ECE5554 – Computer Vision
Lecture 11b – Deep Learning

Creed Jones, PhD

Today's Objectives

- Learning Systems
  - Concepts
- Artificial Neural Networks
  - Architecture
  - Training
  - Generalization

Deep Learning Systems

- What's the difference?
- CNNs for image processing
  - Convolutional layer
  - Pooling layer
  - Activation
- Implementation
- Uses

A learning system is one whose input-output response is determined by a collection of historical data

- Learning assumes some underlying pattern of behavior
  - The input-output relationship has some consistency
  - Not necessarily a parametric model
- Learning assumes stability of the process
  - If future conditions are unlike the training data in important ways, then the learning system will have poor performance
- Learning can ease the burden of system design
  - Often, a learning component replaces a complex analysis step

Machine Learning discovers underlying patterns in the data (at training time) and generates rules that are applied (at run-time)

Most systems implement *supervised learning*, where we provide a labeled data set

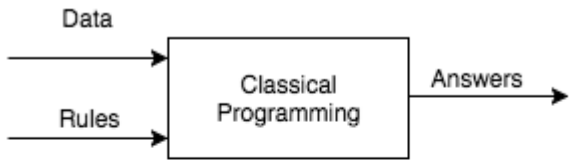
- For each instance in the data set, we assign a desired output (class name, good/bad, output value, etc.)

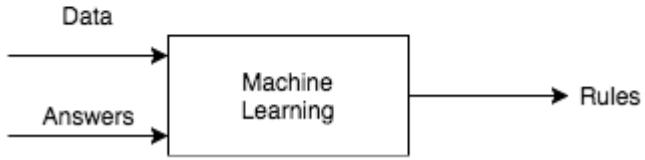
The ML component derives rules to generate the desired outputs from the input data

- This is training

We then apply the rules to new data and hope that the calculated outputs are useful

- This is run-time





Occam's Razor – *"More things should not be used than are necessary"*

- William of Ockham studied logic in the 14th century
- It's stated in many forms:
  - "Nature operates in the shortest way possible."
  - "Whenever possible, substitute constructions out of known entities for inferences to unknown entities."
  - "When you hear hoofbeats, think of horses not zebras."

- When designing a learning system, tend towards:
  - Simplicity
  - A smaller set of variables
  - Major effects

http://blogs.teradata.com/data-points/occams-razor-machine-learning/

Thanks to Peter Andreas (peter.andras@ncl.ac.uk) for a number of these slides

ARTIFICIAL NEURAL NETWORKS

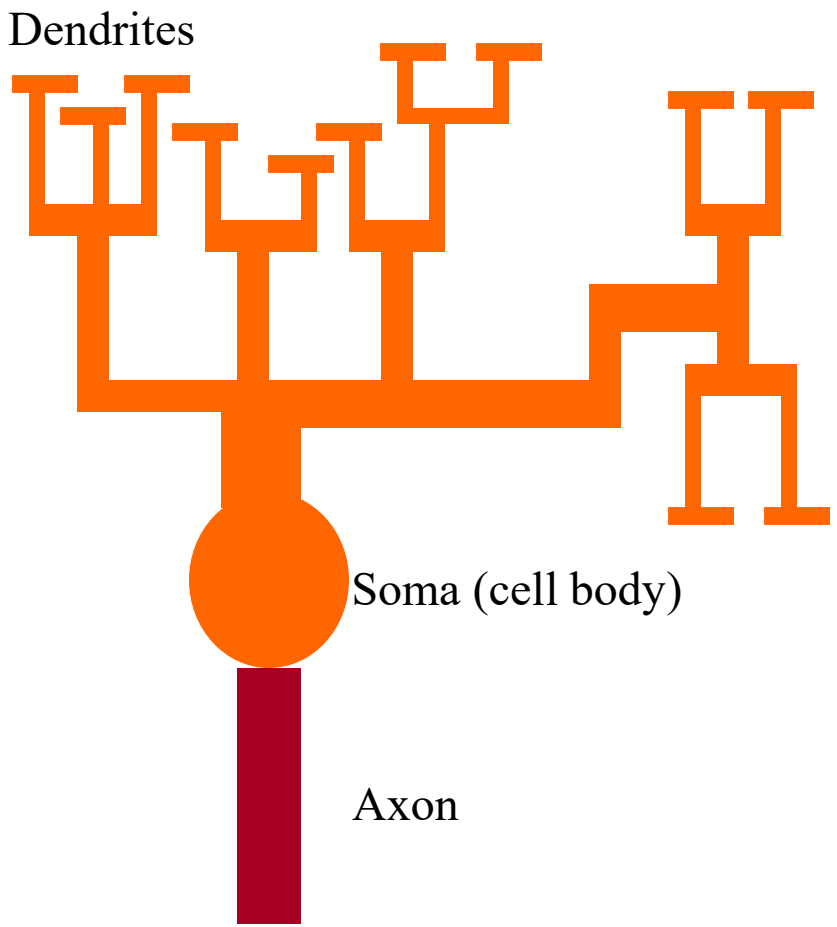
Biological inspiration

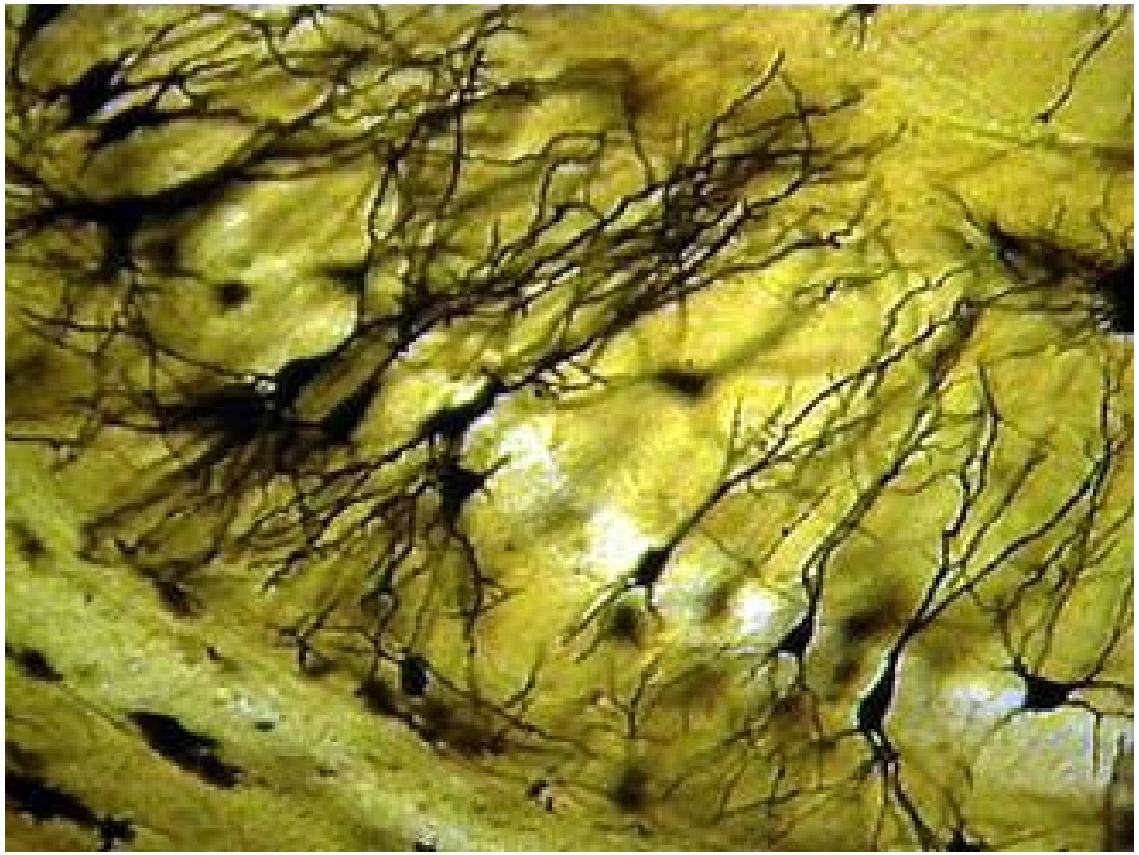
Animals are able to react adaptively to changes in their external and internal environment, and they use their nervous system to perform these behaviours.

