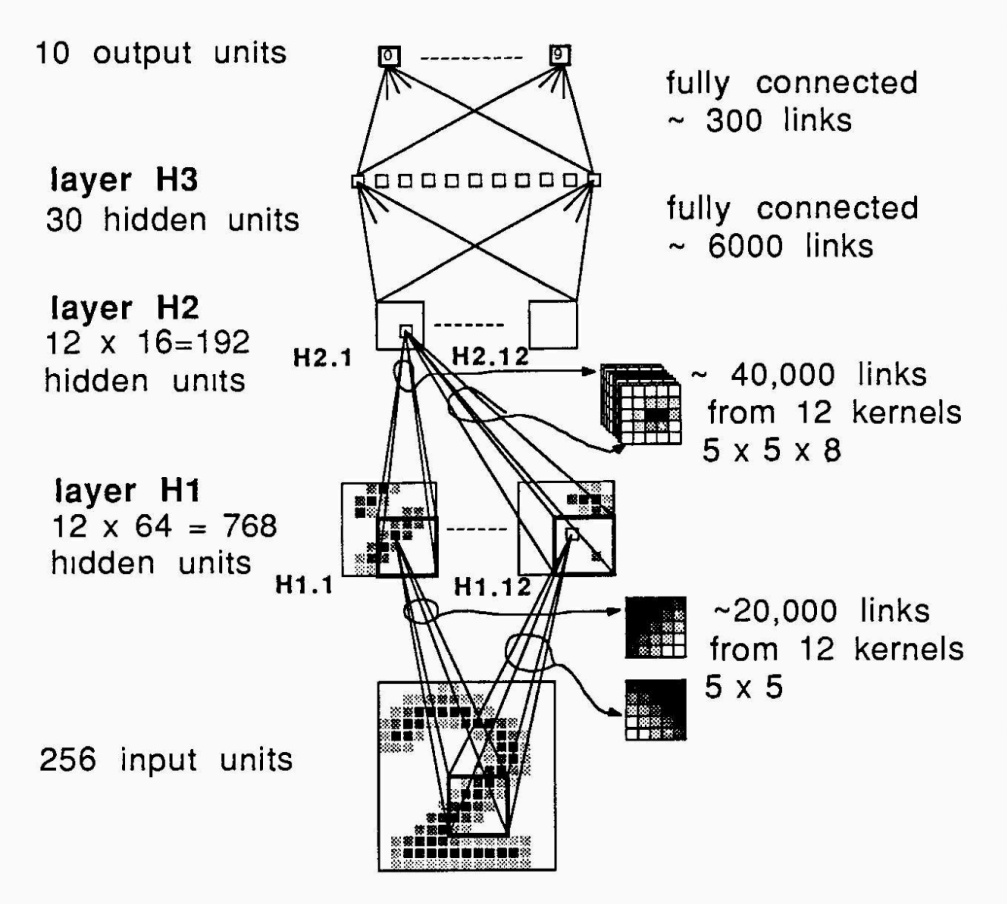
from journal.pone.0183838

CNN: the problem of model selection

- Architecture :
  - How many convolutional layers ?
  - How many filters per layer ?
  - What filter sizes to use ?
  - Where to use a pooling layer ?
  - What activation function to use ?
  - What fully-connected layer configuration is required ?
- Learning algorithm:
  - Optimizer, loss function, learning rates, batch size, training epochs, regularization (L1, L2, dropout), batch normalization

LeNet-5 (1989)

By Yann LeCun and colleagues (AT&T Bell Labs)



- "Backpropagation applied to Handwritten Zip code Recognition", Neural Computation 1
- "Large networks trained by Backpropagation can be applied to real image-recognition problems without complex pre-processing requiring detailed engineering"
- Trained on 9298 digits from US mail during 3 days on a Sun Sparc Station 1

http://yann.lecun.com/exdb/lenet/

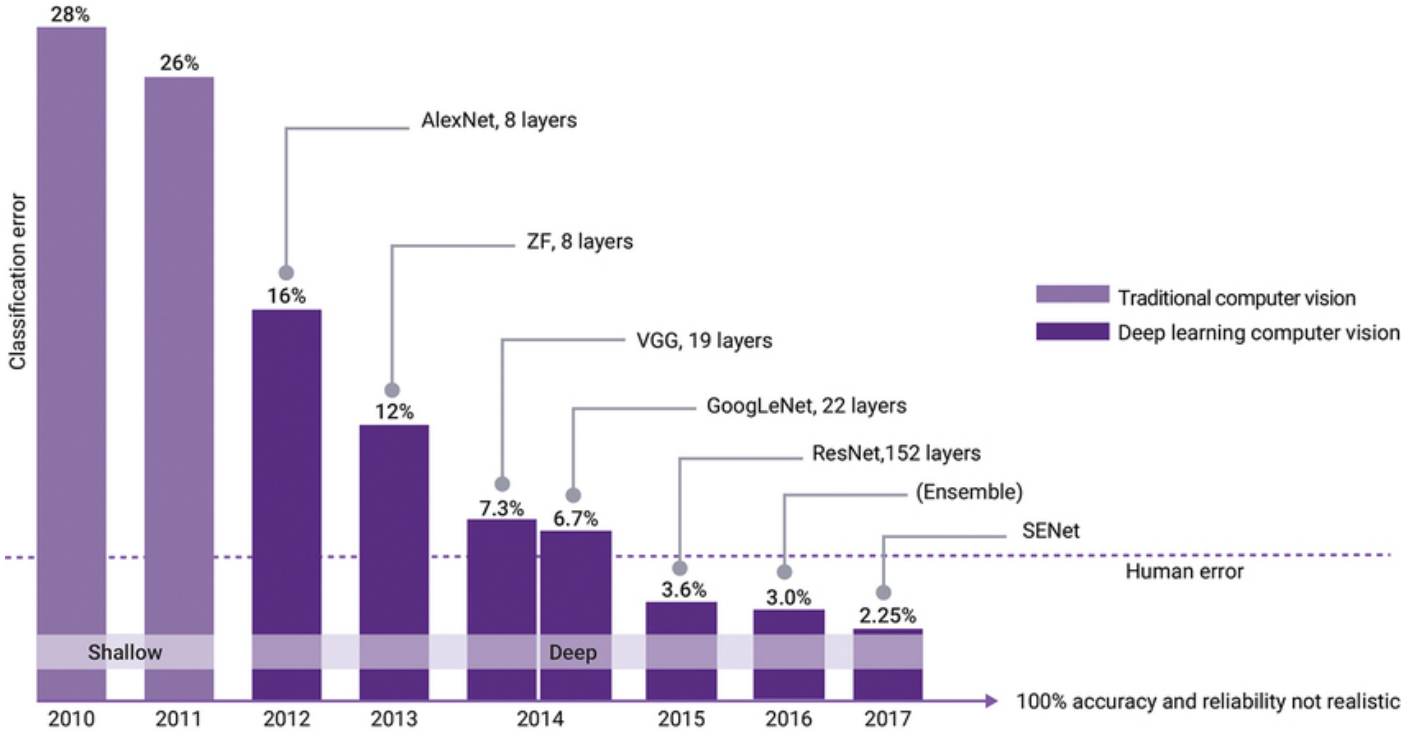
The ILSVRC challenge

- ImageNet Large Scale Visual Recognition Challenges

  Two main competitions:

  I Object detection for 200 fully labeled categories.

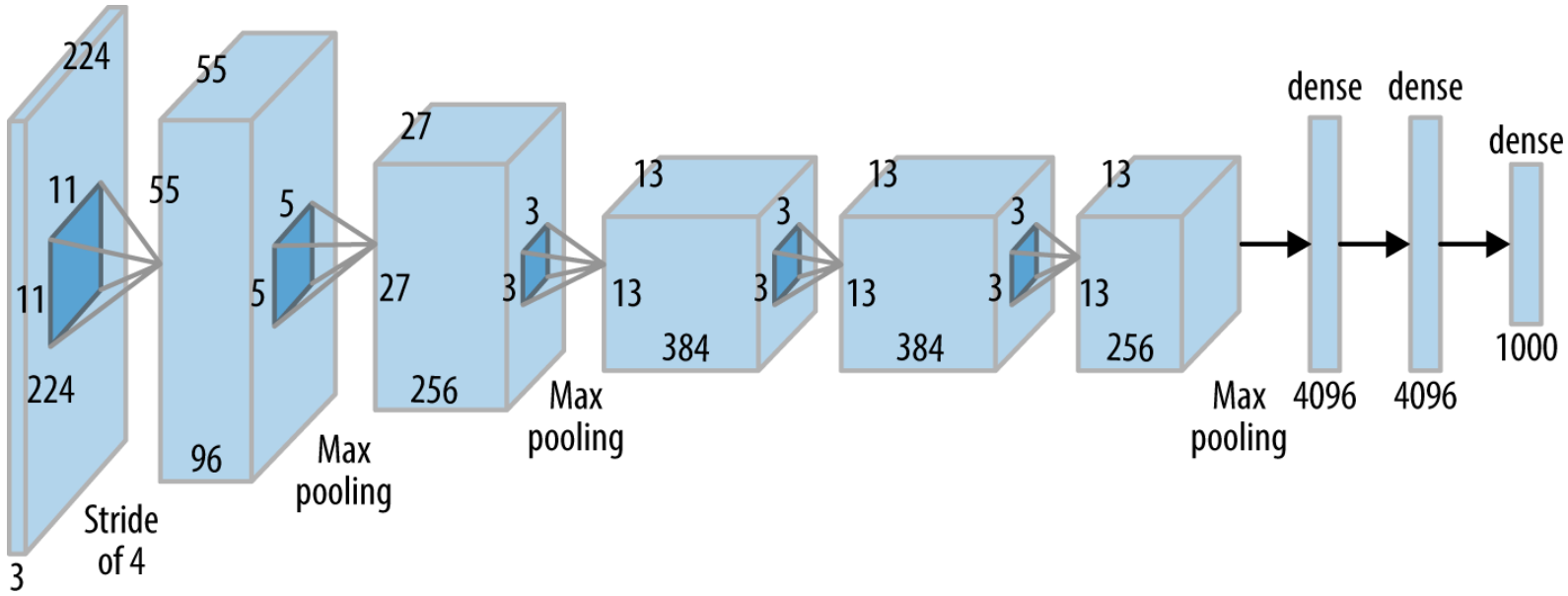
  II Object localization for 1000 categories.



from synopsys.com

AlexNet (2012)

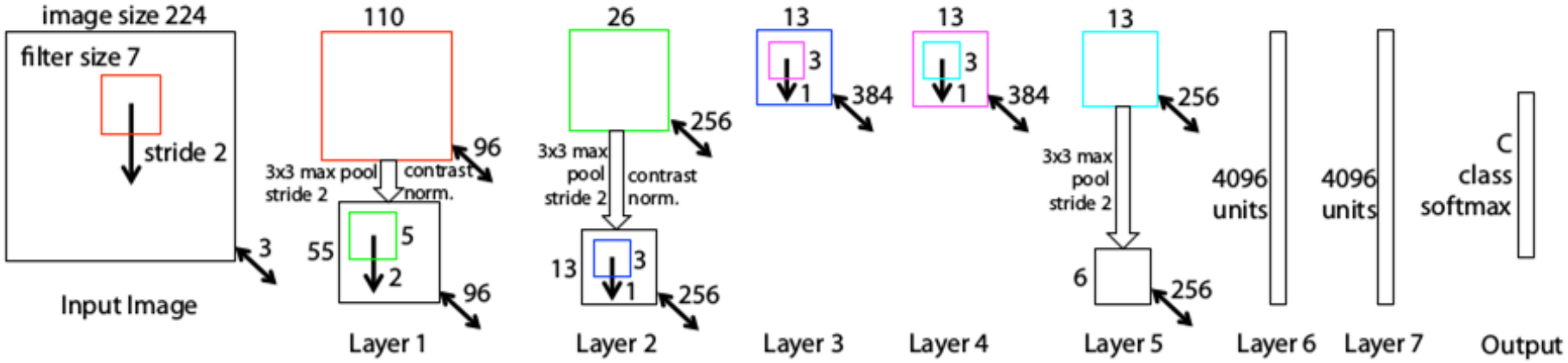
- By Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton
- 62-million parameters model trained on ImageNet data using batch stochastic gradient descent, with specific values for momentum and weight decay. Trained on two GTX 580 GPUs for 5 to 6 days.
- Used ReLus (instead of tanh), data augmentation and dropout.
- Won the 2012 ILSVRC (ImageNet Large-Scale Visual Recognition Challenge): error = 15.4% (second place 26.2%).



from “9 Deep Learning papers ou need to know about”, by Adit Deshpande

ZF Net (2013)

