Application of Machine Learning in Vision: Tools and Techniques

Shubham Sharma and Somnath Rakshit

Contents

- Common ML Tasks in Computer Vision
- The deep learning wave: traditional methods vs deep learning
- Convolutional Neural Networks: VGGNet
- Using Keras, Tensorflow or PyTorch to build/fine-tune a network
- Practical considerations
- Case study: Google Landmark Recognition Challenge

Common ML Tasks in Computer Vision

- Object **Classification**
  what object ?

http://pascallin.ecs.soton.ac.uk/challenges/VOC/

- Object **Detection**
  object or no-object ?

{people | vehicle | … intruder ….}

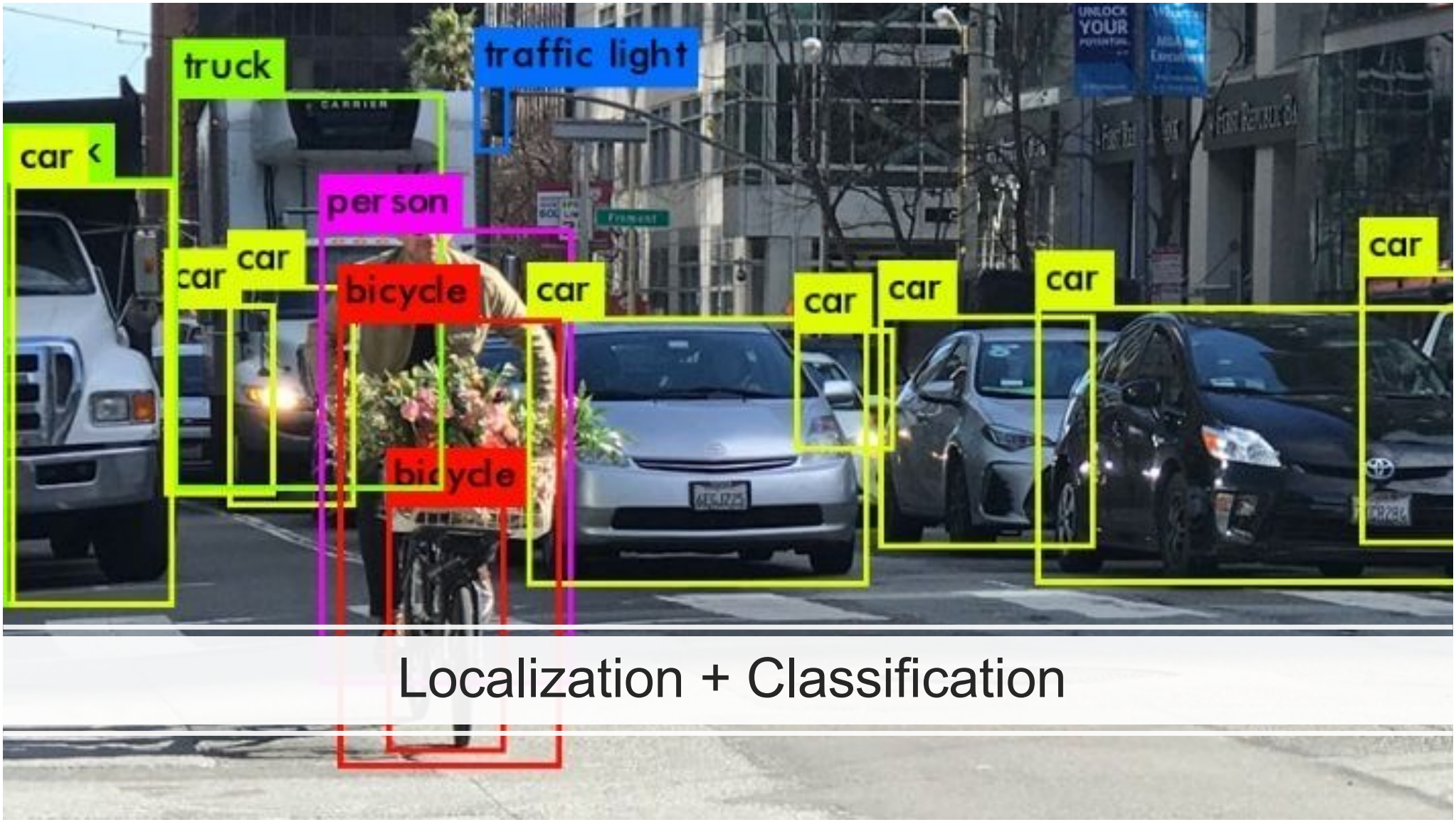
- Instance **Recognition** ?
  who (or what) is it ?

{face | vehicle plate| gait …. → biometrics}

- **Sub-category** analysis
  which object type ?

{gender | type | species | age ……}

- Sequence **{ Recognition | Classification }** ?
  what is happening / occurring ?



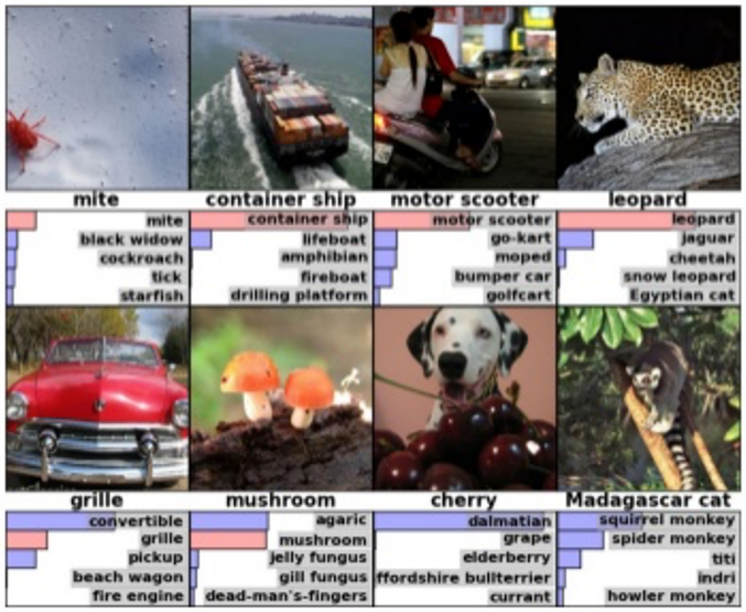
Localization + Classification

Object Classification

ImageNet Challenge

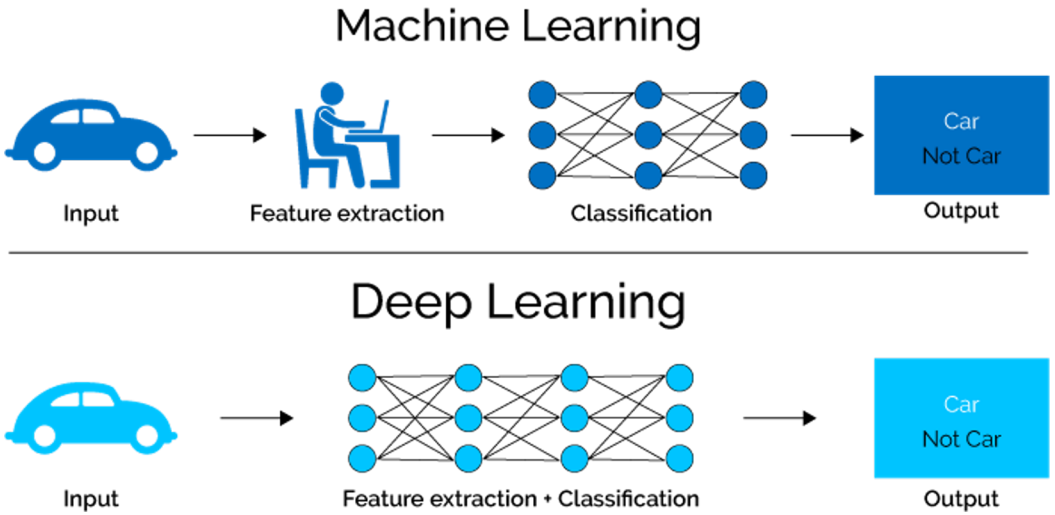
IMAGENET

- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.

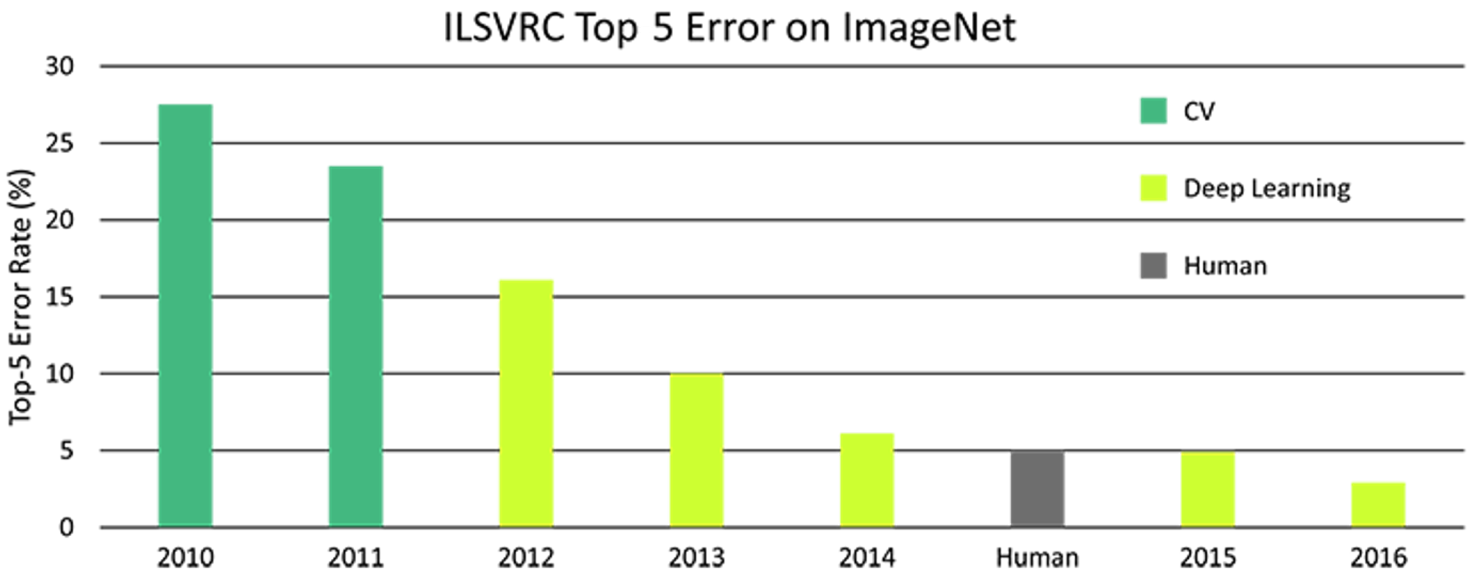


Techniques

- Traditional feature extractors: SIFT, HoG, SURF, Harris Corner Detector, Hough Transform etc.
- Tradition ML algorithm: SVM, ANN, ensemble methods etc.
- Deep Learning Algorithms: AlexNet, VGGNet, ResNet50, Inception etc.