An appropriate model/simulation of the nervous system should be able to produce similar responses and behaviours in artificial systems.

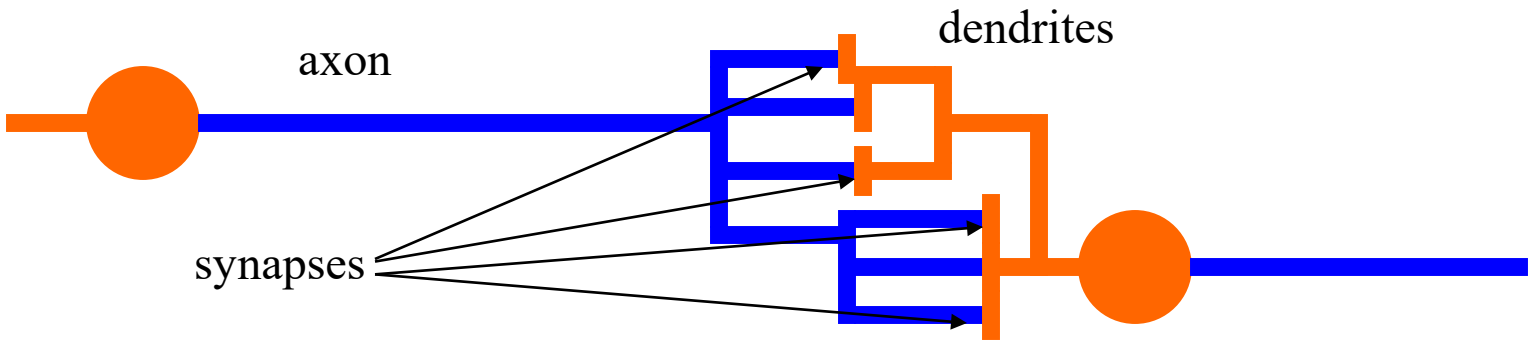
The nervous system is built of relatively simple units, the neurons, so copying their behavior and functionality should be the solution.

A network of neurons for computation/recall is inspired by the biology of the nervous system and brain





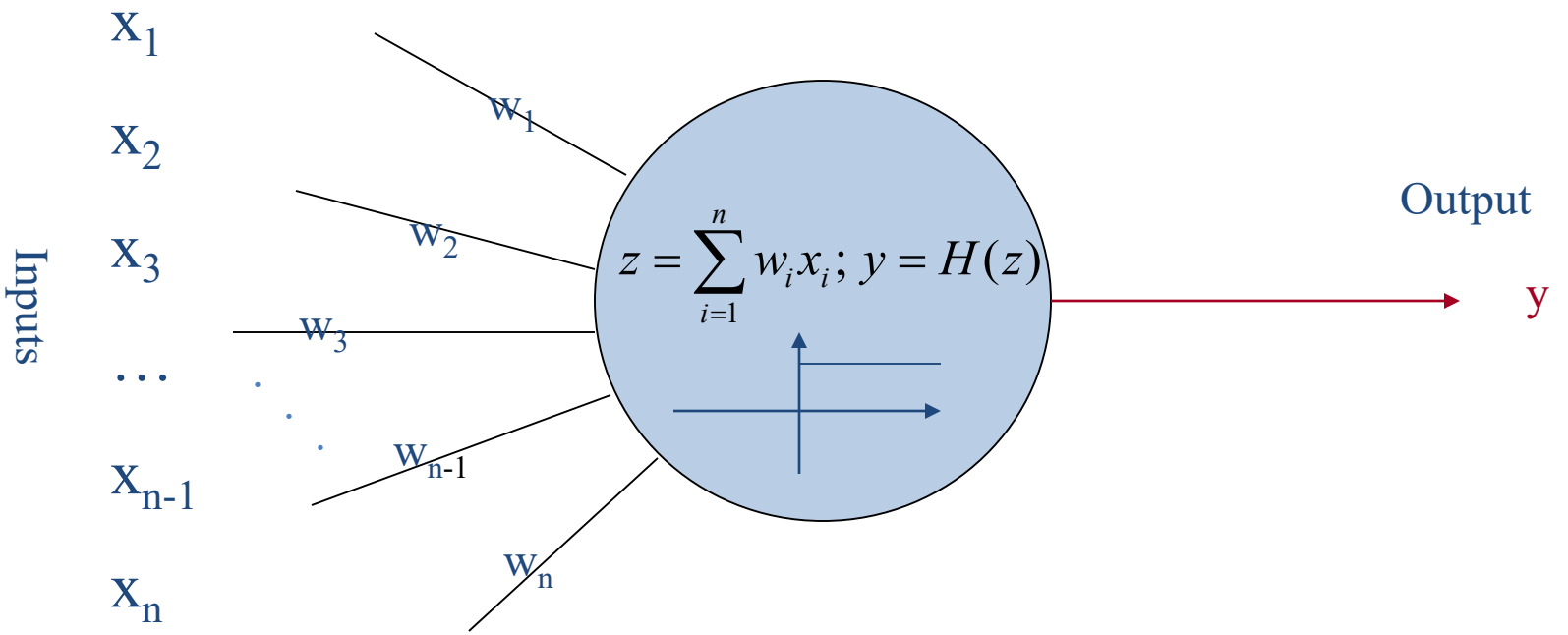
In the biological model, information transmission happens at the synapses



- The spikes travelling along the axon of the pre-synaptic neuron trigger the release of neurotransmitter substances at the synapse.
- The neurotransmitters cause excitation or inhibition in the dendrite of the post-synaptic neuron.
- The integration of the excitatory and inhibitory signals may produce spikes in the post-synaptic neuron.
- The contribution of the signals depends on the strength of the synaptic connection.

Artificial neurons work by processing information

Neurons receive and provide information in form of spikes.