- by Matthew Zeiler and Rob Fergus from NYU
- Trained on only 1.3M images (ImageNet) using batch stochastic gradient descent to minimize cross-entropy.
- Used smaller kernel sizes (7x7) and increasing number of filters
- Trained on two GTX 580 GPUs for 12 days.
- Won the 2013 ILSVRC: error = 11.2%
- Developed DeconvNets to analyze filter activations



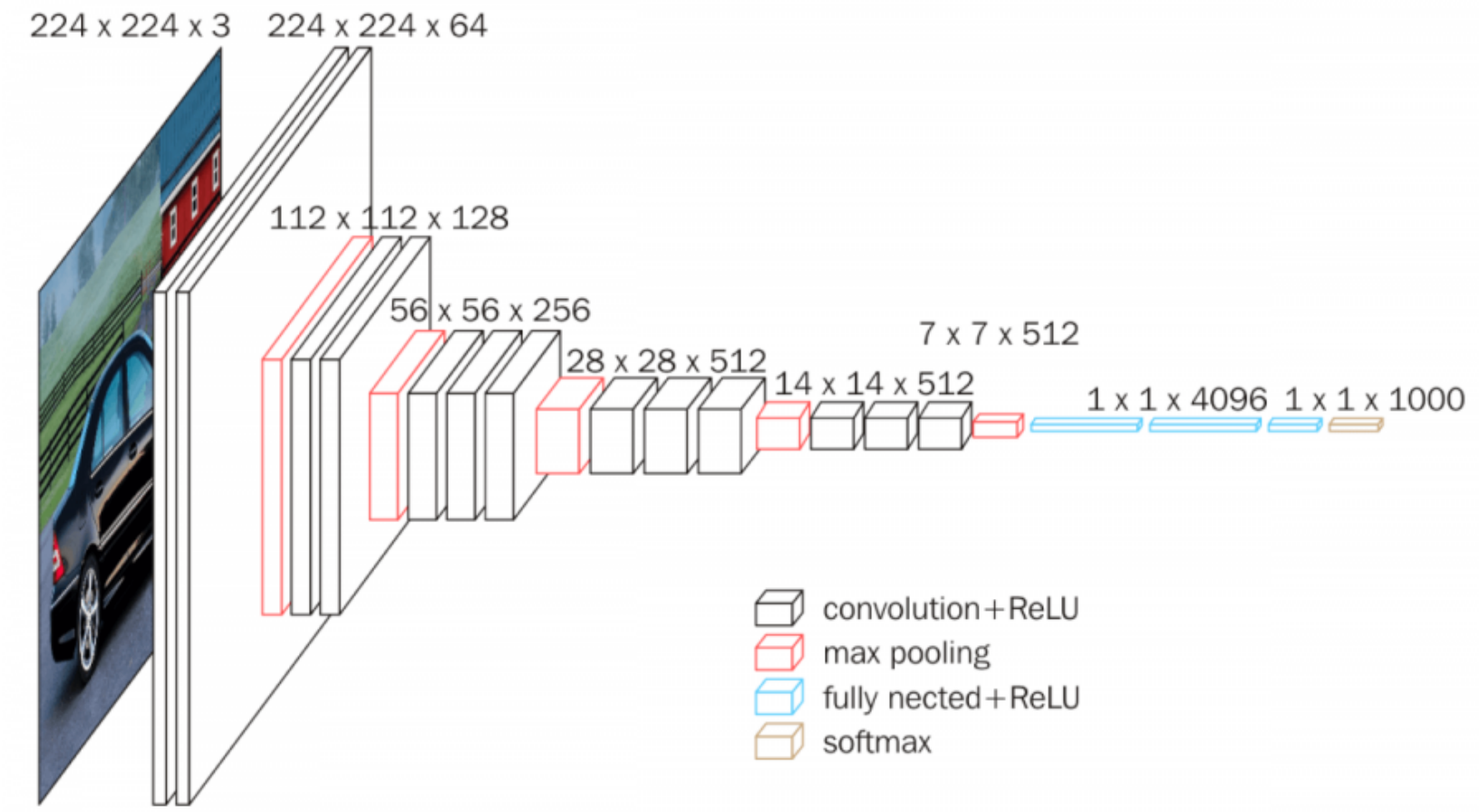
from "9 Deep Learning papers ou need to know about", by Adit Deshpande

VGG Net (2014)

- by K. Simonyan and A. Zisserman from the Visual Geometry Group of the University of Oxford
- VGG-16 is a 16-layer CNN and VGG-19 is a 19-layer CNN that strictly used 3x3 filters with stride and pad of 1, along with 2x2 maxpooling layers with stride 2
- Number of kernels: 64, 128, 256, 512, 512
- Did not win ILSVRC (error 7.3%) but introduced a simple architecture
- Trained on 4 Nvidia Titan Black GPUs for two to three weeks
- VGG-16 has more thant 138-million parameters and VGG-19 has almost143-million parameters. Weights are available at *http://www.robots.ox.ac.uk/~vgg/research/very_deep/*

Ideas behind VGG Nets

- VGGNet was born out of the need to reduce the # of parameters in the CONV layers and improve on training time.

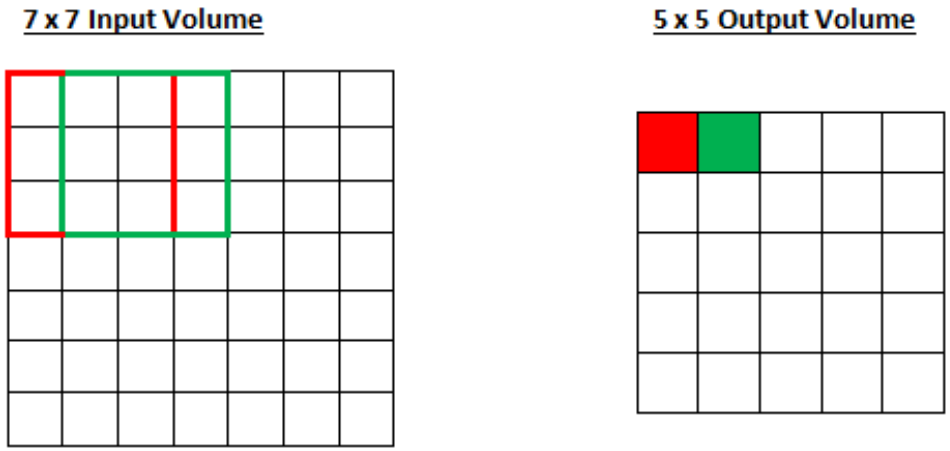


https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

Downsample input with stride

Stride: the amount of movement between applications of the filter to the input image.

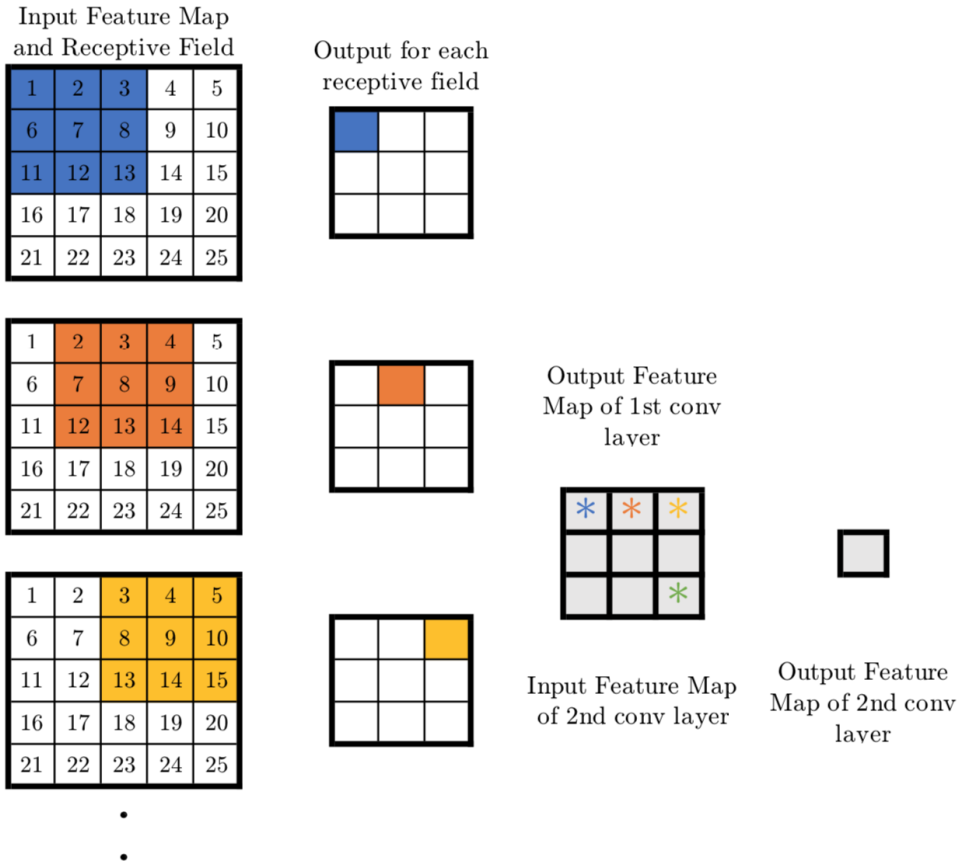
Example: no padding and stride=1



The result is a downsampled image at the output, e.g., 5x5 instead of 7x7.

Ideas behind VGG Nets (2)

- The Alexnet convolutional kernels of sizes 11x11, 5x5, and 3x3 can be replicated by making use of multiple 3x3 kernels as building blocks, but reducing the number of parameters to learn.

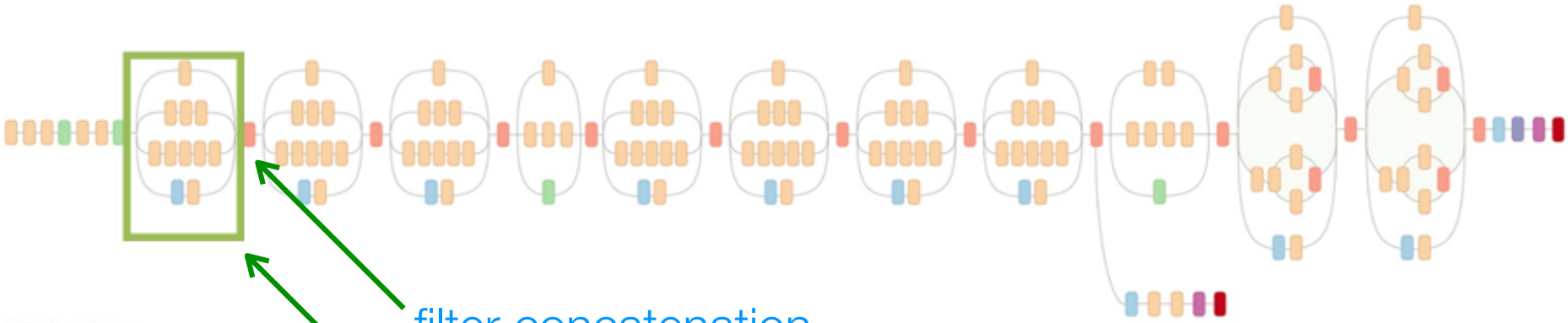


- Suppose a 5x5x1 input. A 5x5 kernel (25 parameters) will produce a 1x1 output
- This can be computed by two layers of 3x3 kernels, stride=1 too (2 x 9 parameters)
- Similarly a 11x11 filter (121 parameters) can be computed by five 3x3 kernels, stride=1 (45 parameters)

https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

GoogLeNet (2015)

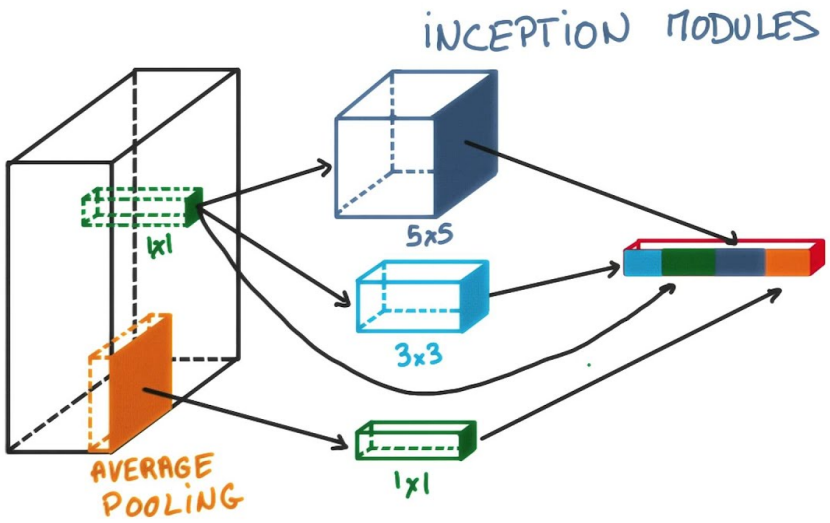
- Winner of ILSVRC'14 (error 6.7%)
- A new architecture with more than 100 layers in total, but not necessarily stacked up sequentially and without fully-connected layers. It uses 12x less weights than AlexNet