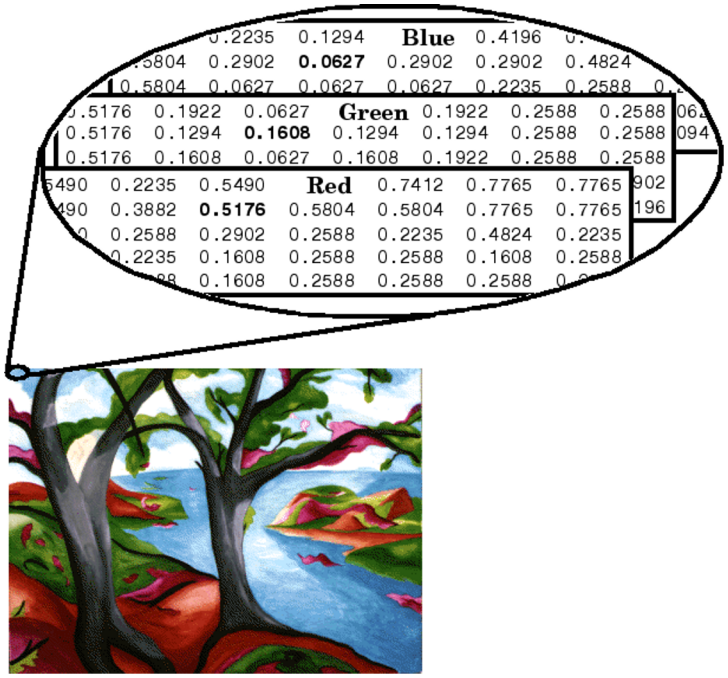
ImageNet Challenge result

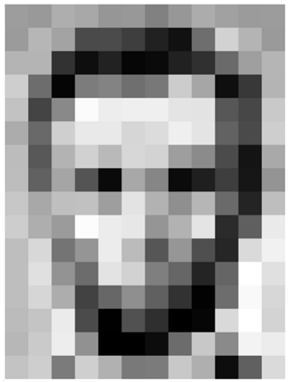


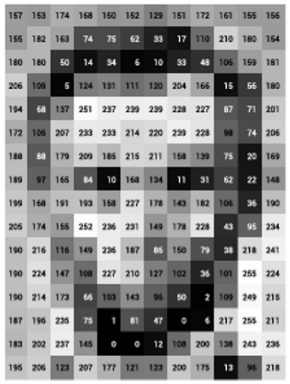
Since 2012

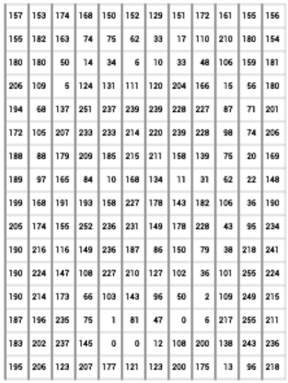

RESEARCHERS
SUPPORT VECTOR MACHINES
DEEP LEARNING
imgflip.com

Representing Images

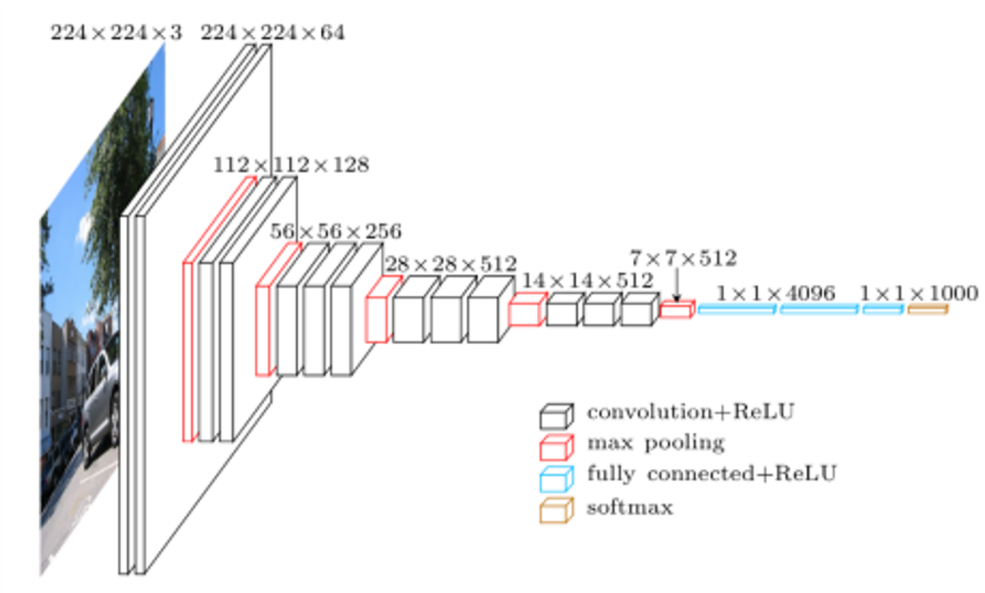








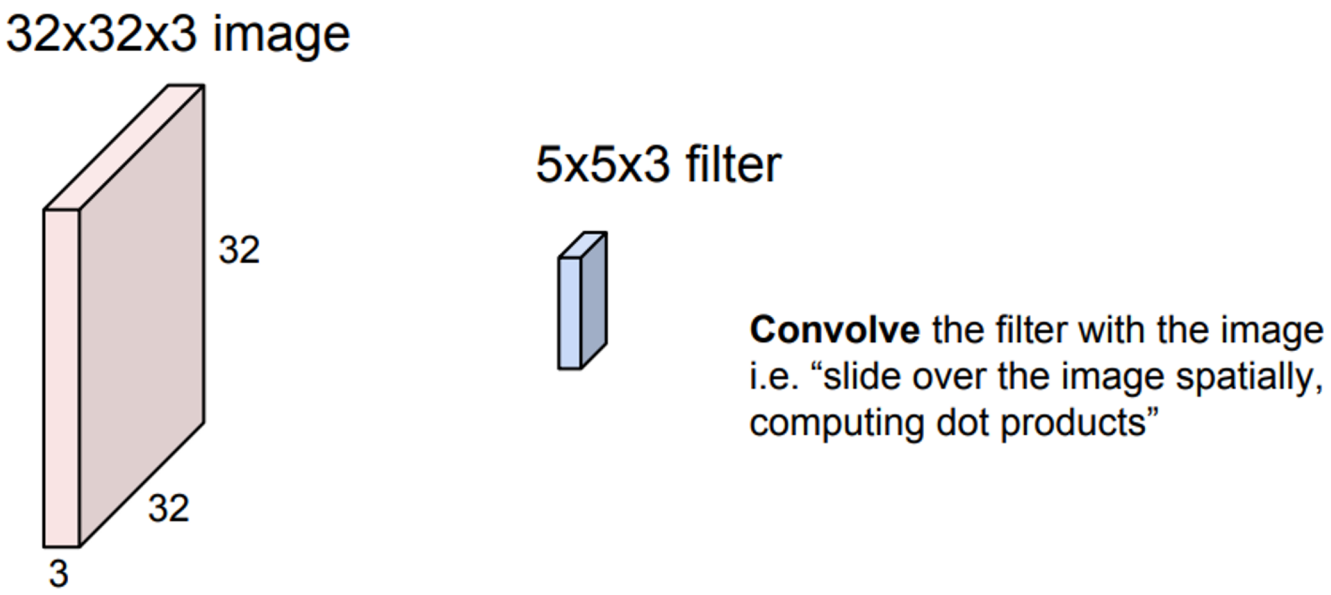
Convolutional Neural Networks



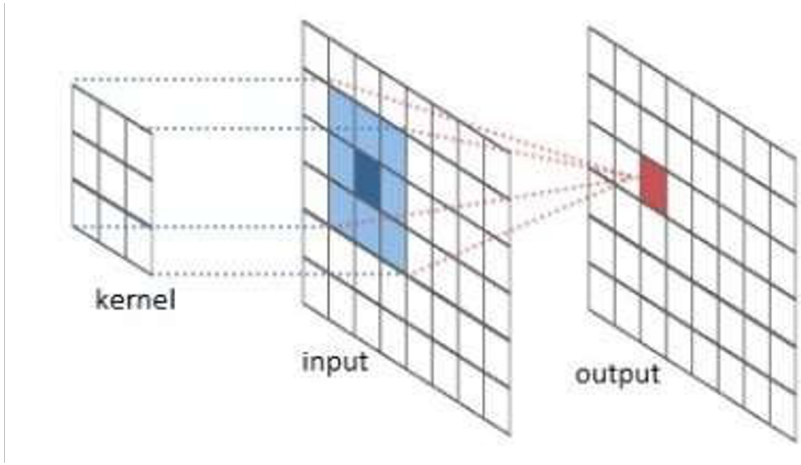
Learning by example: Breaking down VGG16

Convolution Layer

- Given an input image:

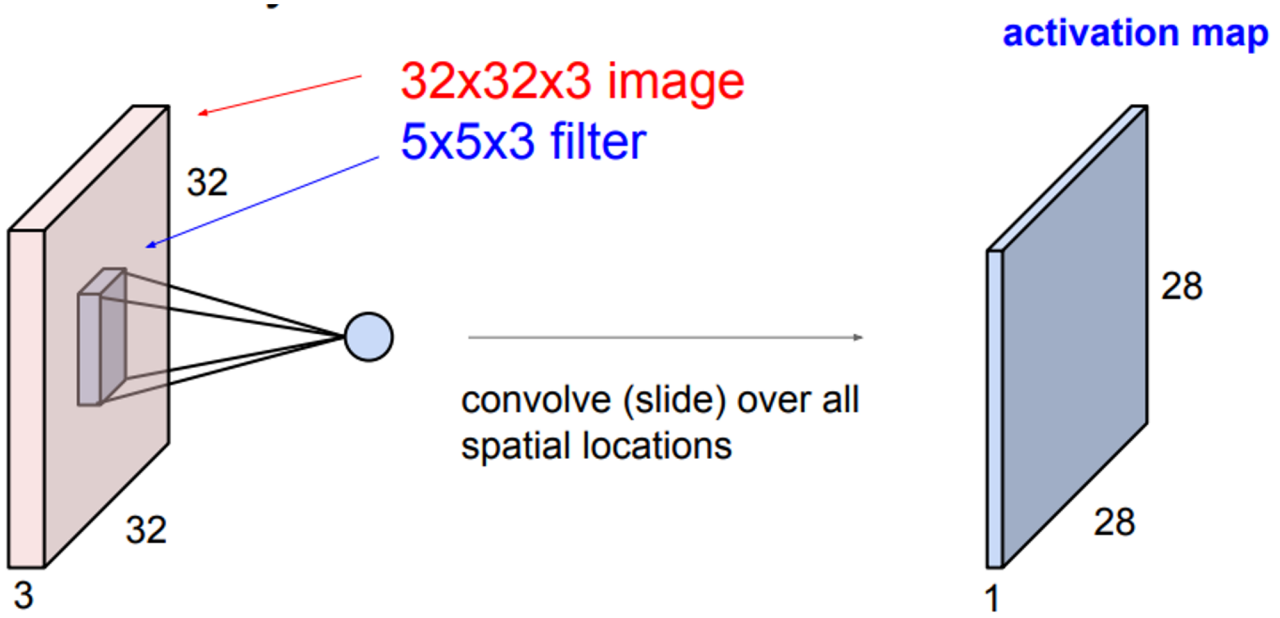


Convolution Layer

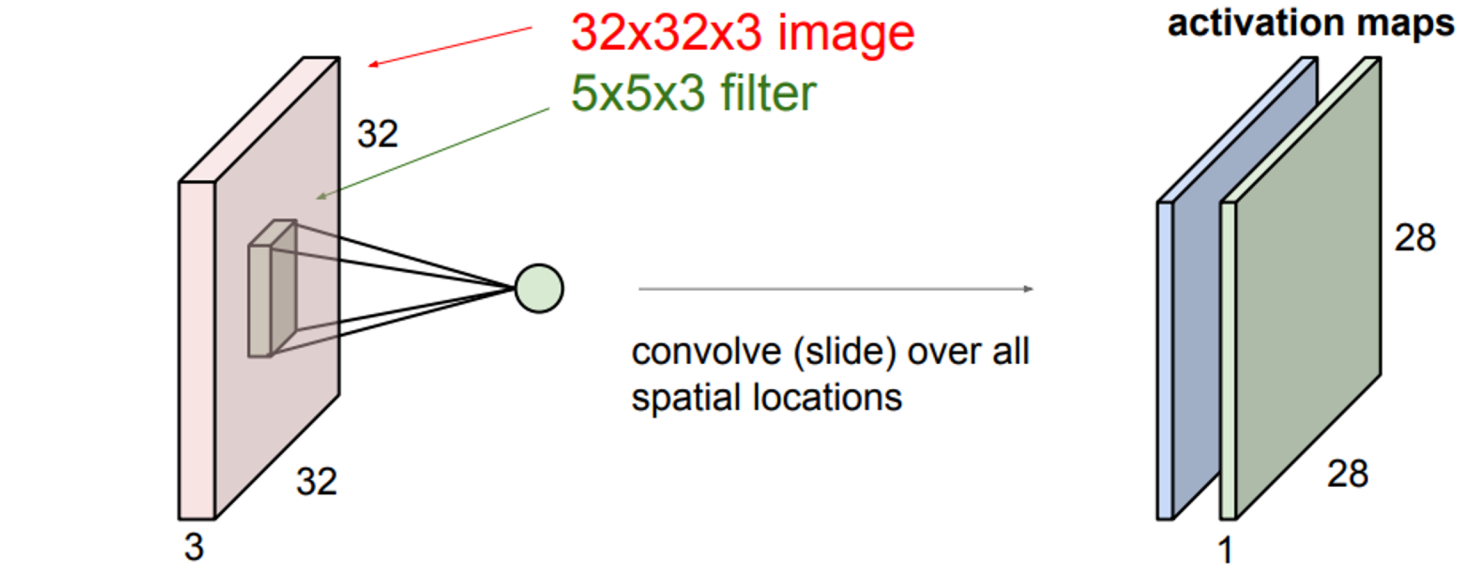


- 2-d weighted average when multiplying kernel over pixel patches.
- Slide the kernel over all pixels of the image. Each kernel starts of with random values and the network updates (learns) these values using back propagation.

Convolution Layer

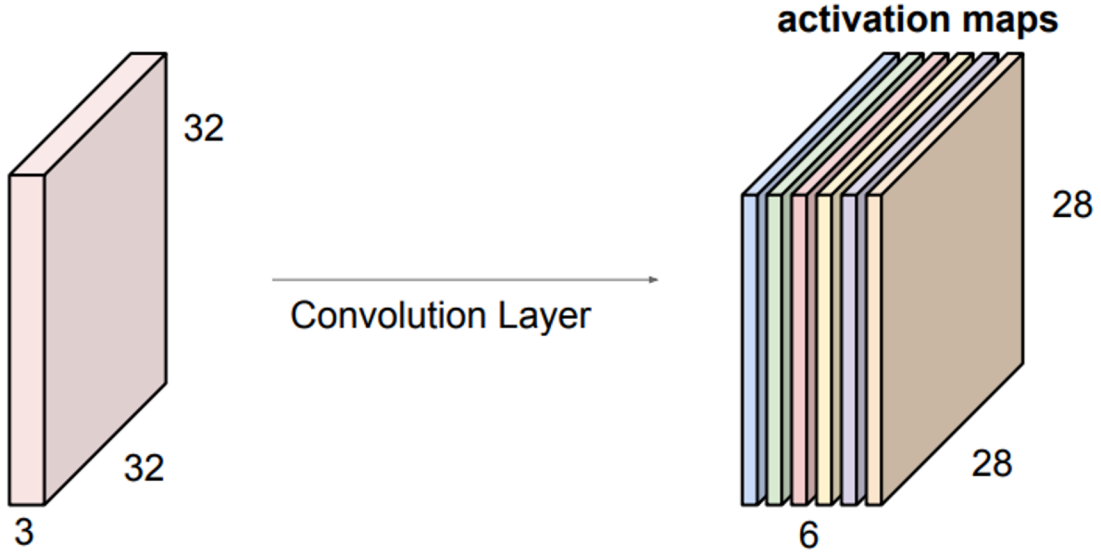


Convolution Layer



Convolution Layer

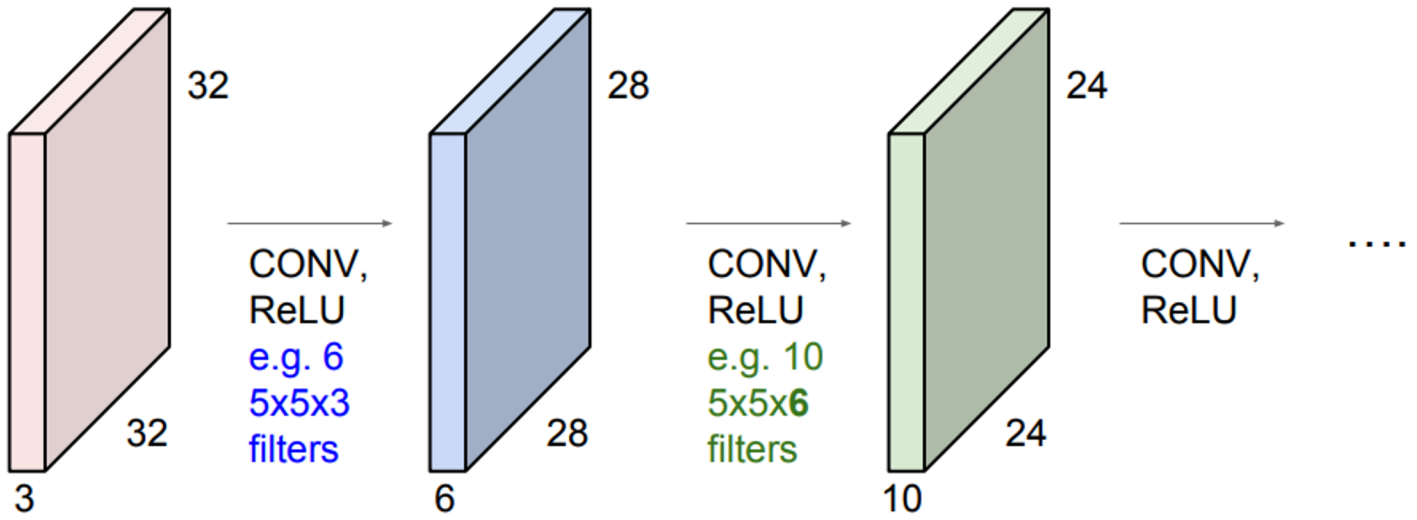
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



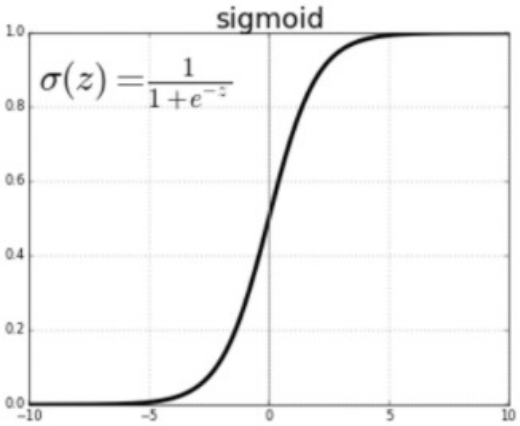
We stack these up to get a "new image" of size 28x28x6!