The McCullogh-Pitts model (note the step function)

The classical model of an artificial neuron comprises a set of weights (including a bias term), a summing function and an activation or "squashing" function

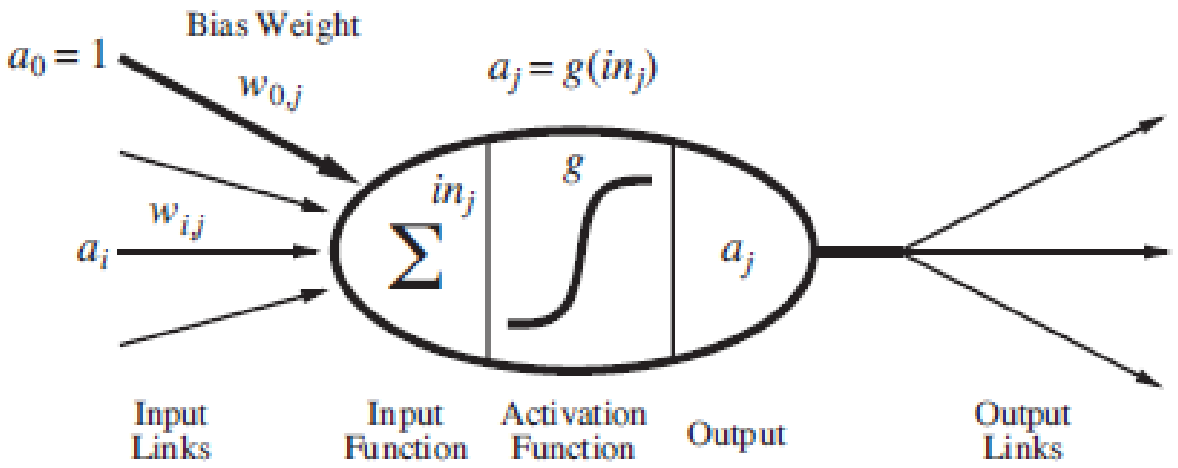
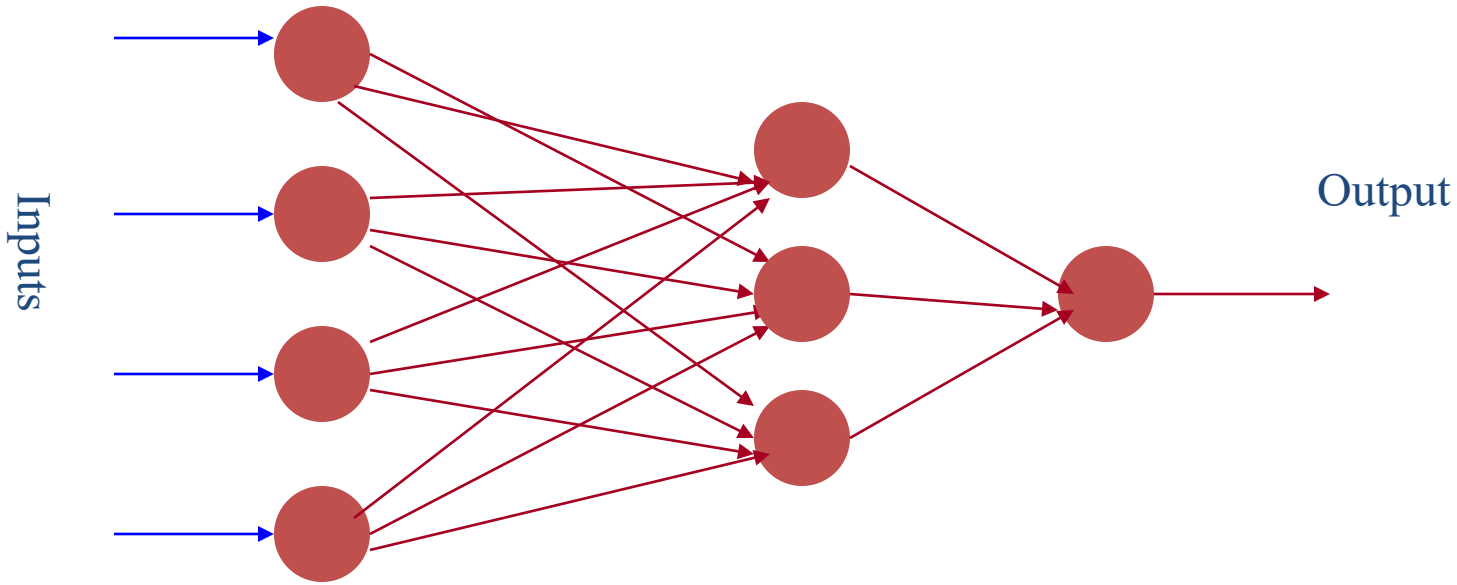


Figure 18.19 A simple mathematical model for a neuron. The unit's output activation is $a_j = g(\sum_{i=0}^{n} w_{i,j} a_i)$, where a_i is the output activation of unit i and $w_{i,j}$ is the weight on the link from unit i to this unit.

The topology of an artificial neural network (this one has a single hidden layer)



An artificial neural network is composed of many artificial neurons that are linked together according to a specific network architecture. The objective of the neural network is to transform the inputs into meaningful outputs.

Learning principle for artificial neural networks

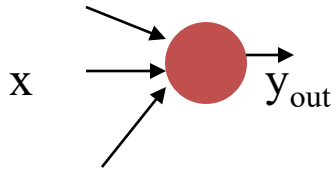
- ENERGY MINIMIZATION
- We need an appropriate definition of energy for artificial neural networks, and having that we can use mathematical optimisation techniques to find how to change the weights of the synaptic connections between neurons.
- ENERGY = measure of task performance error

MLP neural network, assuming sigmoidal activation

MLP = multi-layer perceptron

Perceptron:

$$y_{out} = w^T x$$



MLP neural network:

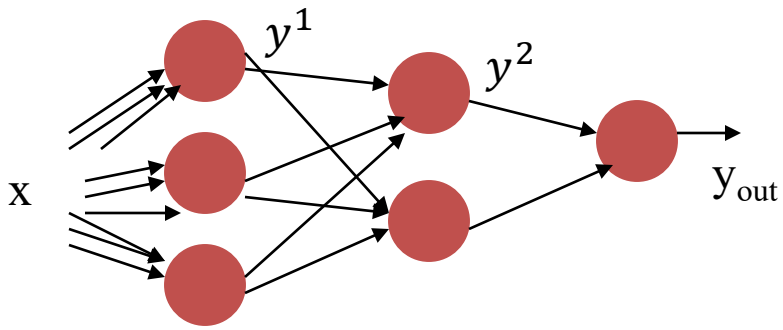
$$y_k^1 = \frac{1}{1+e^{-w^{1kT}x - a_k^1}}, k = 1,2,3$$

$$y^1 = (y_1^1, y_2^1, y_3^1)^T$$

$$y_k^2 = \frac{1}{1+e^{-w^{2kT}y^1 - a_k^2}}, k = 1,2$$

$$y^2 = (y_1^2, y_2^2)^T$$

$$y_{out} = \sum_{k=1}^{2} w_k^3 y_k^2 = w^{3T} y^2$$



Neural network as a function approximator

- control
- classification
- prediction
- approximation

These can be reformulated in general as

FUNCTION APPROXIMATION tasks.