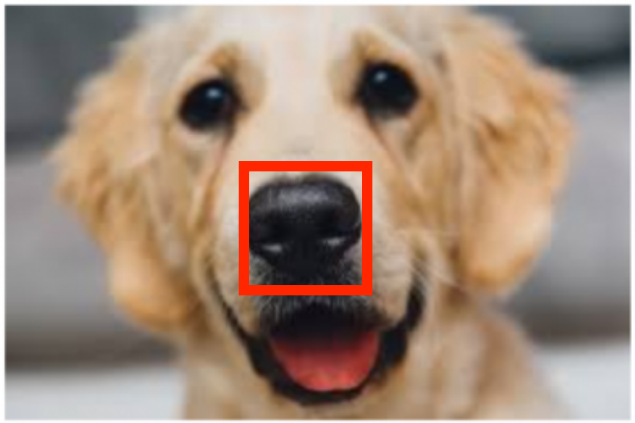
Green box shows parallel region of GoogLeNet

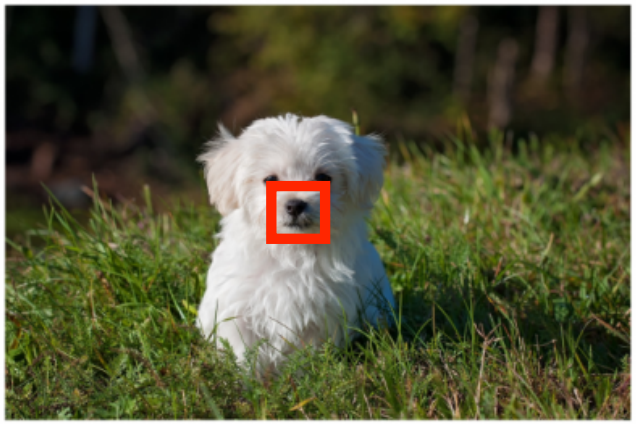
from "9 Deep Learning papers ou need to know about", by Adit Deshpande

Ideas behind Inception Networks



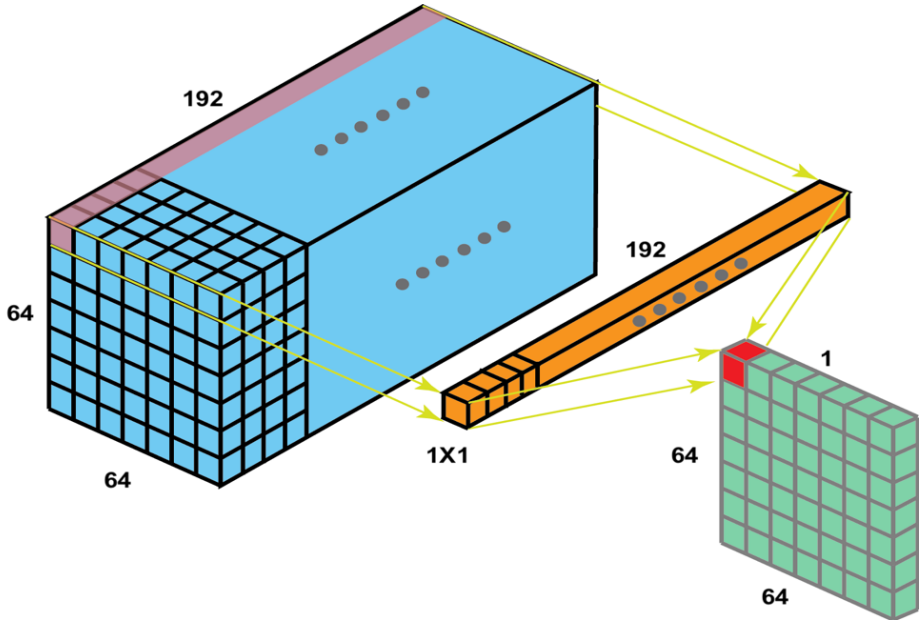
- Instead of choosing a single size (1x1, 3x3 or 5x5) kernel for a layer, use them “all” and concatenate the outputs

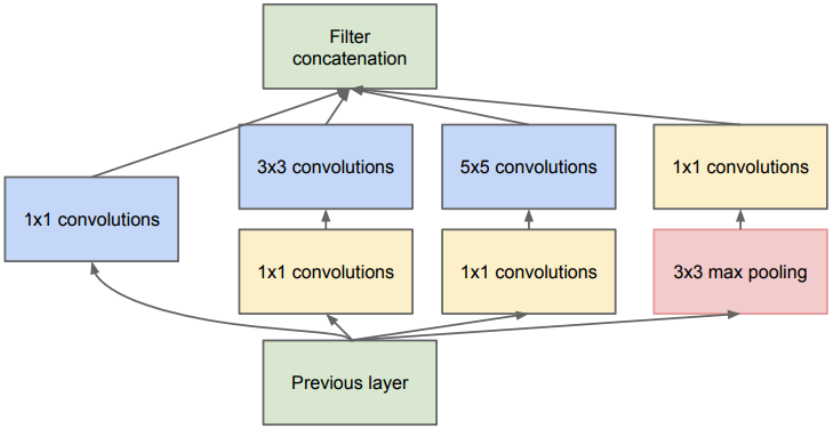




- different kernel sizes allows the identification of features at different scales

Ideas behind Inception Networks (2)





- Apply 1x1 convolutions first to perform feature pooling or dimensionality reduction and reduce the number of operations.

Batch-normalization

Sergey Ioffe and Christian Szegedy from Google (2015)

Problem: when two inputs are in completely different scales, say a range of x1 is [1000–2000] and range of x2 is [0.1–0.5], using them as-is has implications on optimizing the loss function (e.g., by gradient descent). Intuitively, the model will tend to be more influenced by x1.

To deal with this problem, we generally normalize inputs before using a neural network. Normalization, in general refers to squashing a diverse range of numbers to a fixed range.

Ioffe and Szegedy did the same analysis within the network at noticed that the learning was enhanced when the inputs to a subsequent sub-network or layer are also normalized.

Finally, they introduced a “learnable” normalization transform that is applied to every mini-batch during the learning process.

Batch-normalization (2)

Solution: normalize the inputs (per dimension) per batch (i.e., by setting the mean to zero and the standard deviation to 1), but then use the learnable parameters γ and β to scale and shift the input data. γ and β are learned over all the database and serve to restore the expressiveness of the network (e.g., if all input values to a sigmoid were between 0 and 1, we will only exploit the linearity of the system).

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

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

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

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

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

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Batch-normalization (3)

It has been shown that batch normalization accelerates training convergence (i.e., less learning steps are required), allows larger learning rate values, the training of models using saturating nonlinearities (e.g., sigmoids), and not needing dropout.

It was originally proposed to be used before the non-linearity (e.g., relu), but it has been reported to work even better after it.

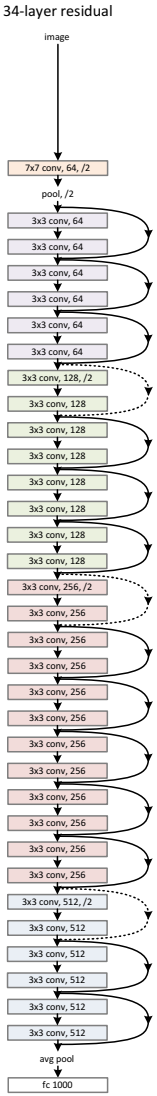
MLP model

```
# example of batch normalization for an mlp
from keras.layers import Dense
from keras.layers import BatchNormalization
...
model.add(Dense(32, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(1))
...
```

CNN model

```
# example of batch normalization for an cnn
from keras.layers import Dense
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import BatchNormalization
...
model.add(Conv2D(32, (3,3), activation='relu'))
model.add(Conv2D(32, (3,3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D())
model.add(Dense(1))
...
```

ResNet (2015)



- A degradation of performance was observed when going deeper (e.g., from 20 to >50 layers) on the ImageNet benchmark.
- A team from Microsoft won ILSVRC'15 (error: 3.6%) using a 152-layer CNN architecture exploiting « shortcut connections » (see below) ResNet152V2 has 60-million weights.
- These so-called ResNet networks appear to have a better performance than plain networks with the same number of weights. Training took 3 weeks using 8 GPUs
- The team tested a 1202-layer network that performed less well.

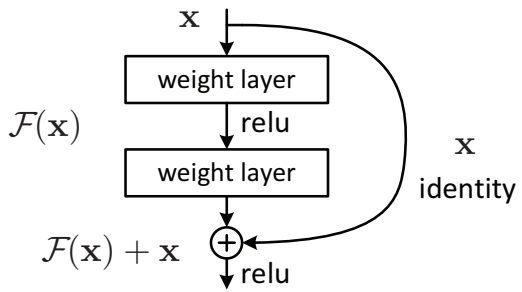
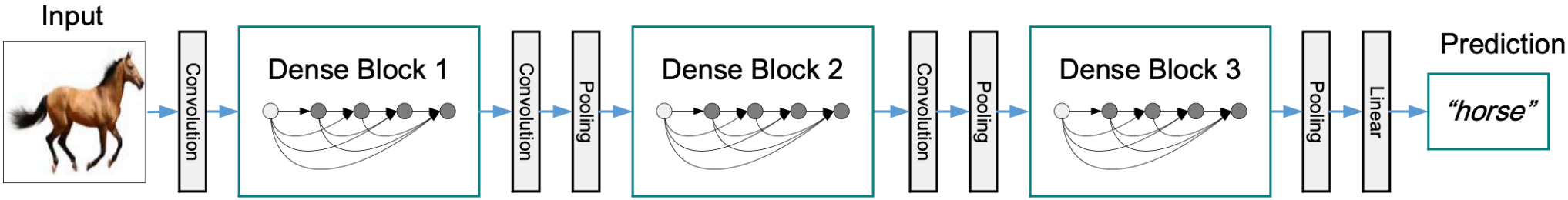
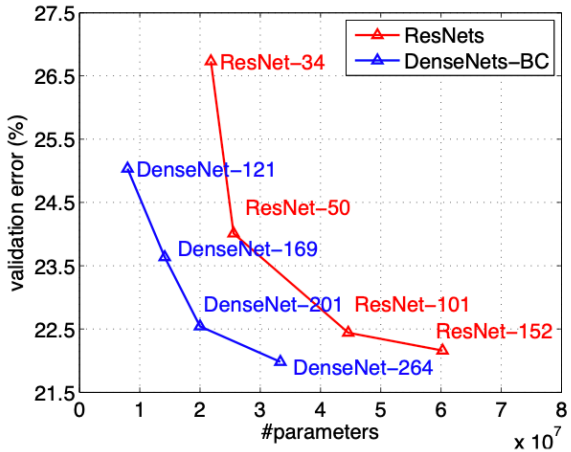


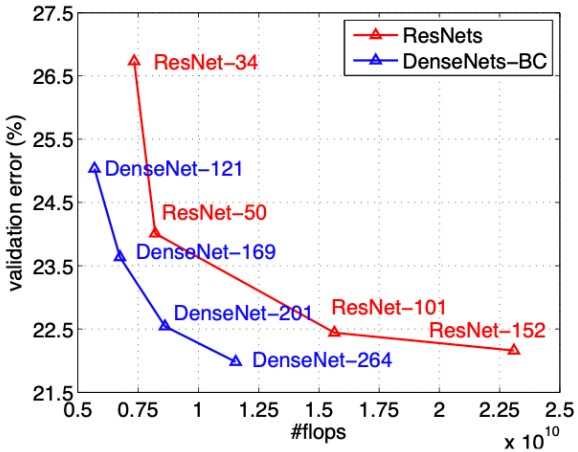
Figure 2. Residual learning: a building block.

DenseNet (2017)



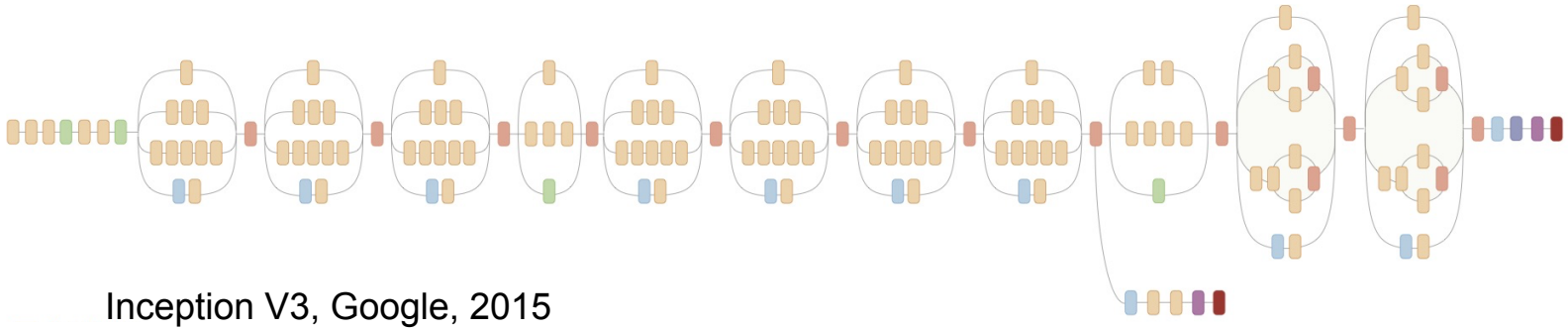
- Researchers from Cornell & Tsinghua universities and Facebook.
- For each layer, the feature-maps (i.e., the output of a convolution) of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters.