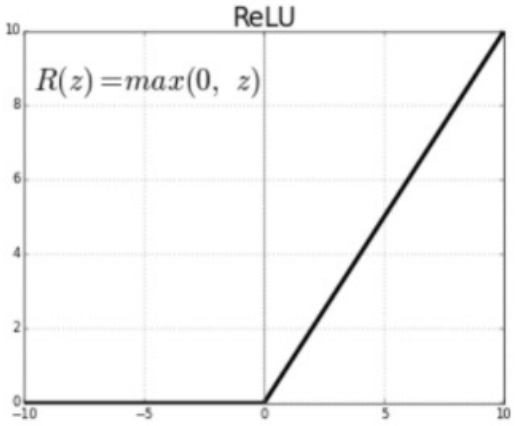
Convolution Layer

Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



Activation Functions



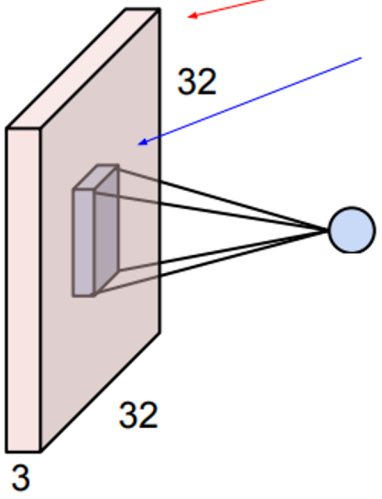


- Vanishing gradient
- Outputs not zero centered
- Computationally expensive

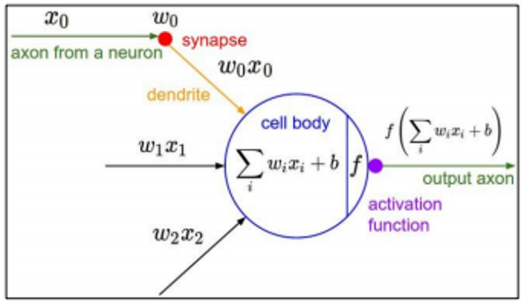
- Computationally efficient
- Dying ReLU problem
- Solved using Leaky ReLU/Parametric ReLU/Swish

Convolution Layer

The brain/neuron view of CONV Layer



32x32x3 image
5x5x3 filter



It's just a neuron with local connectivity...

1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

Convolution Layer: Parameters involved

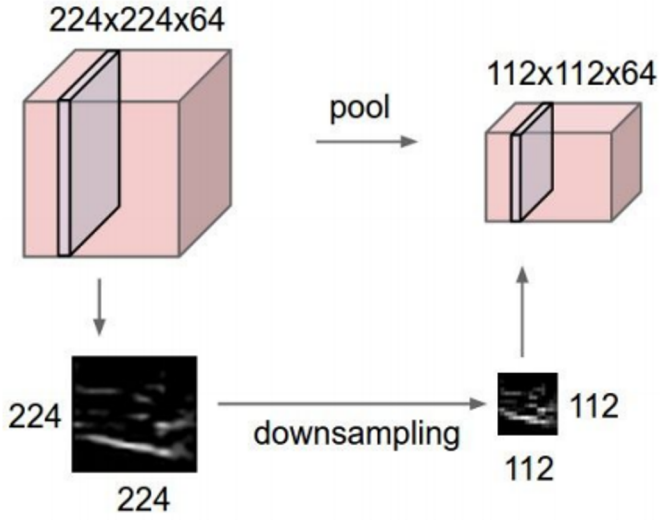
Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

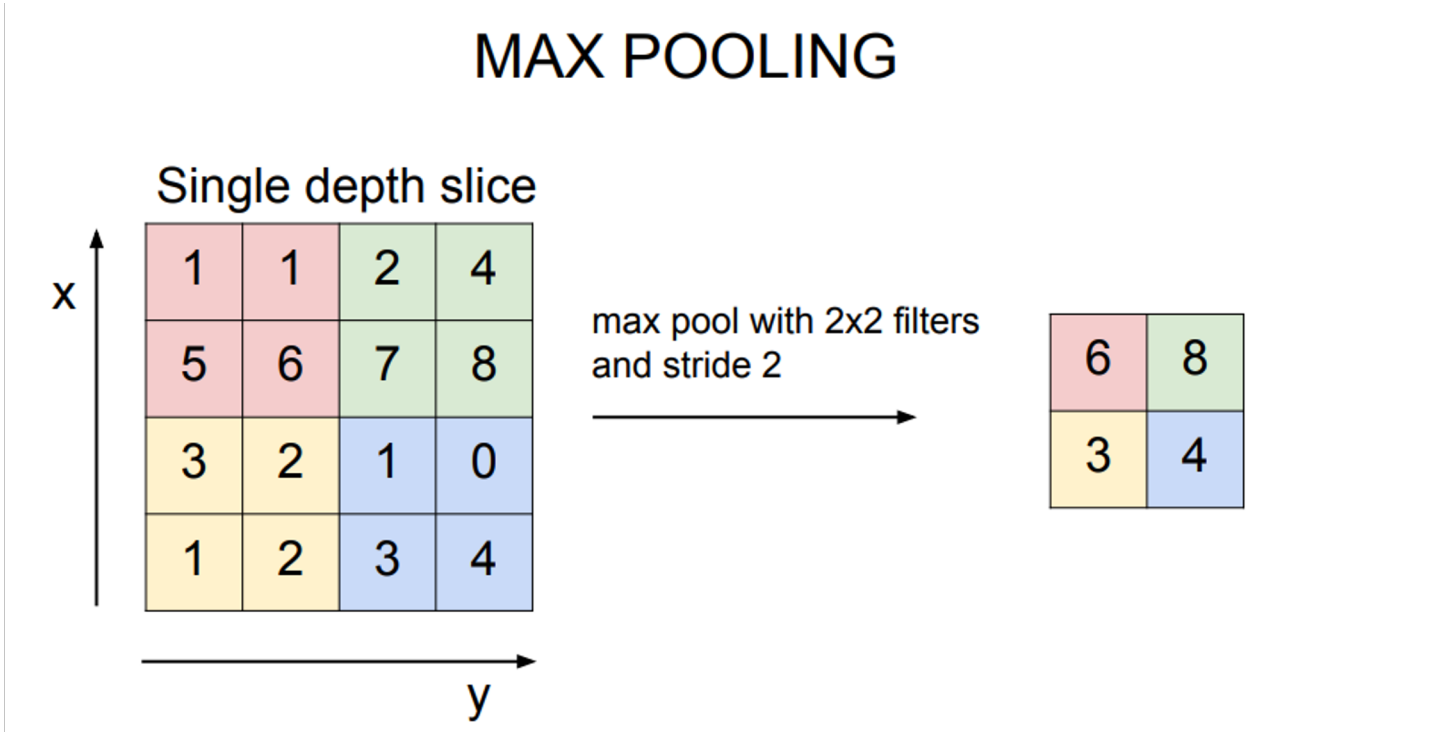
Pooling

Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:

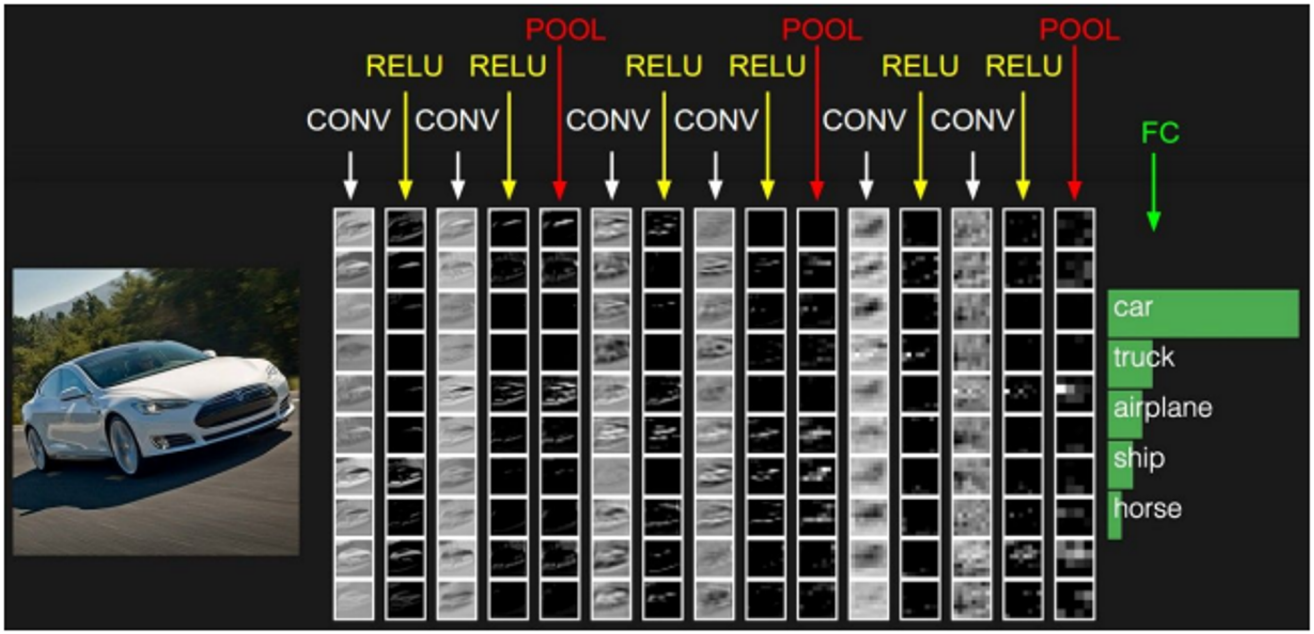


Pooling

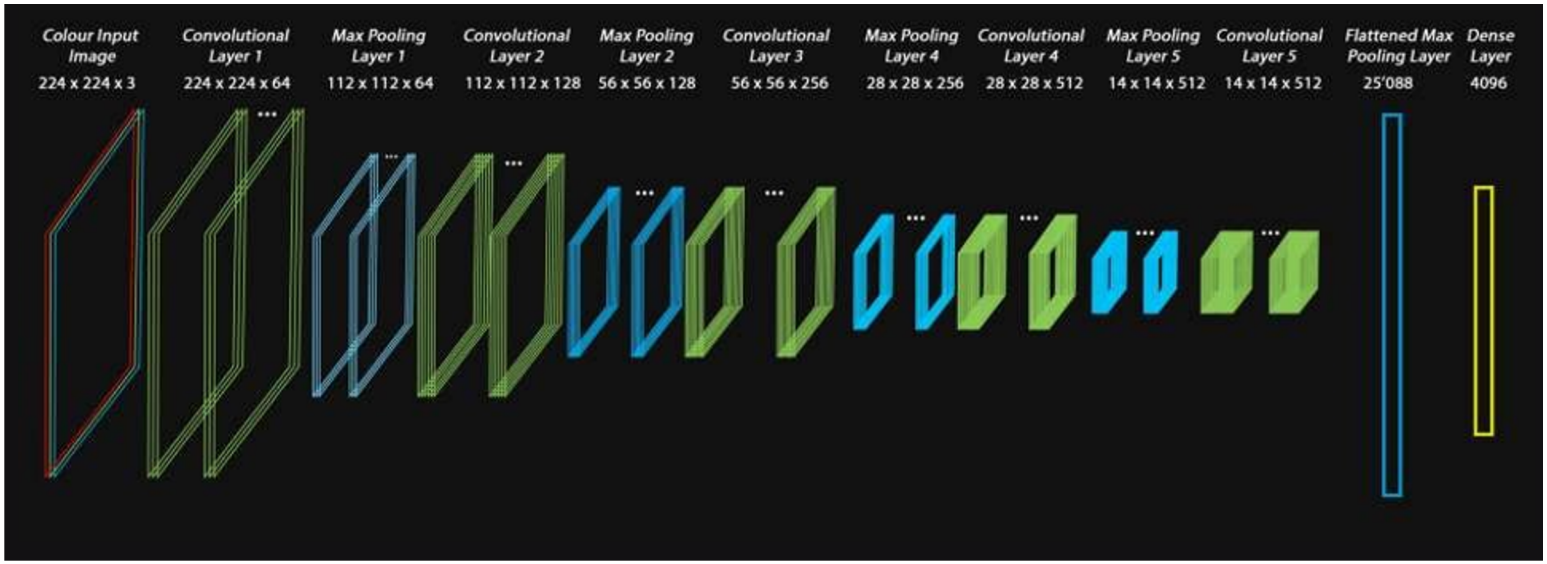

MAX POOLING
Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Fully Connected Layer

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



Bringing it all together: VGG16



Tools for deep learning

- Most commonly used: Caffe, TensorFlow, PyTorch and Keras.
- Easy to implement a network in Keras
- Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano.
- A comparison of deep learning softwares : https://en.wikipedia.org/wiki/Comparison_of_deep_learning_software

Training a deep network from scratch

- Training VGGNet from scratch on ImageNet (variant of VGG16):

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

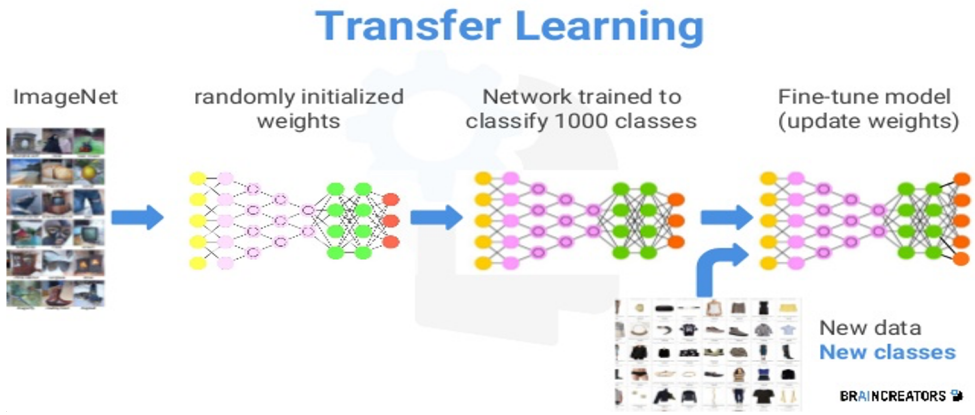
Training a deep network from scratch

With so many parameters and operations and possibly without GPU access

Fortunately, pre-trained models are available in Keras and we can use those for any other application!

Solution: Transfer Learning

- Cut off last layer or last few layers of pre-trained ImageNet winning CNN
- Keep learned network (convolutions) but replace final layer
- Can learn to predict new (completely different) classes
- Re-train a new final layer for a new task



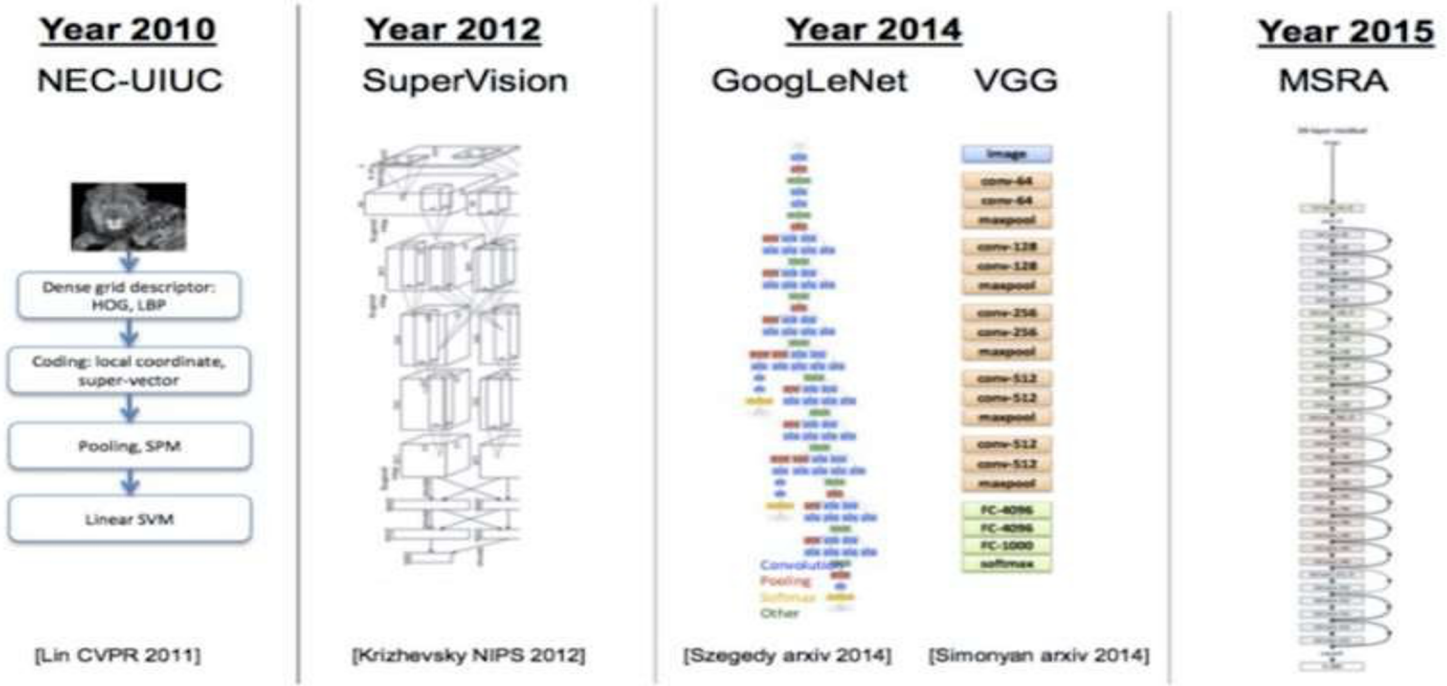
VGG16 in Keras: using a pre-trained model

```
 1 from keras.applications.vgg16 import VGG16
 2 from keras.preprocessing import image
 3 from keras.applications.vgg16 import preprocess_input
 4 import numpy as np
 5
 6 model = VGG16(weights='imagenet', include_top=False)
 7
 8 img_path = 'elephant.jpg'
 9 img = image.load_img(img_path, target_size=(224, 224))
10 x = image.img_to_array(img)
11 x = np.expand_dims(x, axis=0)
12 x = preprocess_input(x)
13
14 preds = model.predict(x)
15 print('Predicted:', decode_predictions(preds, top=3)[0])
16 # Predicted: [(u'n02504013', u'Indian_elephant', 0.82658225), (u'n01871265', u'tusker'
```

Transfer learning in Keras

- Fix weights in convolutional layers (set trainable=False) for these weights.
- Remove final dense layer that predicts 1000 labels and add a new layer or set of layers with a size of the labels you want to predict.
- Now train the model to learn the weights for these new layers based on your new data and classes, utilizing weights from a pre-trained model.
- Example: https://gist.github.com/fchollet/7eb39b44eb9e16e59632d25fb3119975

Advanced methods: Could use an ensemble of many models



Google

Case Study: Google Landmark Recognition Challenge

Problem Statement

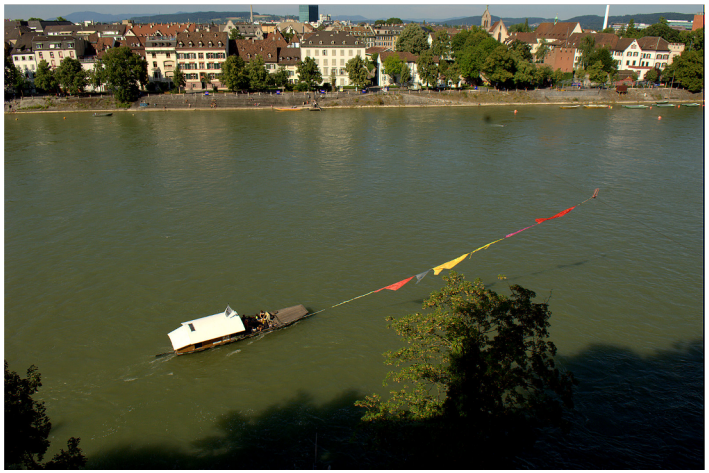
Recognize a landmark (if any) from a dataset of over a million images with around 13000 landmark classes.