Approximation: given a set of values of a function g(x) build a neural network that approximates the g(x) values for any input x.

Neural network function approximation task

- Data: set of value pairs: $(\boldsymbol{x}^t, \boldsymbol{y}_t), \boldsymbol{y}_t = g(\boldsymbol{x}^t) + \boldsymbol{z}_t$; $\boldsymbol{z}_t$ is random measurement noise.
- Objective: find a neural network that represents the input / output transformation (a function) $F(\boldsymbol{x}, \boldsymbol{W})$ such that $F(\boldsymbol{x}, \boldsymbol{W})$ approximates $g(x)$ for all x
- Error measure:

$$E = \frac{1}{N}\sum_{t=1}^{N}(F(\boldsymbol{x}_t; \boldsymbol{W}) - \boldsymbol{y}_t)^2$$

- Rule for changing the synaptic weights:

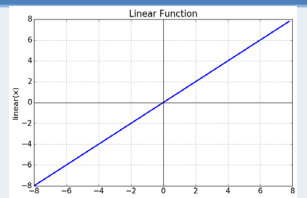
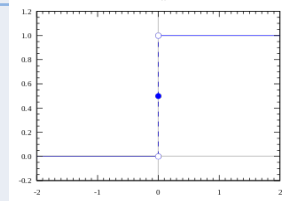
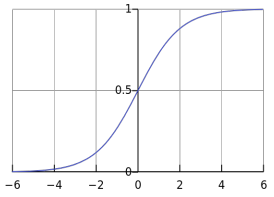
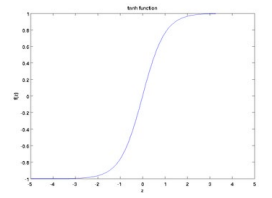
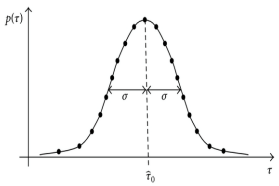
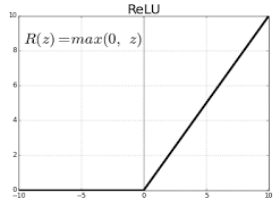
$$\Delta \boldsymbol{w}_i^j = -c \cdot \frac{\partial E}{\partial \boldsymbol{w}_i^j}(\boldsymbol{W})$$

$$\boldsymbol{w}_i^{j,new} = \boldsymbol{w}_i^j + \Delta \boldsymbol{w}_i^j$$

- c is the learning parameter (usually a constant)

The output layer activation function is determined by the desired output of the ANN

Input and hidden layer activation functions can be more difficult to choose

Function name	Equation	Plot	Range
linear	$y = k(\underline{w} \cdot \underline{x})$		$[-\infty, \infty]$
step	$y = \begin{cases} 1, & \underline{w} \cdot \underline{x} \geq 0 \\ 0, & \underline{w} \cdot \underline{x} < 0 \end{cases}$		$[0,1]$
sigmoid	$y = \frac{1}{1 + e^{-(\underline{w} \cdot \underline{x})}}$		$[0,1]$
tanh	$y = \tanh(\underline{w} \cdot \underline{x}) = \frac{2}{1 + e^{-2(\underline{w} \cdot \underline{x})}} - 1$		$[-1,1]$
Gaussian	$y = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{\|\underline{x} - \underline{w}\|^2}{2\sigma^2}}$		$[0,1]$
relu	$y = \max(0, x)$		$[0, \infty]$

Most activation functions produce higher output signals for higher weighted input values

- Linear, sigmoidal, step and tanh functions all cause higher activation for higher sums of input terms
- This generates high signals for (or *recognizes*) input vectors to the node that have specific combinations of large positive and large negative values

- Remember that the inputs to the hidden layer and output layer are outputs of earlier layers

The generalization of an ANN (how it performs on unseen data) is strongly related to the architecture of the network

- "If we choose a network that is too big, it will be able to memorize all the examples by forming a large lookup table, but will not necessarily generalize well to inputs that have not been seen before. In other words, like all statistical models, neural networks are subject to **overfitting** when there are too many parameters in the model."

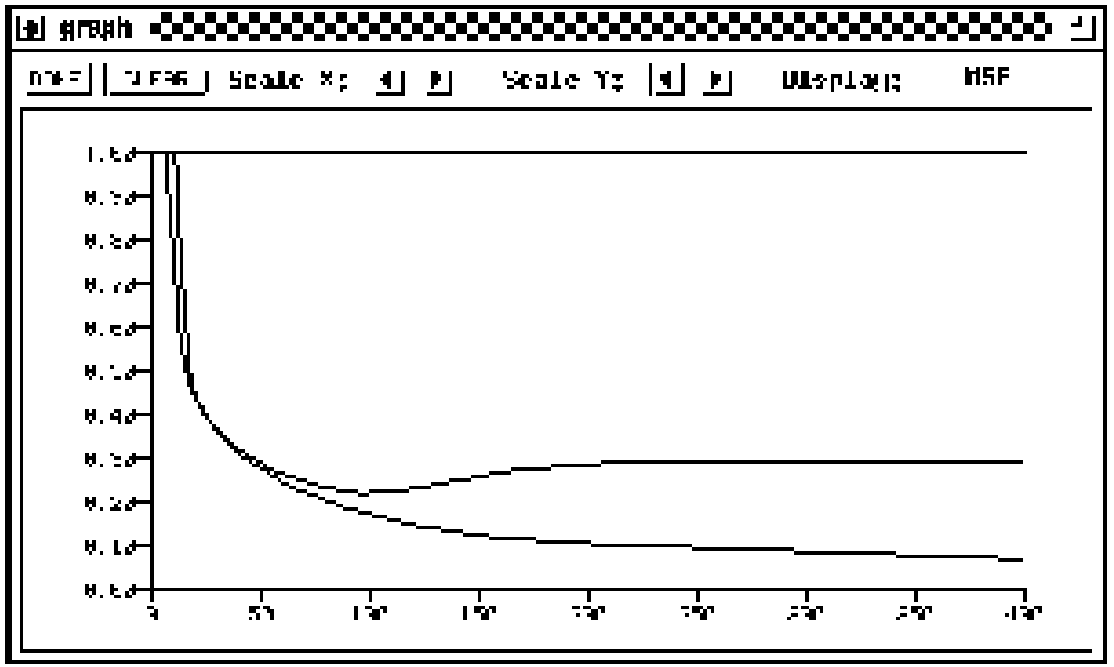
- For an ANN, fewer parameters means fewer hidden layers, few nodes in the hidden layers and fewer weights that are not close to zero.