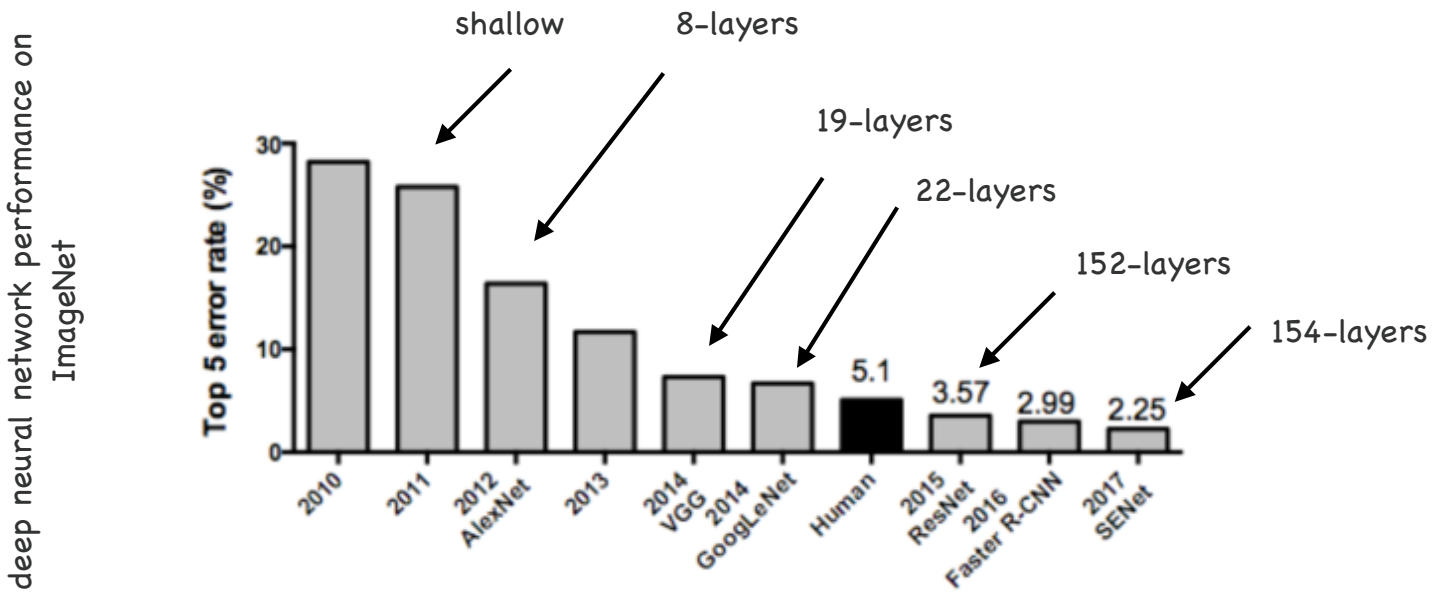


from https://arxiv.org/pdf/1608.06993.pdf

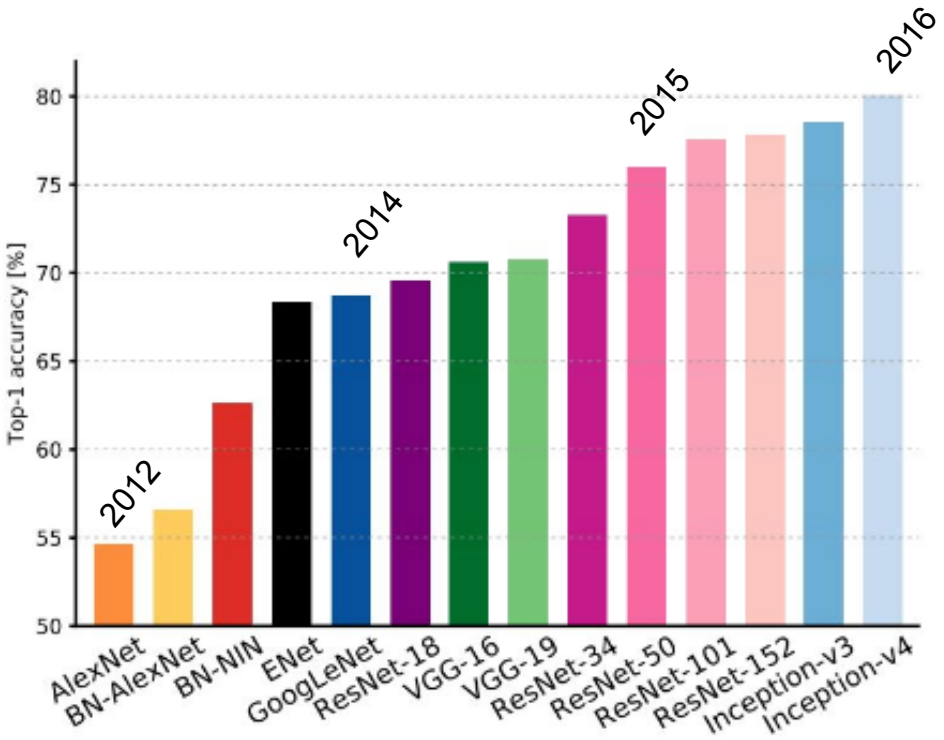
The alchemy of Machine Learning



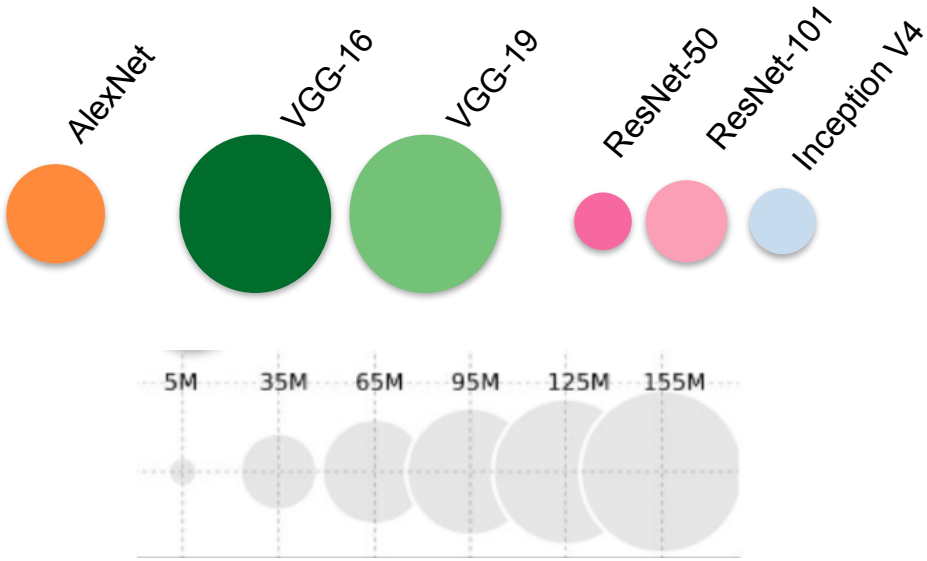
Inception V3, Google, 2015



Evolution of CNN architectures



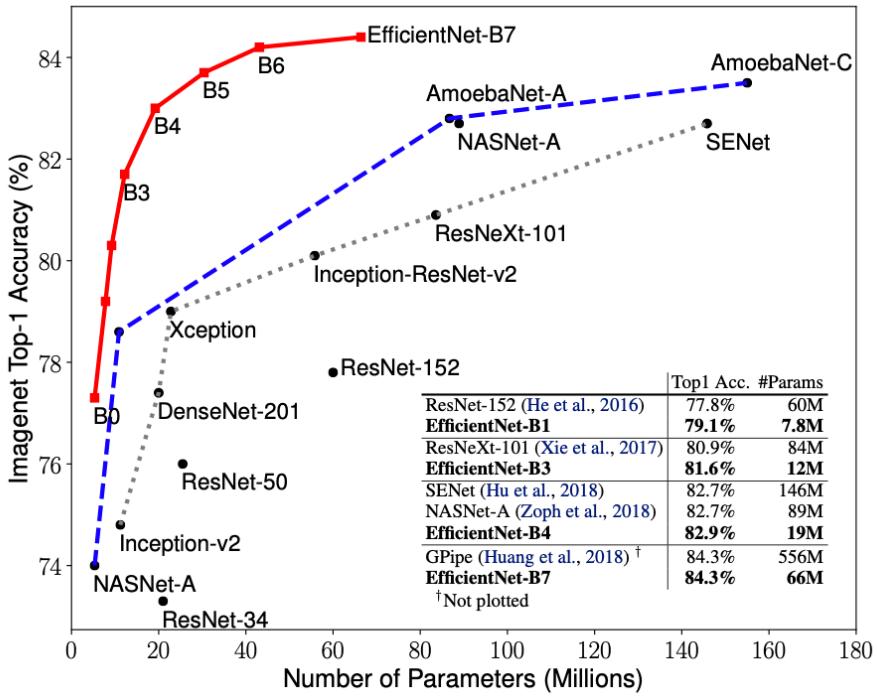
Canziani et al., 2017



Number of parameters

Efficient Nets (2020)

- Mingxing Tan and Quoc Le from the Brain Team of Google Research systematically studied model scaling and identified that carefully balancing network depth, width, and resolution can lead to better performances.



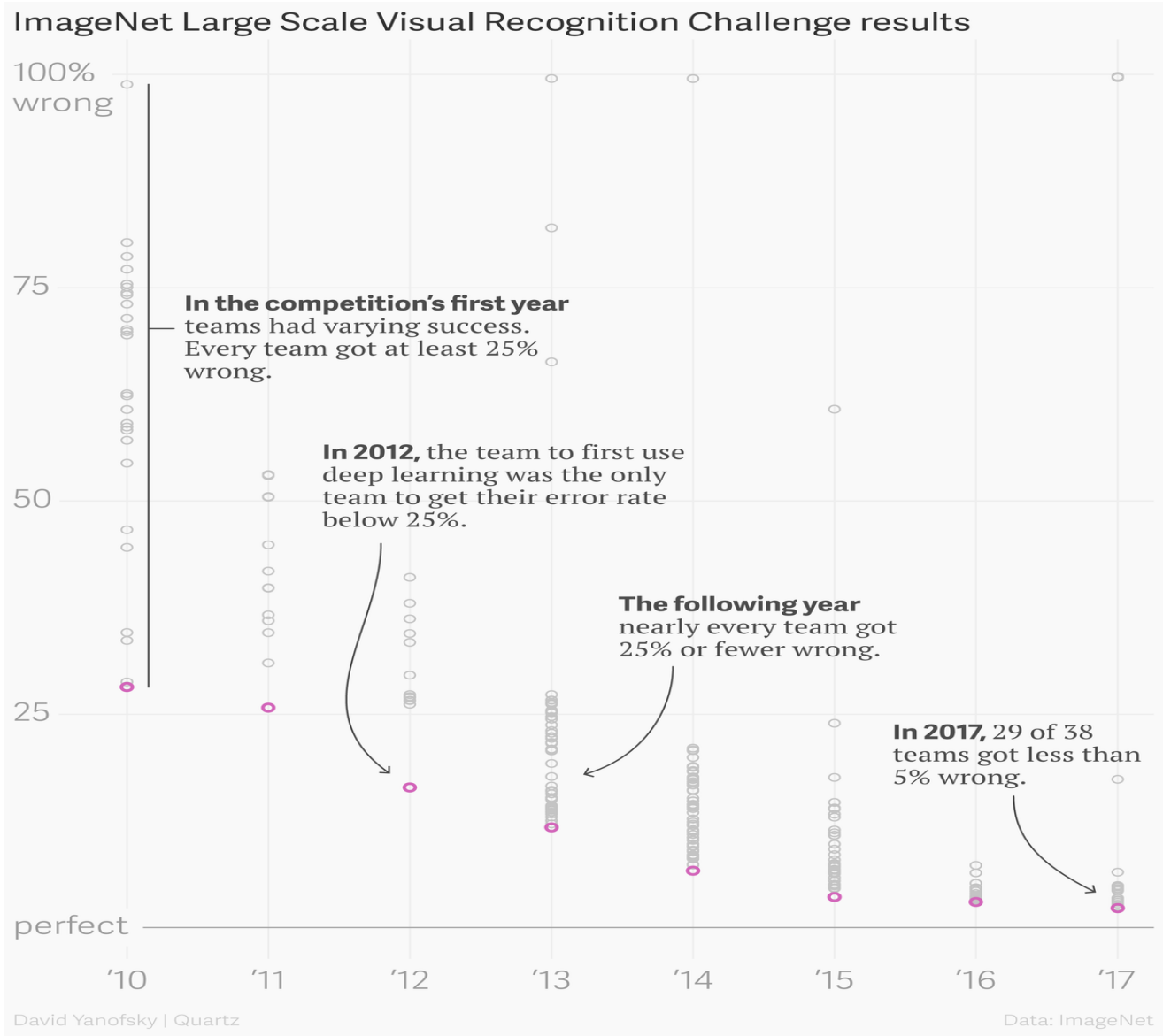
	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