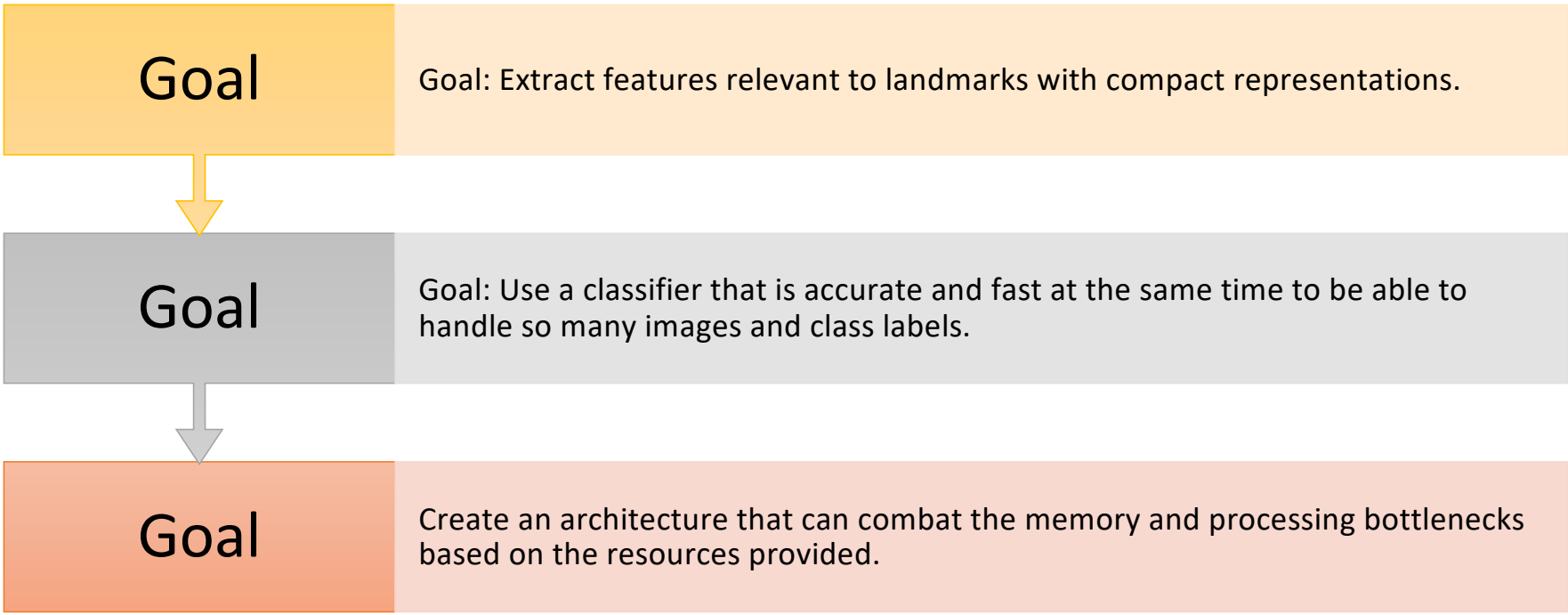
Examples

Variation in images for the same landmark

Challenges in data

Challenges in data





Project Goals

Example Image Results

Original Image

Feature descriptor+ ML model

Deep Learner

Example Image Results

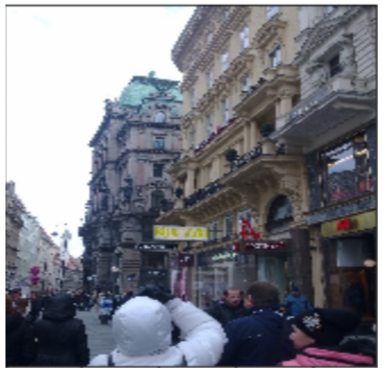
Original Image

Feature Descriptor+ML model

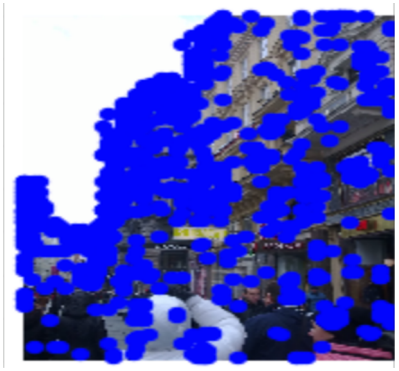
Deep Learner

Example Image Results

Original Image



Feature Descriptor + ML model



Deep Learner

Example Image Results: Both models are incorrect

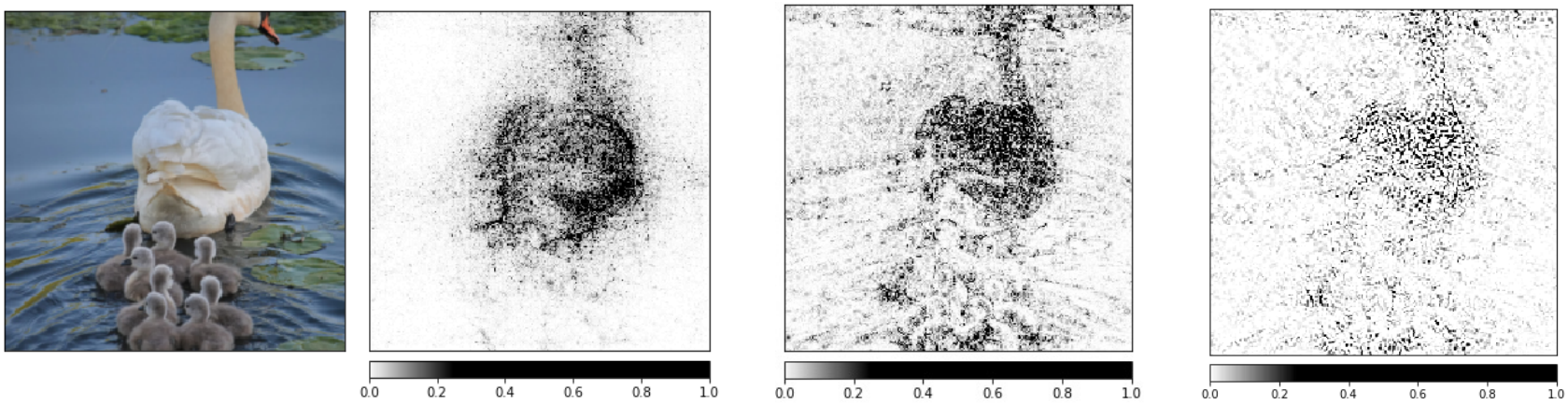
Original Image

Feature Descriptor + ML model

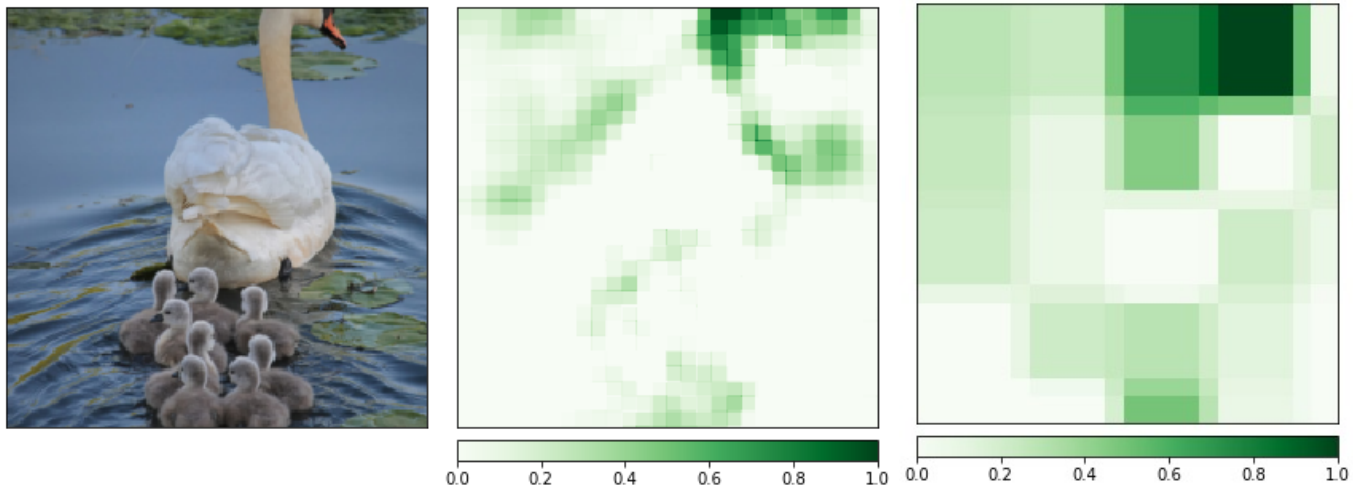
Deep Learner

Model interpretability

Gradient based attribution



Occlusion based attribution



Model interpretability (Paper)