- How do we do this?
  - 1) Start out with a large net and remove extraneous nodes and connections
  - 2) Start from zero and add nodes and layers until the performance is suitable

Generalization is perhaps the single most important attribute for an ANN

Principles:

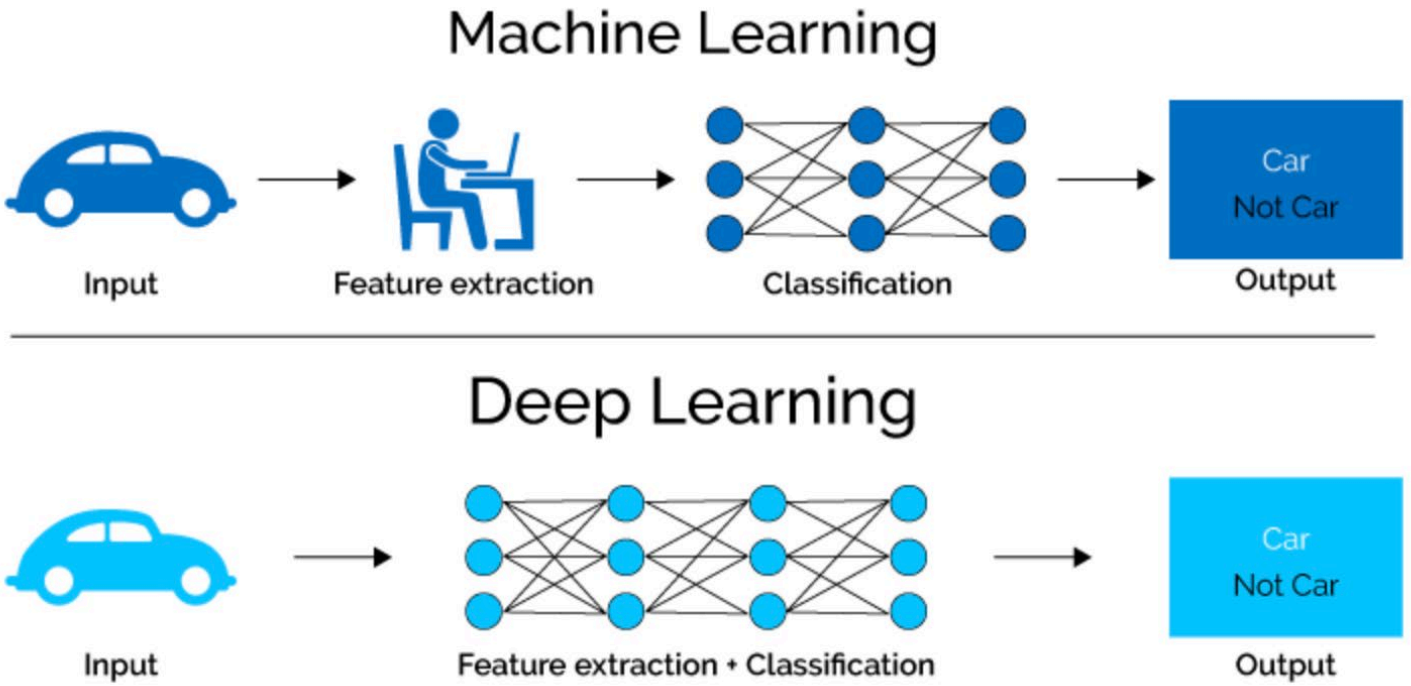
- Consider carefully the division into training and test data sets
  - Assign members randomly
- Stop training when the error on test data (not used for training) consistently increases
- Avoid overfitting by keeping the ANN architecture as simple as possible



DEEP LEARNING

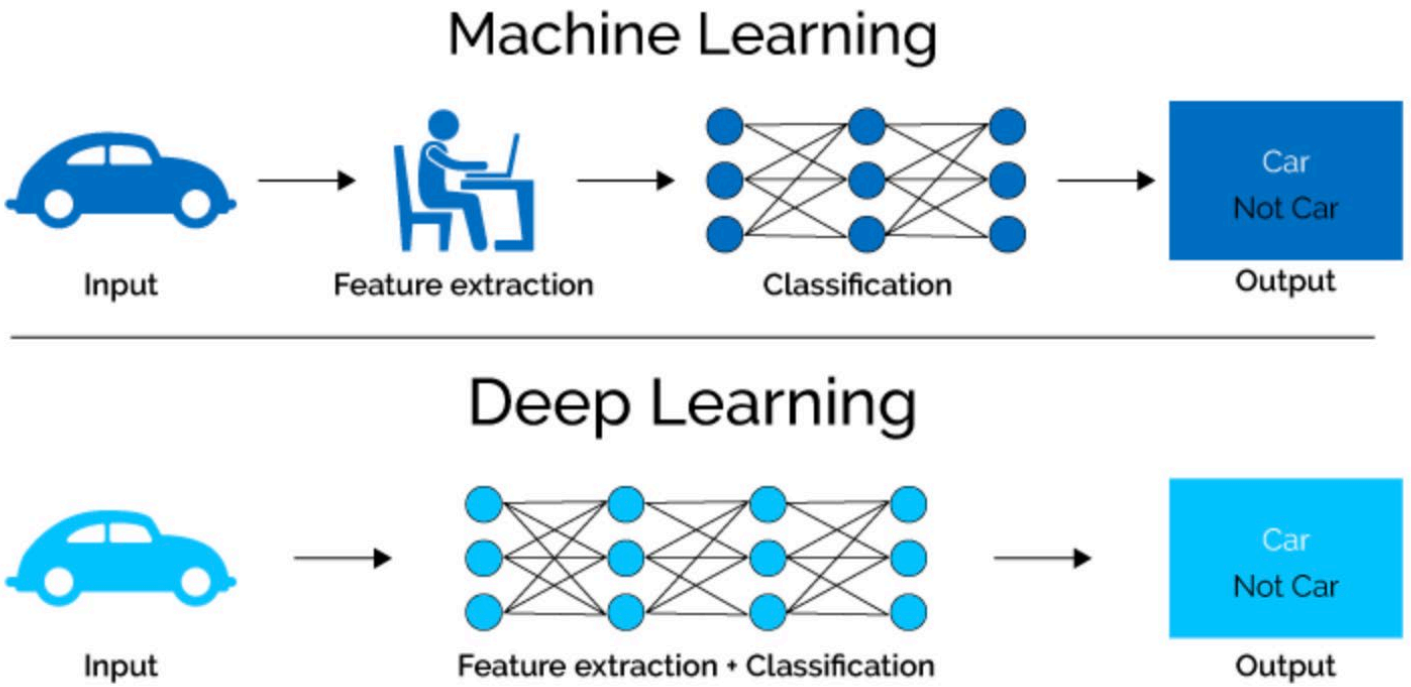
Deep Learning refers to systems built on the realization that data (and patterns in data) occur at several levels of abstraction

- Deep learning algorithms typically include the selection/calculation of input features for a given situation, as well as the calculation of net weights



Deep Learning refers to systems built on the realization that data (and patterns in data) occur at several levels of abstraction