- EfficientNet-B0 performs better than ResNet-50 (~5x less parameters)
- EfficientNet-B1 performs better than ResNet-152 (~7.5x less parameters)
- etc…

from https://arxiv.org/pdf/1905.11946v5.pdf

« Democratization » of Deep Learning



By D. Gershgorn, Quartz. 2017

Pre-trained CNNs

- Object recognition
  - VGGs, ResNets, Inceptions, DenseNets
  - MobileNet (light model for embedded systems)
- Face recognition
  - VGG-Face
- Object localization
  - Mask R-CNN, YOLO, SSD
- Semantic segmentation
  - FCN, U-NET Object localization
- Pose estimation
  - PoseNet, OpenPose
- X-Ray diagnosis
  - CheXNet

Off-the-shelf architectures/models

Available models

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27.0	6.7
NASNetLarge	343	82.5%	96.0%	88.9M	533	344.5	20.0
EfficientNetB0	29	77.1%	93.3%	5.3M	132	46.0	4.9
EfficientNetB1	31	79.1%	94.4%	7.9M	186	60.2	5.6
EfficientNetB2	36	80.1%	94.9%	9.2M	186	80.8	6.5
EfficientNetB3	48	81.6%	95.7%	12.3M	210	140.0	8.8
EfficientNetB4	75	82.9%	96.4%	19.5M	258	308.3	15.1
EfficientNetB5	118	83.6%	96.7%	30.6M	312	579.2	25.3
EfficientNetB6	166	84.0%	96.8%	43.3M	360	958.1	40.4
EfficientNetB7	256	84.3%	97.0%	66.7M	438	1578.9	61.6
EfficientNetV2B0	29	78.7%	94.3%	7.2M	-	-	-
EfficientNetV2B1	34	79.8%	95.0%	8.2M	-	-	-
EfficientNetV2B2	42	80.5%	95.1%	10.2M	-	-	-
EfficientNetV2B3	59	82.0%	95.8%	14.5M	-	-	-
EfficientNetV2S	88	83.9%	96.7%	21.6M	-	-	-
EfficientNetV2M	220	85.3%	97.4%	54.4M	-	-	-
EfficientNetV2L	479	85.7%	97.5%	119.0M	-	-	-

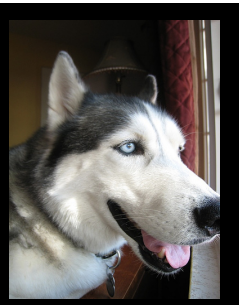
Successful architecture and models (e.g., including weights from previous training) are available for further use and adaptation.

from https://keras.io/applications/

Using a pre-trained model

Let's simply read the weights of a pre-trained model (e.g., Resnet-50 trained with the ImageNet database) and use it to recognize the object in a given image:

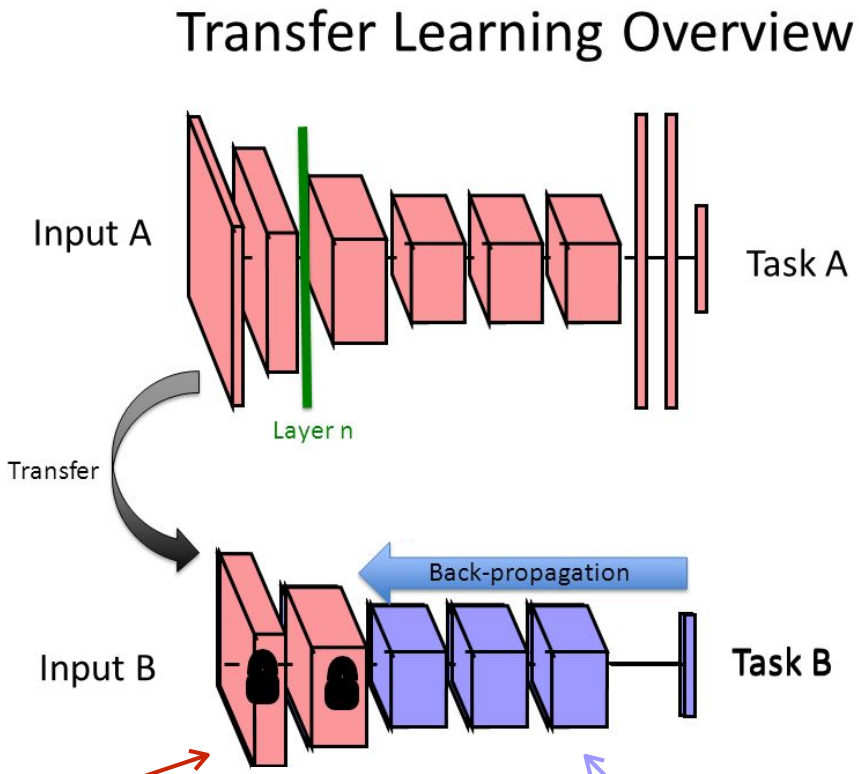
```
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions
import numpy as np

model = ResNet50(weights='imagenet')

img_path = '/ILSVRC2012_val_00005019.JPEG'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

preds = model.predict(x)
# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
print('Predicted:', decode_predictions(preds, top=3)[0])
#Predicted: [('n02109961', 'Eskimo_dog', 0.48957556), ('n02110185', 'Siberian_husky', 0.35920256), ('n02110063',
'malamute', 0.15049036)]
```



from https://keras.io/applications/

Transfer learning (1)

- The idea is to use the first layers of a CNN that was previously trained (i.e., with lots of data) and expect to be able to fine-tune only the subsequent ones in order to use it for a new task.

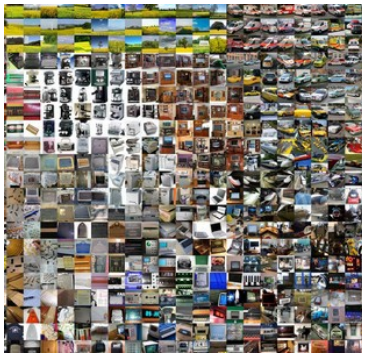


frozen weights (copied from the pre-trained model)

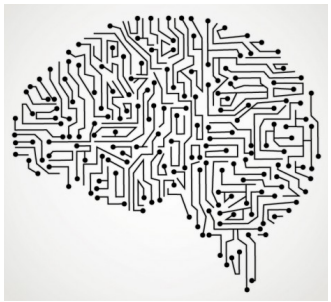
adapted weights

https://arxiv.org/abs/1411.1792

Transfer learning (2)



1.2M images



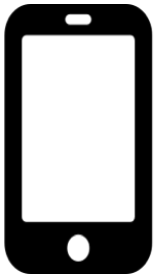
MobileNet
(pre-trained
model by
Google)

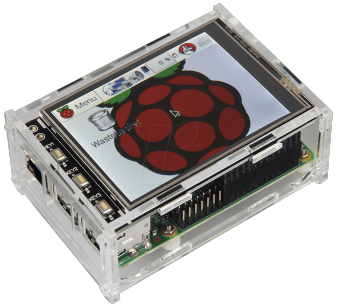
+ MyImages

fine-tuning

Customized
system

Increasing
number of
available models

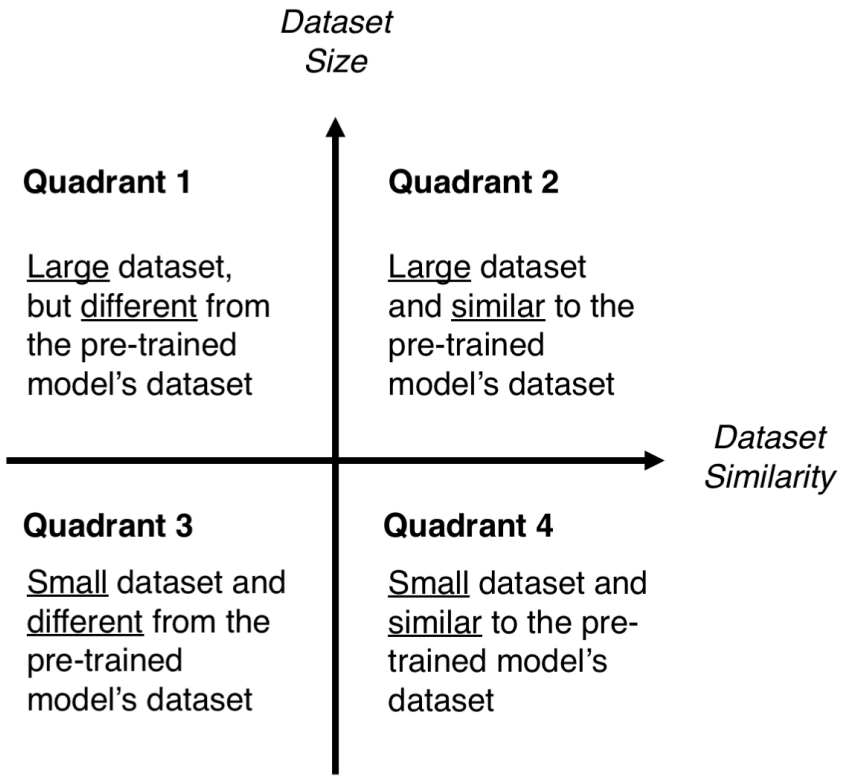


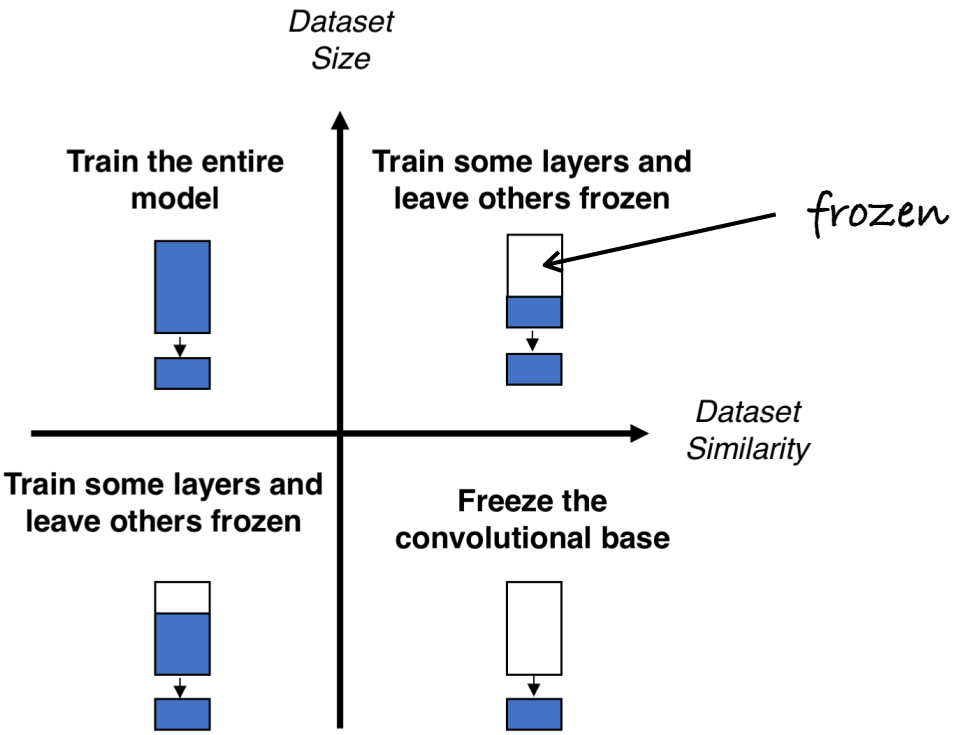


Example: Tensorflow for Poets:

https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/

Quadrants of transfer learning





from Fei Fei Li and https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751

Transfer learning using Keras (1)