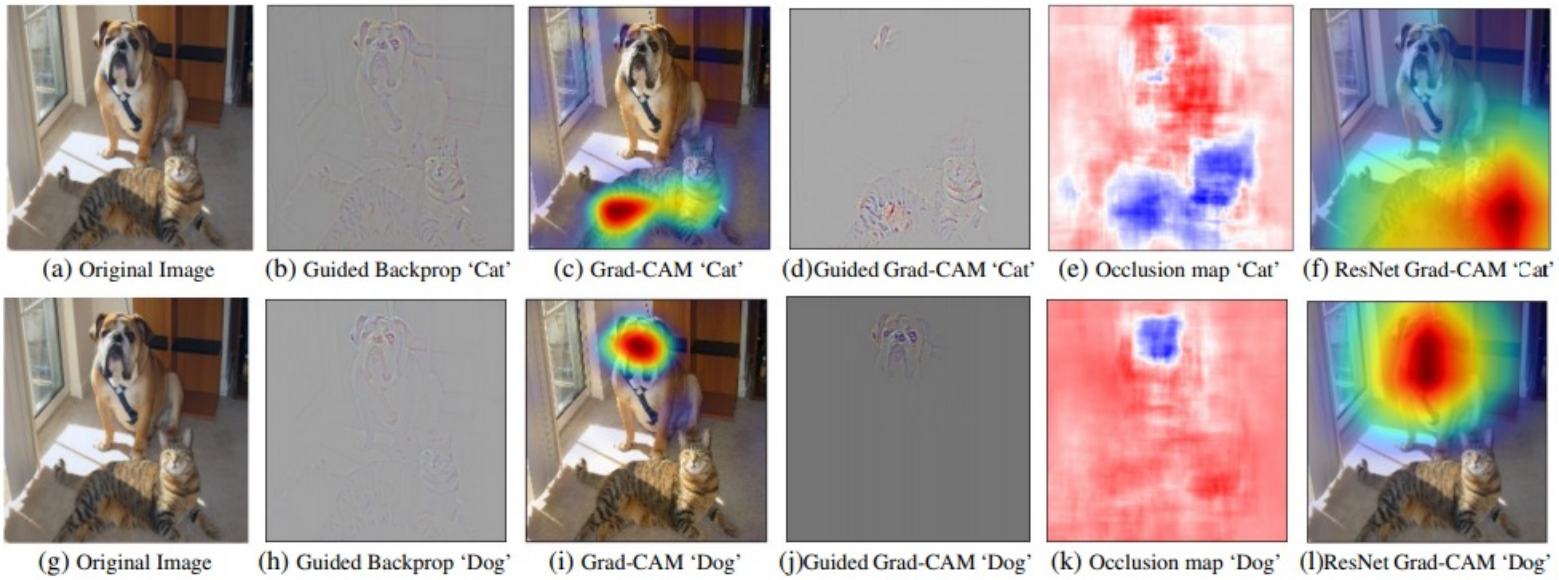


Fig. 1: (a) Original image with a cat and a dog. (b-f) Support for the cat category according to various visualizations for VGG-16 and ResNet. (b) Guided Backpropagation [53]: highlights all contributing features. (c, f) Grad-CAM (Ours): localizes class-discriminative regions, (d) Combining (b) and (c) gives Guided Grad-CAM, which gives high-resolution class-discriminative visualizations. Interestingly, the localizations achieved by our Grad-CAM technique, (c) are very similar to results from occlusion sensitivity (e), while being orders of magnitude cheaper to compute. (f, l) are Grad-CAM visualizations for ResNet-18 layer. Note that in (c, f, i, l), red regions corresponds to high score for class, while in (e, k), blue corresponds to evidence for the class. Figure best viewed in color.

Model interpretability (Details)

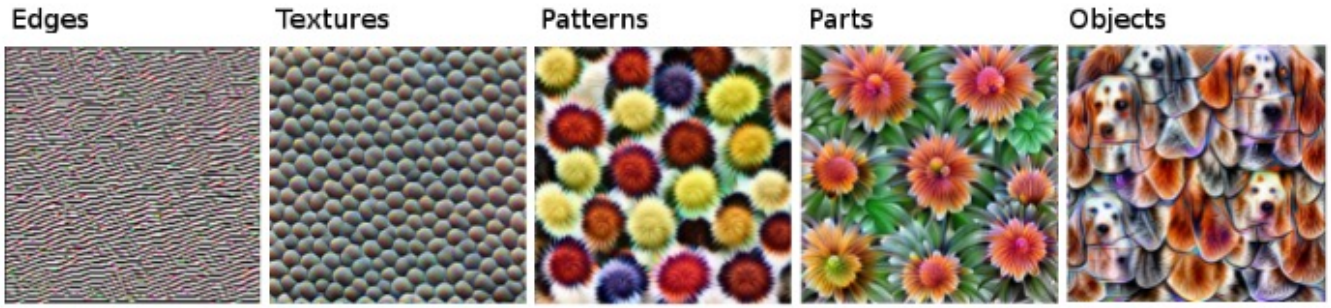
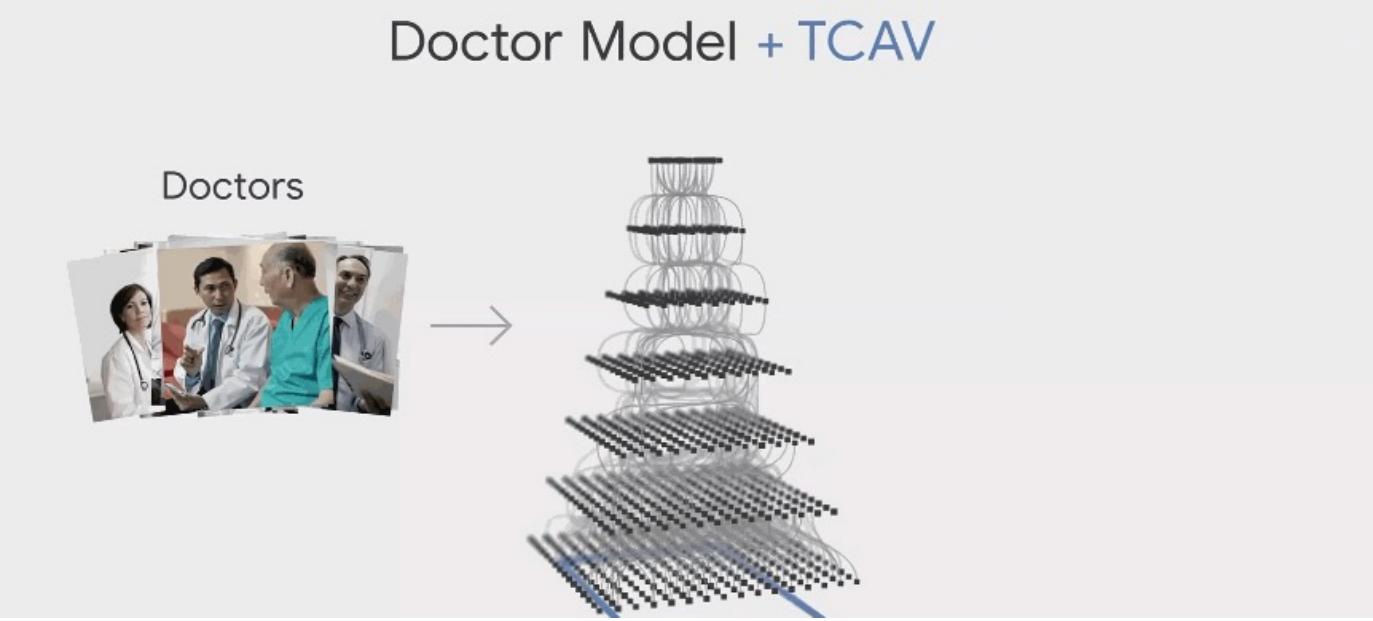


FIGURE 7.1: Features learned by a convolutional neural network (Inception V1) trained on the ImageNet data. The features range from simple features in the lower convolutional layers (left) to more abstract features in the higher convolutional layers (right). Figure from Olah, et al. 2017 (CC-BY 4.0) https://distill.pub/2017/feature-visualization/appendix/.

Model interpretability (Google TCAV Paper, Slides, Video)



Moving from Scikit Learn

Scikit Learn

- Sklearn - Implementation of standard machine learning models. Limited flexibility in changing models.
- Lesser parallel processing - too slow for deep networks.

Deep Learning libraries

- Array manipulation libraries with automatic differentiation.
- Optimized parallelisation for running on multiple cores of CPU or GPUs.

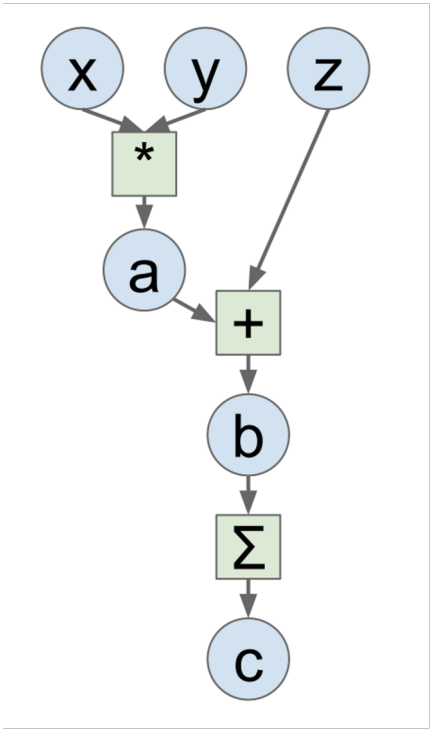
Auto-Differentiation in various packages (Code)

- There are many modern libraries that can do auto-differentiation. You should use them wherever you can to make life easier.
- In the next few slides, we will see how libraries like Tensorflow and Pytorch can be used to do this, as compared to Numpy, which does not have this capability.
- Here, we generate 3 matrices that will be used throughout this demo.

```
import numpy as np

np.random.seed(0)

N, D = 3, 4

p  = np.random.randn(N, D)
q = np.random.randn(N, D)
r = np.random.randn(N, D)

print(f'{p}\n\n{q}\n\n{r}')
```

```
[[ 1.76405235  0.40015721  0.97873798  2.2408932 ]
 [ 1.86755799 -0.97727788  0.95008842 -0.15135721]
 [-0.10321885  0.4105985   0.14404357  1.45427351]]

[[ 0.76103773  0.12167502  0.44386323  0.33367433]
 [ 1.49407907 -0.20515826  0.3130677  -0.85409574]
 [-2.55298982  0.6536186   0.8644362  -0.74216502]]

[[ 2.26975462 -1.45436567  0.04575852 -0.18718385]
 [ 1.53277921  1.46935877  0.15494743  0.37816252]
 [-0.88778575 -1.98079647 -0.34791215  0.15634897]]
```

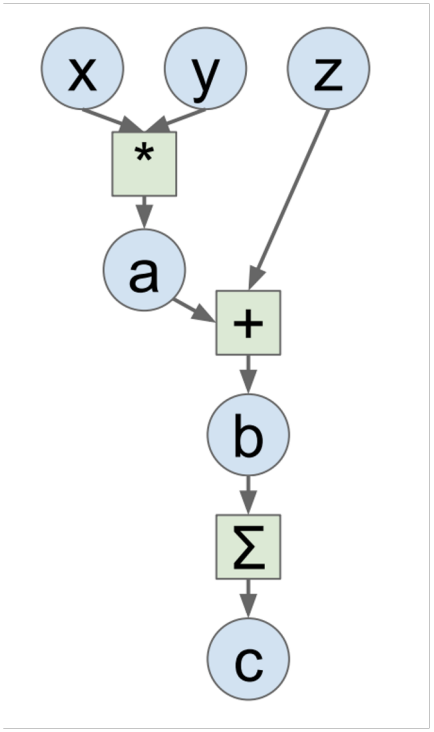
Computational graphs - Numpy




```
x = p
y = q
z = r

a = x*y
b = a+z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c*np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a*y
grad_y = grad_a*x

print(f'{grad_x}\n\n{grad_y}\n\n{grad_z}')
```

```
[[ 0.76103773  0.12167502  0.44386323  0.33367433]
 [ 1.49407907 -0.20515826  0.3130677  -0.85409574]
 [-2.55298982  0.6536186   0.8644362  -0.74216502]]

[[ 1.76405235  0.40015721  0.97873798  2.2408932 ]
 [ 1.86755799 -0.97727788  0.95008842 -0.15135721]
 [-0.10321885  0.4105985   0.14404357  1.45427351]]

[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

Computational graphs - Tensorflow

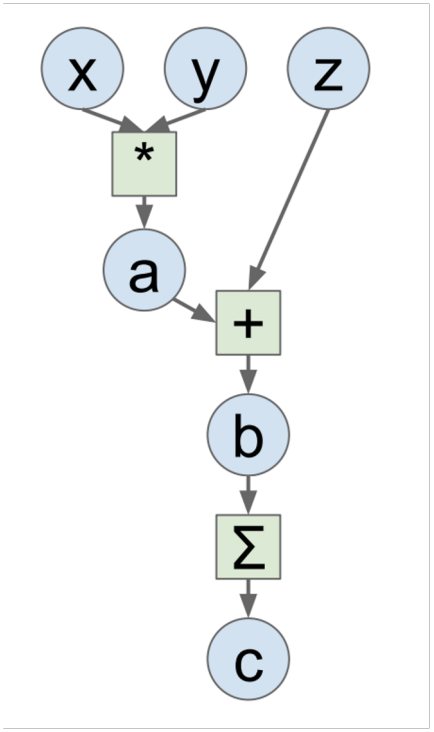