- Deep learning algorithms typically include the selection/calculation of input features for a given situation, as well as the calculation of net weights



This diagram appears to be showing a "fully-connected" network – most Computer Vision applications use convolutional layers instead

HOW A DEEP NEURAL NETWORK SEES

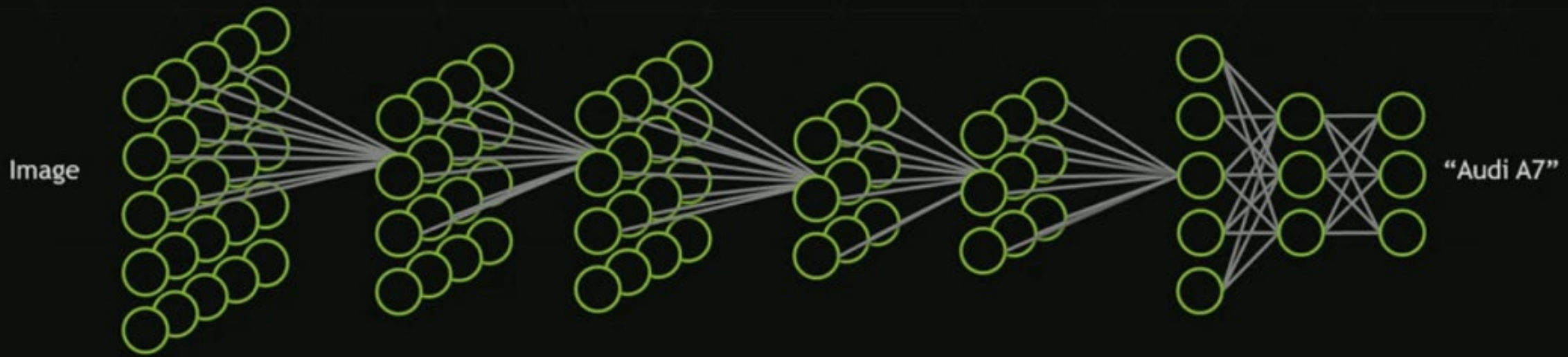


Image source: "Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks" ICML 2009 & Comm. ACM 2011. Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Ng.

REVIEW

doi:10.1038/nature14539

VIRGINIA TECH

ece BRADLEY DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

Deep learning

Yann LeCun[1,2], Yoshua Bengio[3] & Geoffrey Hinton[4,5]

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio, whereas recurrent nets have shone light on sequential data such as text and speech.

Machine-learning technology powers many aspects of modern society: from web searches to content filtering on social networks to recommendations on e-commerce websites, and it is increasingly present in consumer products such as cameras and smartphones. Machine-learning systems are used to identify objects in images, transcribe speech into text, match news items, posts or products with users' interests, and select relevant results of search. Increasingly, these applications make use of a class of techniques called deep learning.

Conventional machine-learning techniques were limited in their ability to process natural data in their raw form. For decades, constructing a pattern-recognition or machine-learning system required careful engineering and considerable domain expertise to design a feature extractor that transformed the raw data (such as the pixel values of an image) into a suitable internal representation or feature vector from which the learning subsystem, often a classifier, could detect or classify patterns in the input.

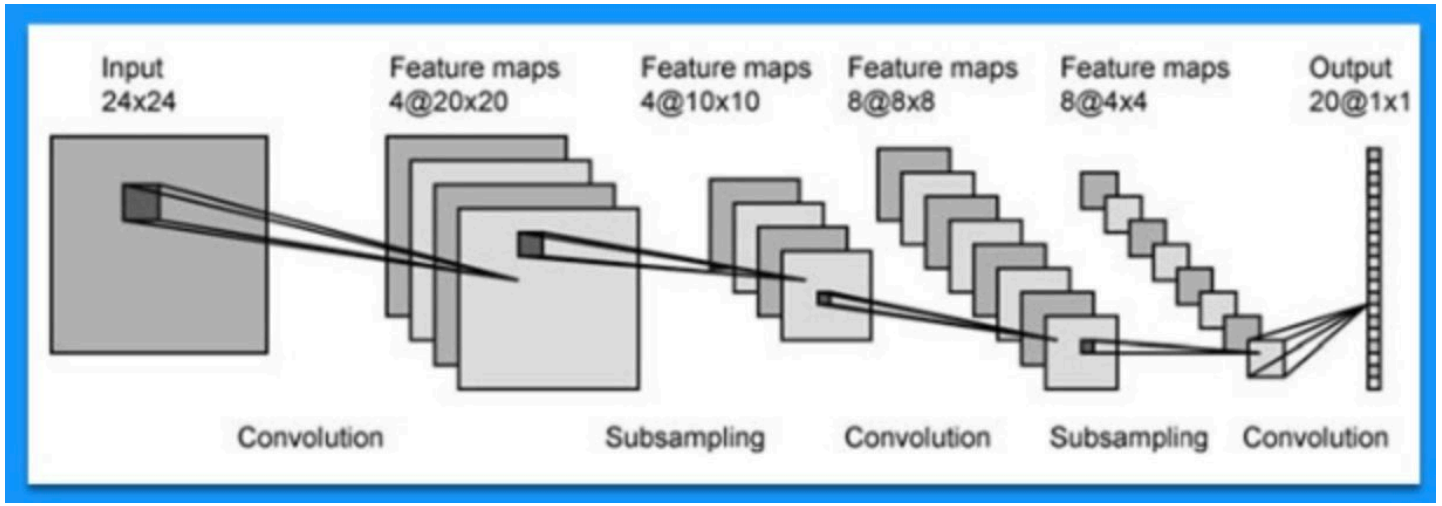
Representation learning is a set of methods that allows a machine to be fed with raw data and to automatically discover the representations needed for detection or classification. Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level. With the composition of enough such transformations, very complex functions can be learned. For classification tasks, higher layers of representation amplify aspects of the input that are important for discrimination and suppress irrelevant variations. An image, for example, comes in the form of an array of pixel values, and the learned features in the first layer of representation typically represent the presence or absence of edges at particular orientations and locations in the image. The second layer typically detects motifs by spotting particular arrangements of edges, regardless of small variations in the edge positions. The third layer may assemble motifs into larger combinations that correspond to parts of familiar objects, and subsequent layers would detect objects as combinations of these parts. The key aspect of deep learning is that these layers of features are not designed by human engineers: they are learned from data using a general-purpose learning procedure.