The following example defines a new model based on the MobileNet architecture taking all but the last layer. It computes the average of the features computed with the convolutional layers (e.g., using a layer GlobalAveragePooling2D), it adds a Dense layer (1024 neurons) and defines a new input for a 3 classes problem using a softmax activation function.

```
from tensorflow.keras.applications.mobilenet import MobileNet
from tensorflow.keras.applications.mobilenet import preprocess_input, decode_predictions
from keras.layers import Dense,GlobalAveragePooling2D

base_model=MobileNet(weights='imagenet',include_top=False) #imports the mobilenet model and discards
the last 1000 neuron layer.

x=base_model.output
x=GlobalAveragePooling2D()(x)
x=Dense(1024,activation='relu')(x) #we add dense layers so that the model can learn more complex
functions and classify for better results.
predictions=Dense(3,activation='softmax')(x) #final layer with softmax activation

new_model=Model(inputs=base_model.input,outputs=predictions)
```

from https://towardsdatascience.com/transfer-learning-using-mobilenet-and-keras-c75daf7ff299

Transfer learning using Keras (2)

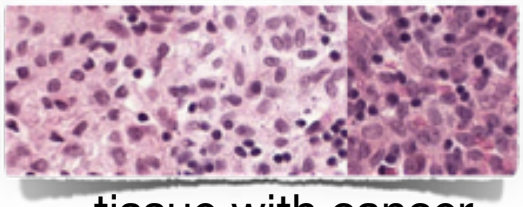
The following code prints the layers composing the new model defined in the previous slide. The second part of the code sets the first 87 layers to “non-trainable” (we also say that we freeze that part of the model) and sets the final two Dense layers to trainable. Finale we compile the new model and train it with the new data.

```
for i,layer in enumerate(new_model.layers):
  print(i,layer.name)

# Freeze the first 87 layers
for layer in new_model.layers[:87]:
    layer.trainable=False
for layer in new_model.layers[87:]:
    layer.trainable=True

# Compile the new model
new_model.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'])

# Fine-tune the new model
new_model.fit(new_train_data, epochs=epochs, validation_data=validation_new_data)
```

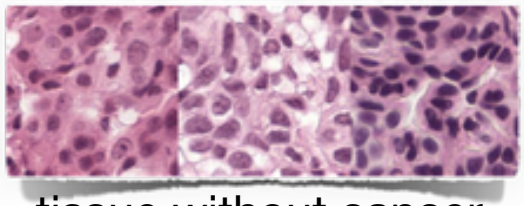
from https://towardsdatascience.com/transfer-learning-using-mobilenet-and-keras-c75daf7ff299

Superhuman performance

Cancer [Google, 03/2017]



tissue with cancer

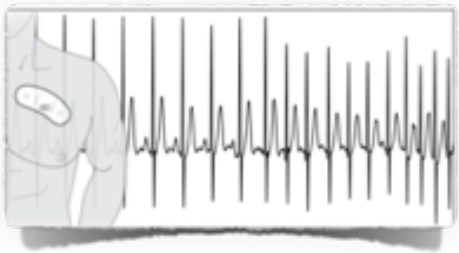


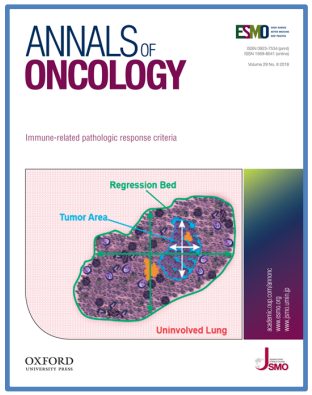
tissue without cancer

AI performance: ~0.92
Human experts perf. : 0.73

Cardiac arrhythmia
[Stanford, 07/2017]

AI performance: ~0.8
Cardiologist perf.: ~0.75





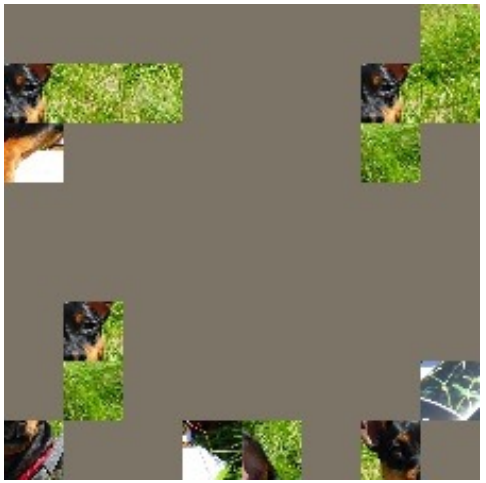
Aug. 2018

Study from University of Heidelberg showed that AI outperformed most of 58 dermatologists (30 experts) in dermoscopic melanoma recognition

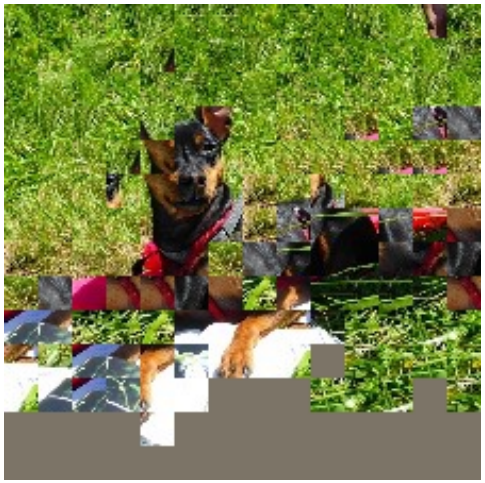
Troubles with Deep Networks

Output maximization by image occlusion

0.47



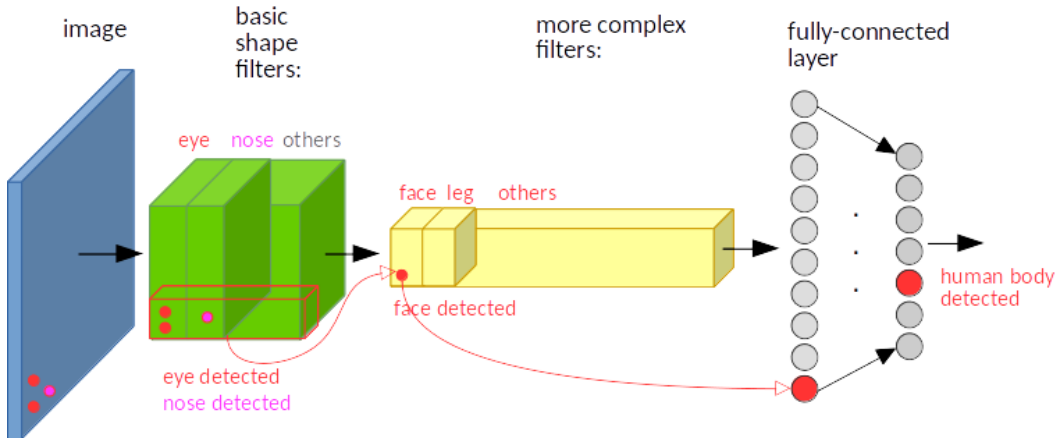
0.97

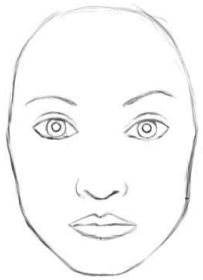


0.99

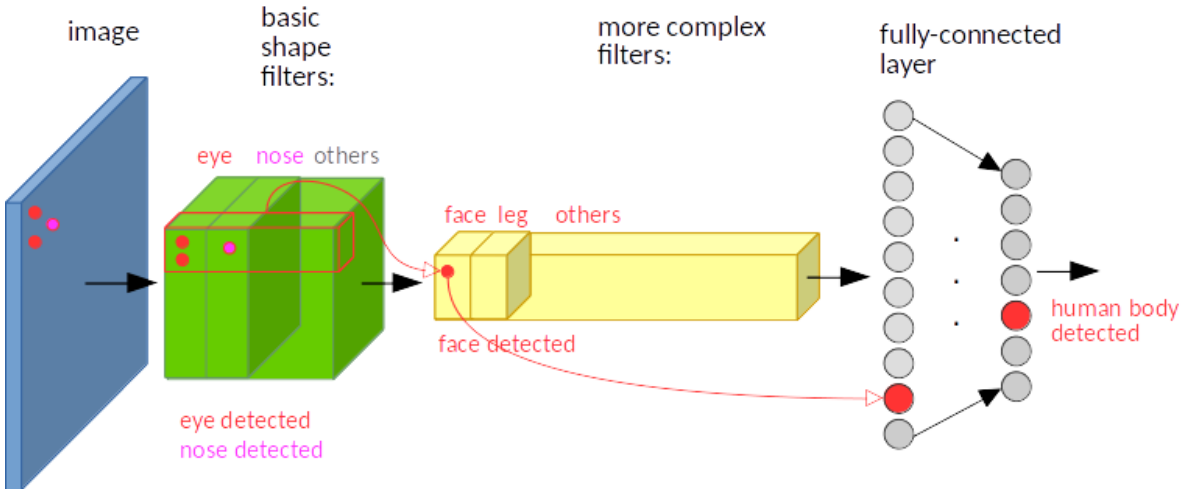
pepita = miniature pinscher ? (Satizabal, 2016)

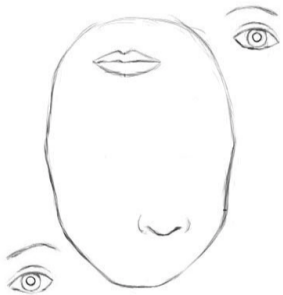
Spatial translation invariance





person: 0.88

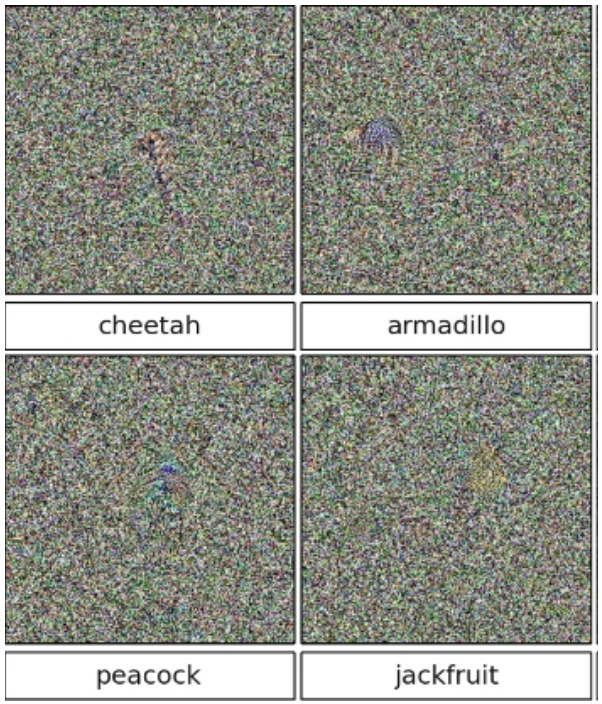




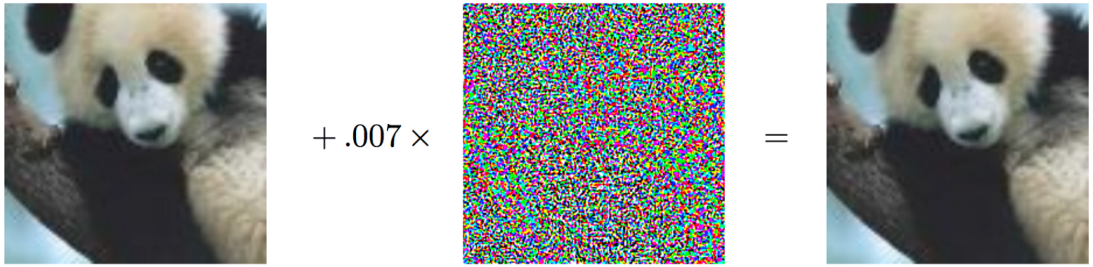
person: 0.85

from hackernoon.com

Adversarial attacks



Nguyen, Yosinski, Clune, 2014

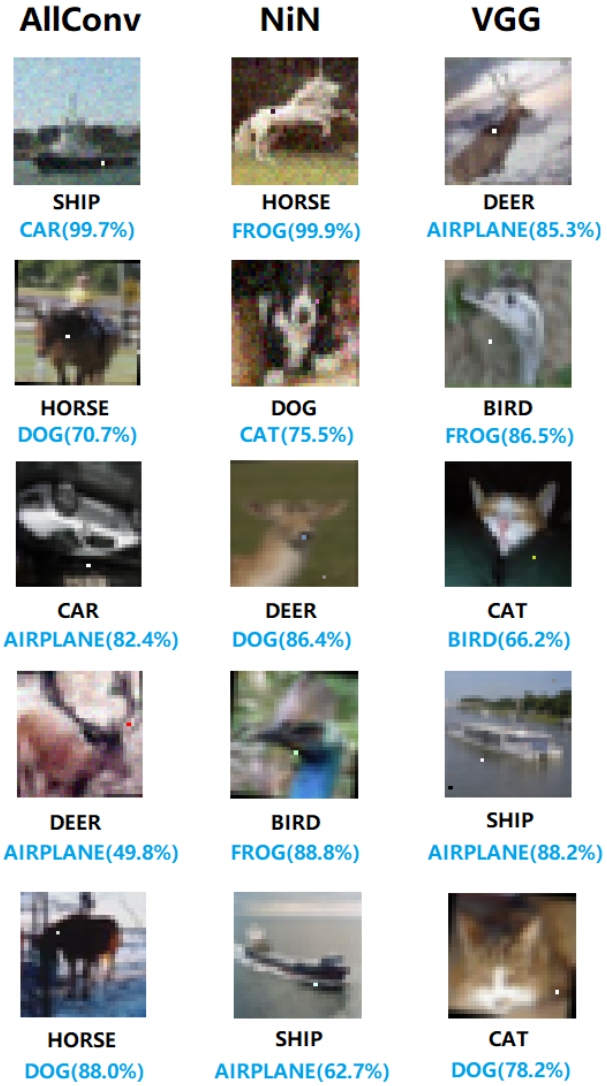


Goodfellow et al, ICLR 2015

Physical-robust attack: a STOP signal is perceived as a speed limit sign (max 45 mph)