```
import tensorflow as tf

x = tf.Variable(p)
y = tf.Variable(q)
z = tf.Variable(r)

with tf.GradientTape(persistent=True) as tape:
    a = x*y
    b = a+z
    c = tf.reduce_sum(b)

grad_x = tape.gradient(c, x)
grad_y = tape.gradient(c, y)
grad_z = tape.gradient(c, z)

print(f'{grad_x}\n\n{grad_y}\n\n{grad_z}')
```

```
[[ 0.76103773  0.12167502  0.44386323  0.33367433]
 [ 1.49407907 -0.20515826  0.3130677  -0.85409574]
 [-2.55298982  0.6536186   0.8644362  -0.74216502]]

[[ 1.76405235  0.40015721  0.97873798  2.2408932 ]
 [ 1.86755799 -0.97727788  0.95008842 -0.15135721]
 [-0.10321885  0.4105985   0.14404357  1.45427351]]

[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

Example

Computational graphs - PyTorch




```
import torch

x = torch.tensor(p, requires_grad=True)
y = torch.tensor(q, requires_grad=True)
z = torch.tensor(r, requires_grad=True)

a = x*y
b = a+z
c = b.sum()

grad_x = torch.autograd.grad(outputs=c, inputs=x, retain_graph=True)[0]
grad_y = torch.autograd.grad(outputs=c, inputs=y, retain_graph=True)[0]
grad_z = torch.autograd.grad(outputs=c, inputs=z)[0]

print(f'{grad_x}\n\n{grad_y}\n\n{grad_z}')
```

```
tensor([[ 0.7610,  0.1217,  0.4439,  0.3337],
        [ 1.4941, -0.2052,  0.3131, -0.8541],
        [-2.5530,  0.6536,  0.8644, -0.7422]], dtype=torch.float64)

tensor([[ 1.7641,  0.4002,  0.9787,  2.2409],
        [ 1.8676, -0.9773,  0.9501, -0.1514],
        [-0.1032,  0.4106,  0.1440,  1.4543]], dtype=torch.float64)

tensor([[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]], dtype=torch.float64)
```

Example

Tensorflow vs PyTorch

Tensorflow

- Static graph (past), dynamic graph (present).
- Less Pythonic.
- Difficult to debug in static mode.
- Preferred for deploying models.

PyTorch

- Dynamic graph.
- Tightly integrated with python - all Pytorch models are python classes.
- Can directly print the tensor - easier to debug.
- Preferred for fast experimentation.

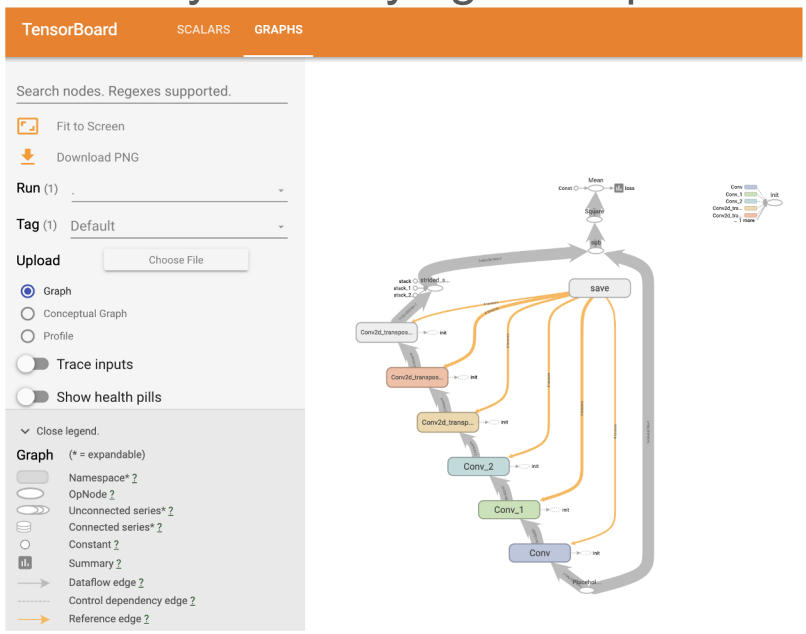
Both libraries are aggressively adapting each other's features since the past few years.

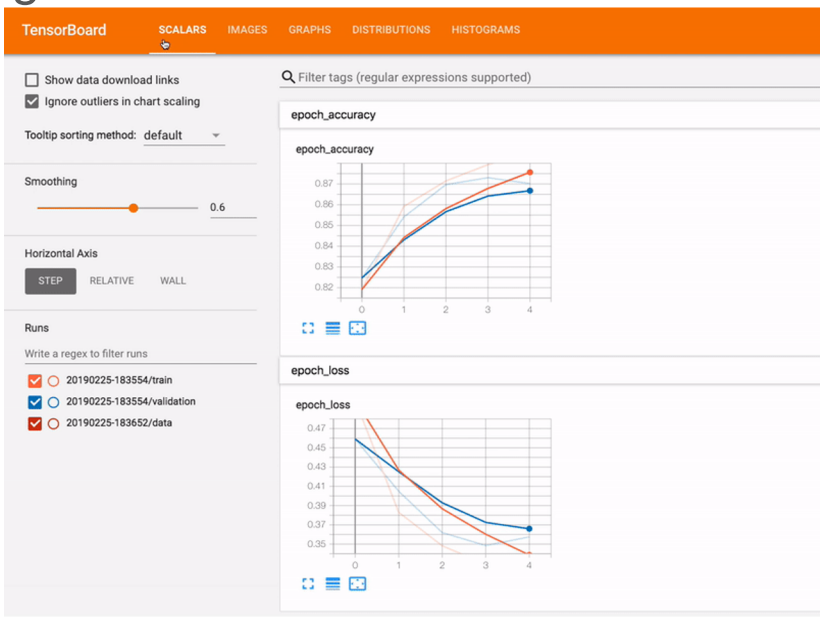
Blog comparing speed of the two frameworks

Debugging

Use Tensorboard

- You can visualize the tensor graph and track how values such as loss and accuracy are varying with epochs using Tensorboard.





- Tensorboard tutorial: https://www.tensorflow.org/tensorboard/get_started
- Use TensorboardX with PyTorch - https://github.com/lanpa/tensorboardX

Baby steps

- Ensure that no component is disconnected from rest of the graph.
- Do not start training on your entire data - takes too much time and it would be wasted if some simple bug is there in your code.
- Train on 10 data points - make sure you get 100% accuracy on these train examples before starting to train on the entire dataset.
- Save the model after every few epochs and when the validation performance is the best so far.

Tensorflow

```
# Save the weights
model.save_weights('./checkpoints/my_checkpoint')

# Create a new model instance
model = create_model()

# Restore the weights
model.load_weights('./checkpoints/my_checkpoint')
```

PyTorch

```
torch.save(model, PATH)
model = torch.load(PATH
```

Other tips

- Optimizer: Adam - the go to optimizer. Generally, initial learning rate of 0.01 works!
- You should decay the learning rate if you are using SGD based optimizers.
- Batch size - higher batch size results in more stable gradients (32 to 256)
- If your model is overfitting - use dropout (10% to 40%).
- If your model is underfitting - add more parameters!
- Hyperparameter tuning - do it systematically, save the models in different folders and note down all the hyperparameters for that model.
- Watch this video lecture for many more tricks.

Summary

- Machine learning tools can be used for computer vision tasks: recent deep learning techniques provide the best accuracy.
- While a fine-tuned deep learner can be used to generate some reasonable accuracy, it is important to choose the kind of pre-trained model and the approach you take based on your input data.
- Time and memory bottlenecks can be handled by parallelizing the network, using simpler models and if nothing else works, considering a subset of the dataset for training.
- Using visual inspection tools is crucial to understanding how your model is learning for the task.

Resources

- https://github.com/haofanwang/Awesome-Computer-Vision: List containing lots of resources to CV in general as well as in specific areas.
- https://forums.fast.ai/t/share-you-work-here-highlights/57140: Interesting projects in CV as well as in other topics of deep learning
- https://lilianweng.github.io/lil-log/2017/12/31/object-recognition-for-dummies-part-3.html: Tutorial regarding object detection models

References: Tools and Techniques

- http://cs231n.stanford.edu/ : Course to understand convnets
- https://www.tensorflow.org/tutorials/images/transfer_learning: Tensorflow implementation of fine-tuned MobileNet V2.
- http://openaccess.thecvf.com/content_ICCV_2017/papers/Noh_Large-Scale_Image_Retrieval_ICCV_2017_paper.pdf : DeLF for landmark challenge
- https://keras.io/getting_started/intro_to_keras_for_engineers/: Introductory Keras tutorial for deep learning
- https://www.pyimagesearch.com/2018/09/10/keras-tutorial-how-to-get-started-with-keras-deep-learning-and-python/ : Another introductory Keras tutorial
- https://keras.io/examples/ : Code examples for using Keras to solve various type of ML problems.
- http://www.image-net.org/ : Details on the ImageNet Challenge