Deep learning is making major advances in solving problems that have resisted the best attempts of the artificial intelligence community for many years. It has turned out to be very good at discovering intricate structures in high-dimensional data and is therefore applicable to many domains of science, business and government. In addition to beating records in image recognition[1–4] and speech recognition[5–7], it has beaten other machine-learning techniques at predicting the activity of potential drug molecules[8], analysing particle accelerator data[9,10], reconstructing brain circuits[11], and predicting the effects of mutations in non-coding DNA on gene expression and disease[12,13]. Perhaps more surprisingly, deep learning has produced extremely promising results for various tasks in natural language understanding[14], particularly topic classification, sentiment analysis, question answering[15] and language translation[16,17].

http://pages.cs.wisc.edu/~dyer/cs540/handouts/deep-learning-nature2015.pdf

Deep Learning generally requires multiple hidden layers and large numbers of nodes to execute; so, learning time can be a problem

- The Convolutional Neural Network (CNN) is a common building block of DL systems for image analysis



- Many DL libraries use multiple CPUs and/or GPUs to make learning reasonable
- "Recent ConvNet architectures have 10 to 20 layers of ReLUs, hundreds of millions of weights, and billions of connections between units. Whereas training such large networks could have taken weeks only two years ago, progress in hardware, software and algorithm parallelization have reduced training times to a few hours." – LeCun, *Nature*

Layers in a convolutional neural network have unique and specific roles – unlike those in a flat MLP

- Convolutional layers apply a set of 2D convolutional masks to all parts of an image, and produce feature maps for input to the next layer
- Batch Normalization applies transformations to feature maps to preserve constant mean and variance
- Activation functions are as in an MLP – the ReLU (rectified linear unit) is the most common internal activation function in Deep Learning
- Pooling reduces the size of feature maps produced by internal layers, usually by taking the max of the output in a neighborhood
- Downsampling / Upsampling layers transform the feature maps for processing by the next layer

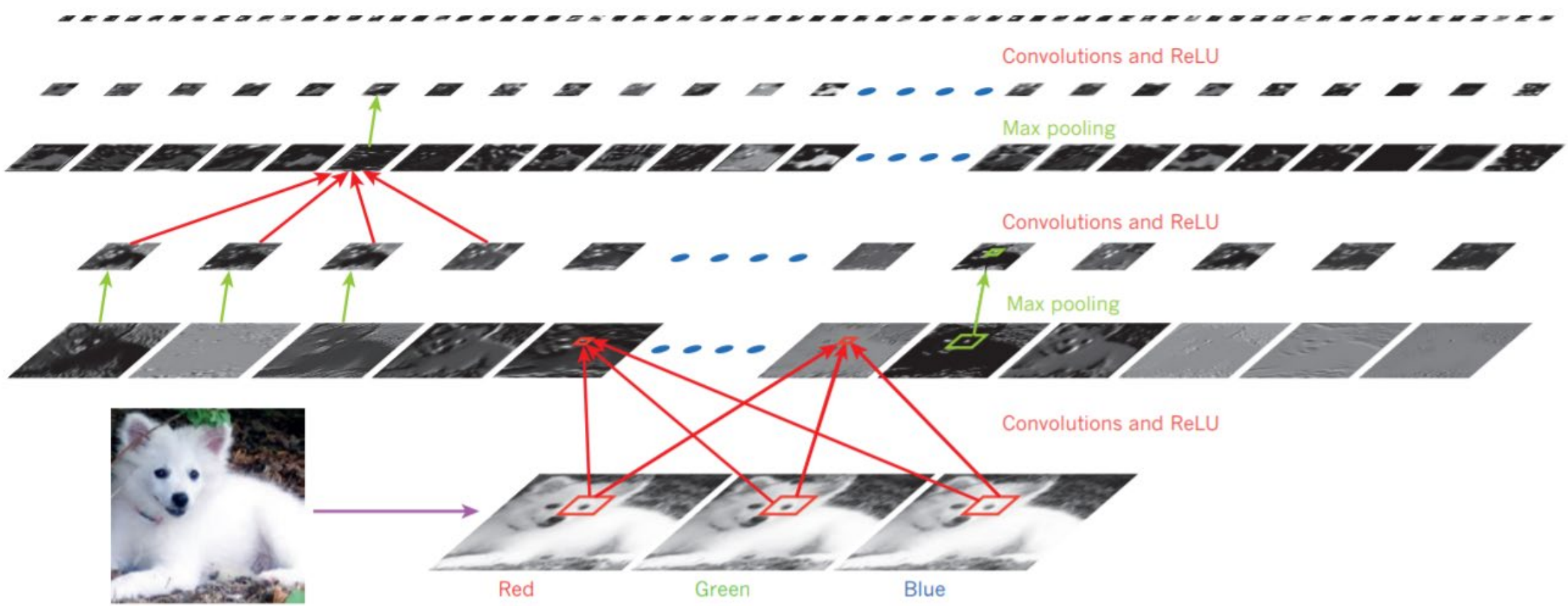


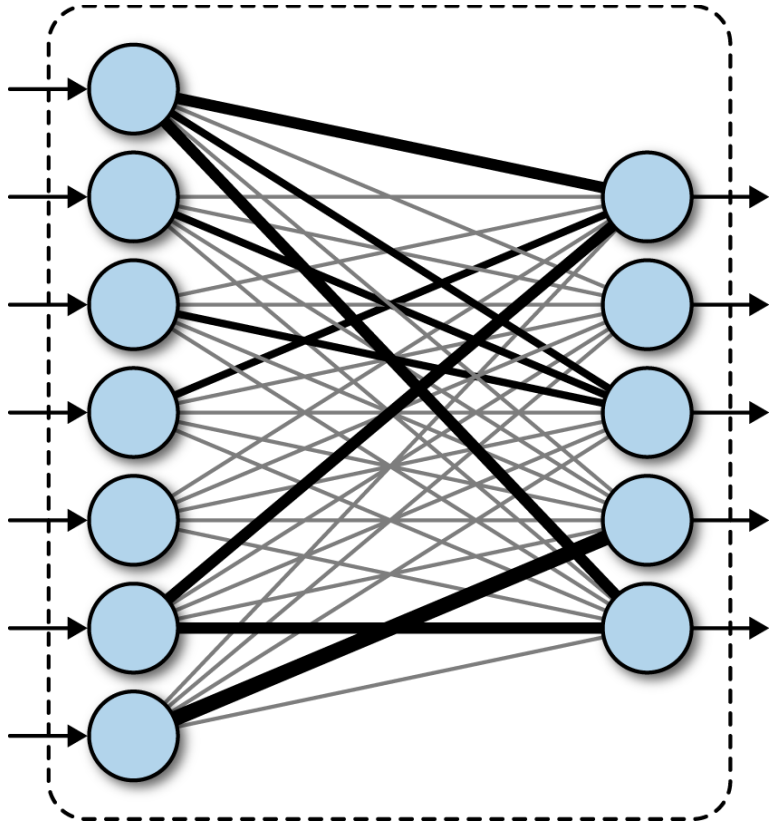
Figure 2 | Inside a convolutional network. The outputs (not the filters) of each layer (horizontally) of a typical convolutional network architecture applied to the image of a Samoyed dog (bottom left; and RGB (red, green, blue) inputs, bottom right). Each rectangular image is a feature map corresponding to the output for one of the learned features, detected at each of the image positions. Information flows bottom up, with lower-level features acting as oriented edge detectors, and a score is computed for each image class in output. ReLU, rectified linear unit.