K Eykholt et al, CVPR 2018

One pixel attack for fooling CNNs



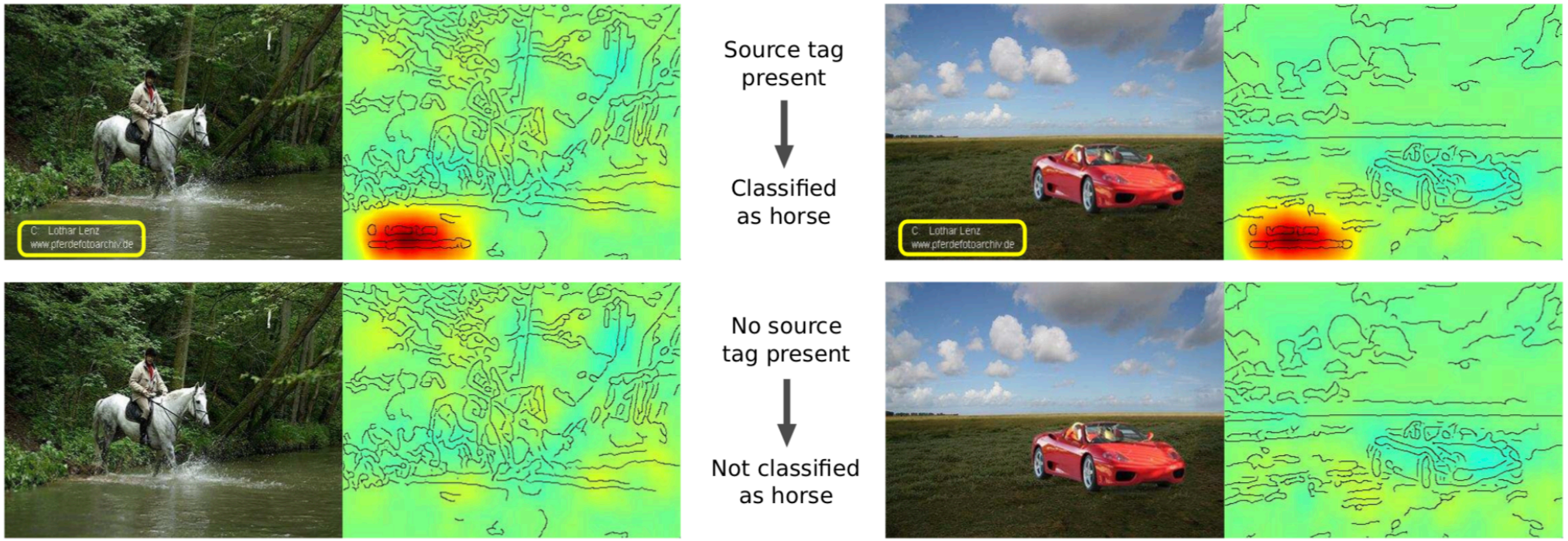
- Jiawei Sun, Danilo Vasconcellos and K. Sakourai from Kyushu University showed that +40% of the ImageNet validation dataset can be perturbed to at least one target class by modifying a single pixel !
- Three CNNs were used for this study: AllConv, NiN and VGG.

arXiv:1710.08864v4 (22.2.18)

Chihuahua or muffin ?

Astonishingly, a fine-tuned CNN (transfer learning) works very well on the chihuahua vs muffin challenge!

Is there a horse in the image ?

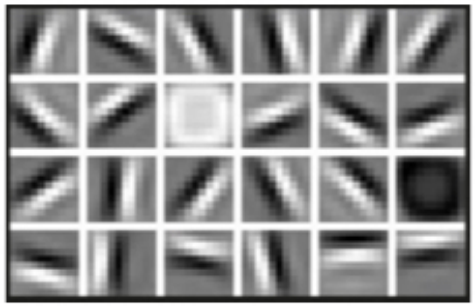


The winner solution of the Pascal VOC challenge classified images as horses using the presence of the copyright legend

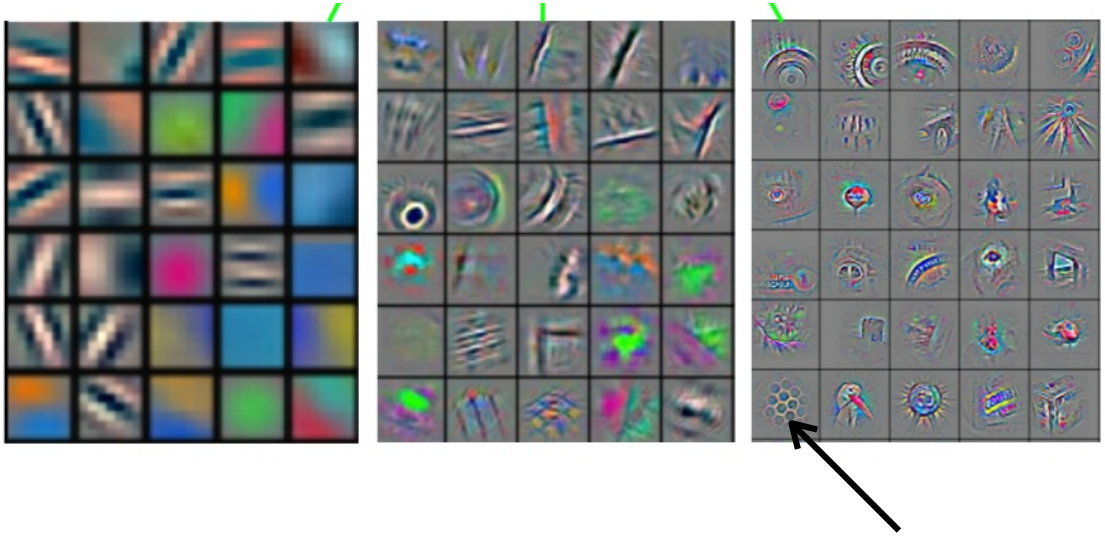
S Lapuschkin et al., 2019

Visualization tools

kernels

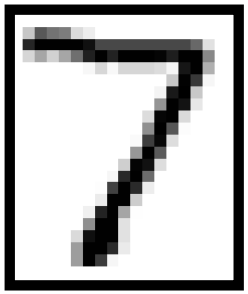


First Layer Representation | Second Layer Representation | Third Layer Representation



images that maximize the activation of each filter

MNIST digit recognition with CNNs



input (28x28)

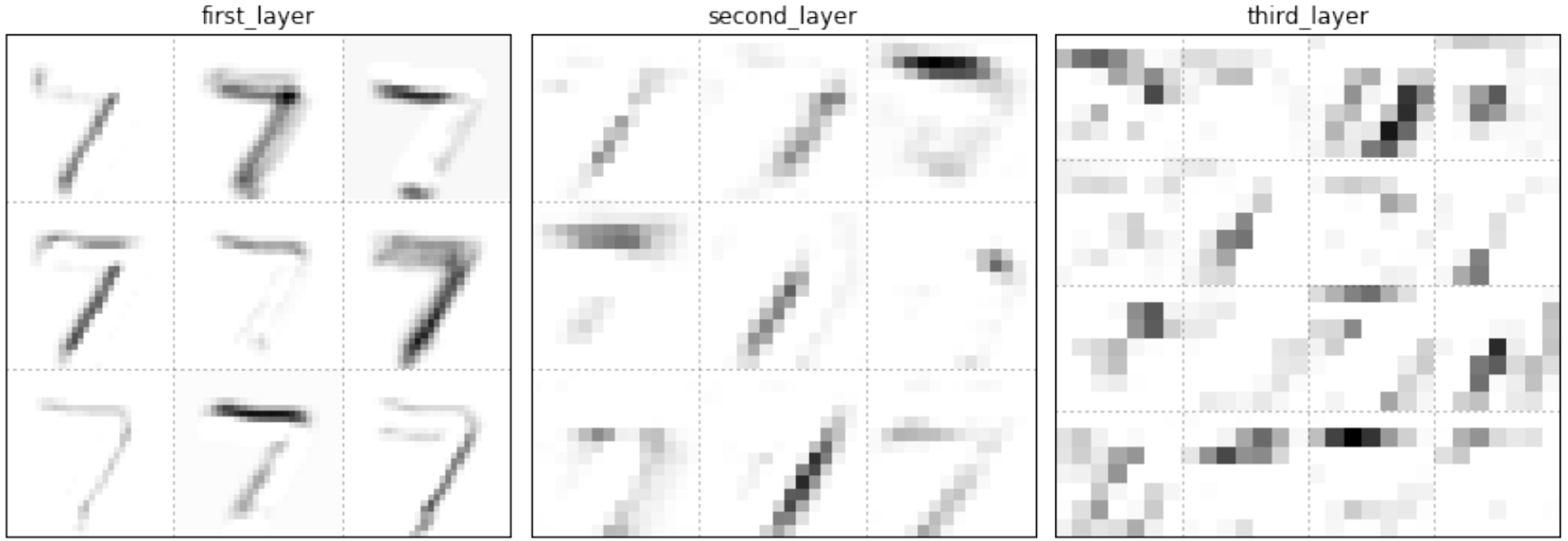
9x[5x5]

maxpool

9x[5x5]

maxpool

16x[3x3]



28x28 14x14 7x7

Outputs of the convolutional filters

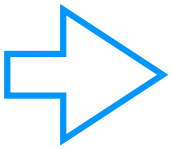
Visualization tools (1)

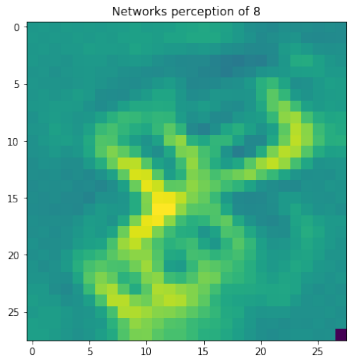
Activation maximization

- The output of the first convolutional layer is easy to interpret. Let's simply visualize it as an image.
- Subsequent convolutional filters operate over the outputs of previous filters (which indicate the presence or absence of some "templates"), making them hard to interpret.
- Idea: what sort of input patterns maximize the activation of a particular filter ?
- Use: ∂ Activation / ∂ input to modify a random input image and maximize the activation of a given output. Example:

28x28

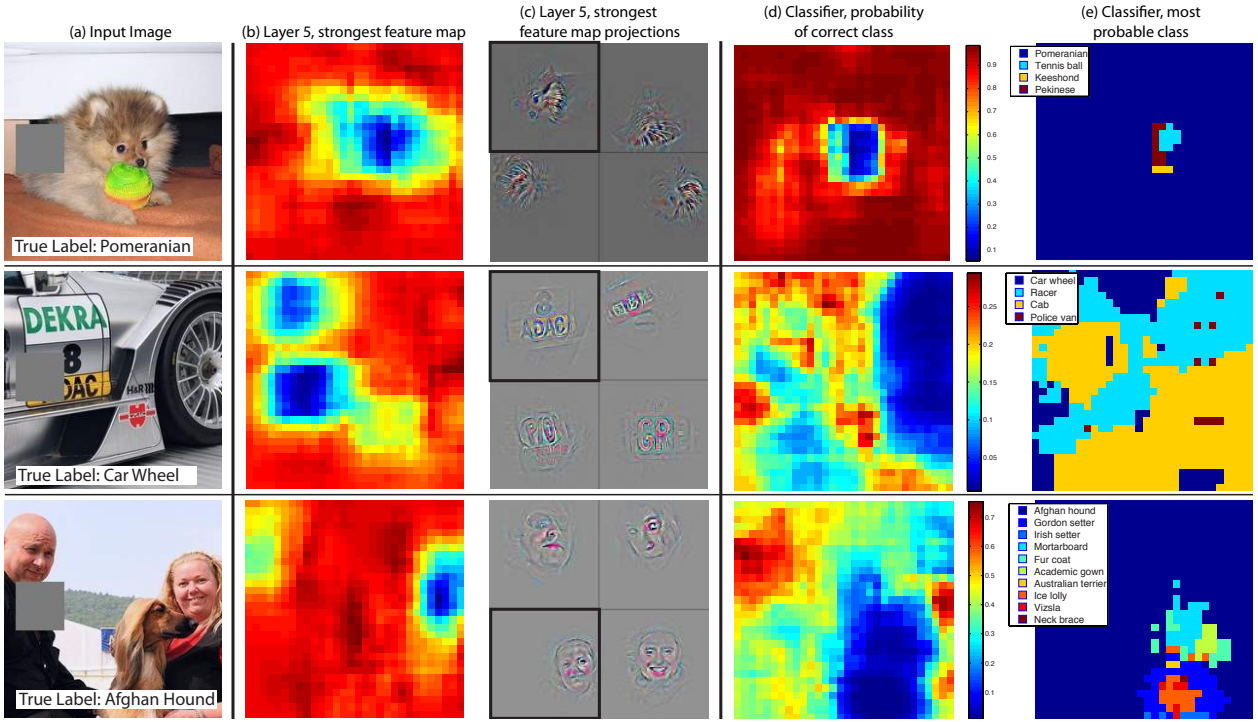
random image





Visualization tools (2)

- Occlusion analysis (Feiler & Fergus, 2013)



What is the most discriminative object or part of the image that lets the CNN decide what is the label of a given image ?

Visualization tools (3)

- Class Activation Maps (CAM)
- A Global Average Pooling (GAP) layer computes the mean of the activations of the filters of a given layer.
- We replace the fully-connected part by a GAP layer followed by a dense linear layer using a softmax output. We train this network and obtain weights w_1 to w_n.
- The CAM for a given image is the weighted sum of the last-layer's filter activations. It is finally upsampled to match the input image size.

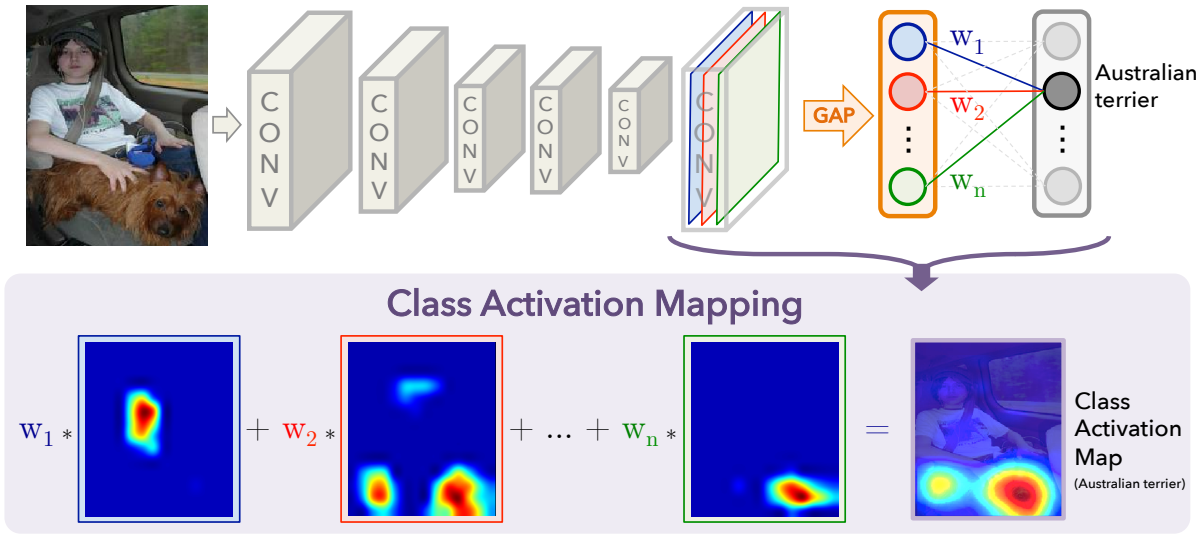
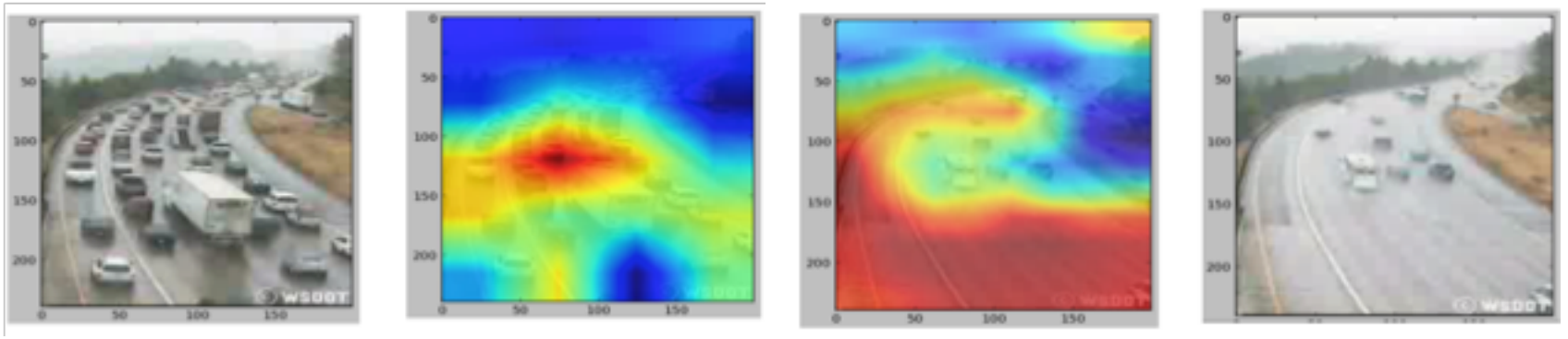


Figure 2. Class Activation Mapping: the predicted class score is mapped back to the previous convolutional layer to generate the class activation maps (CAMs). The CAM highlights the class-specific discriminative regions.

from arXiv:1512.04150v1

Example: traffic density estimation

Examples of low, medium and high traffic

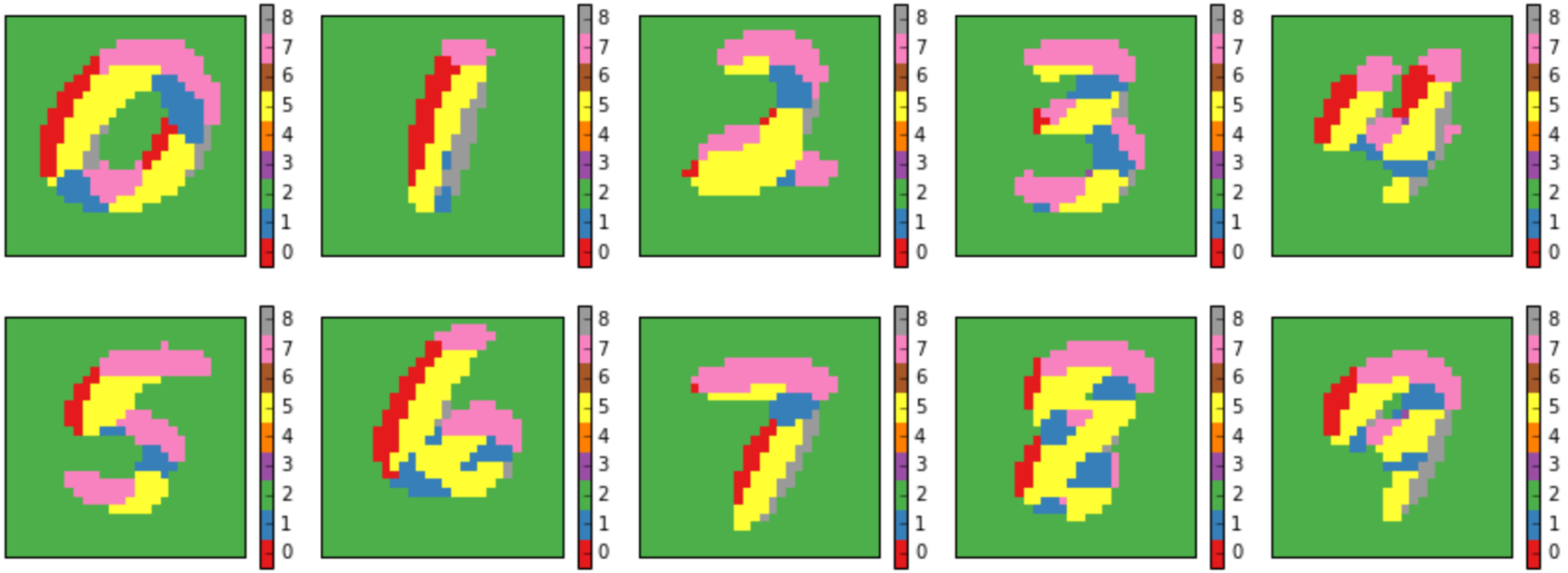


presence of many vehicles = high traffic / absence of vehicles = low traffic

from Sabbani, Perez-Uribe, Bouttane, El Moudni (2018)

Visualization tools (4)

- Filter activation statistics (Satizabal, 2016)



The red pixels indicate that for them, the kernel 0 is the most frequent filter (for all same digits in the database) with the highest activation

Microscope OpenAI

ResNet v2 50

ResNets use skip connections to enable stronger gradients in much deeper networks. This variant has 50 layers.

ResNet v2 50

1,001 predictions/Softmax
1,001 1,1 logits/Conv2D
2,048 block4/unit_3/add
3,3 block4/unit_3/conv2...
1,1
2,048 block4/unit_2/add
3,3
1,1
2,048 block4/unit_1/add
3,3
1,1
1,024 block3/unit_6/add
3,3
1,1
1,024 block3/unit_5/add
3,3
1,1
1,024 block3/unit_4/add
3,3
1,1
1,024 block3/unit_3/add
3,3

block4/unit_3/conv2/Relu

Type: Relu
Channels: 512
Convolution: [3,3]

Technique

- Feature Visualization
- DeepDream
- Dataset Samples
- Caricature
- Text Feature Visualization

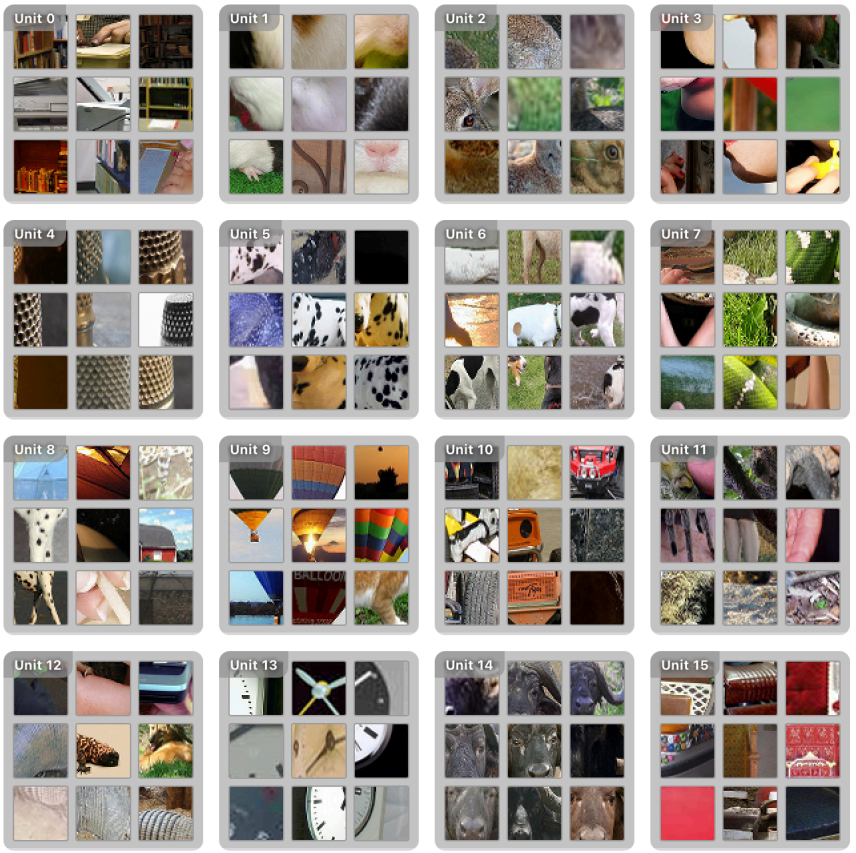
Pieces of images from the training dataset that result in the largest activations from the given unit.

View

Image Size

Resize Behavior

- Crop image
- Scale image



https://microscope.openai.com/models