From LeCun 2015, "Deep Learning", in *Nature*

In a fully-connected layer, each node receives inputs from every node in the previous layer

- This is the most general network form, and allows for ultimate flexibility in automated selection of image features

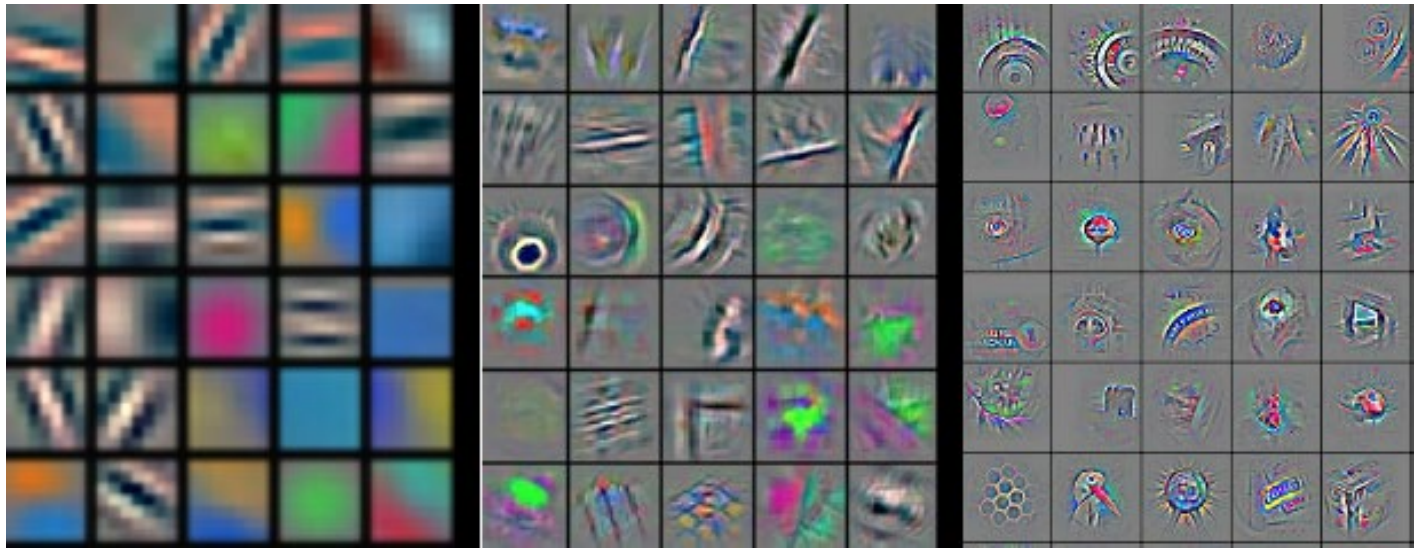
- Some weights can approach zero
  - Which is equivalent to "no connection"
- The number of weights is huge
- Training time can be long

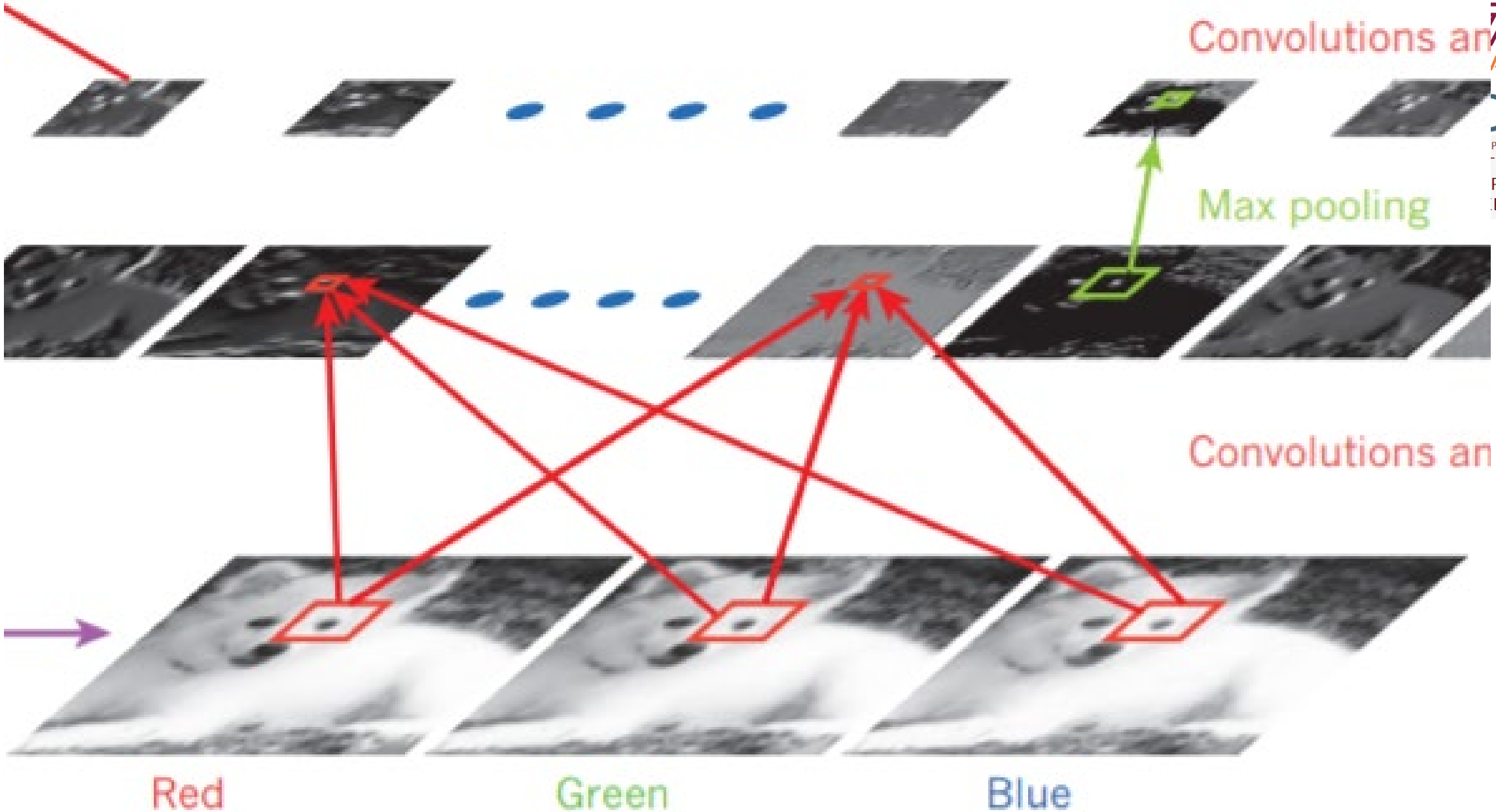


Zadeh, *TensorFlow for Deep Learning*, O'Reilly

In a Convolutional layer, a set of kernels are applied to the input image (or result of the previous layer)

- This is analogous to human vision
- The "size" of the kernel is determined by system design
  - In a fully-connected network, the kernel is the size of the image
- Learning will produce kernel coefficients as required
  - Many sets of weights in a CNN are "tied" – constrained to have the same value
- Recall that we have access to all color planes of the input



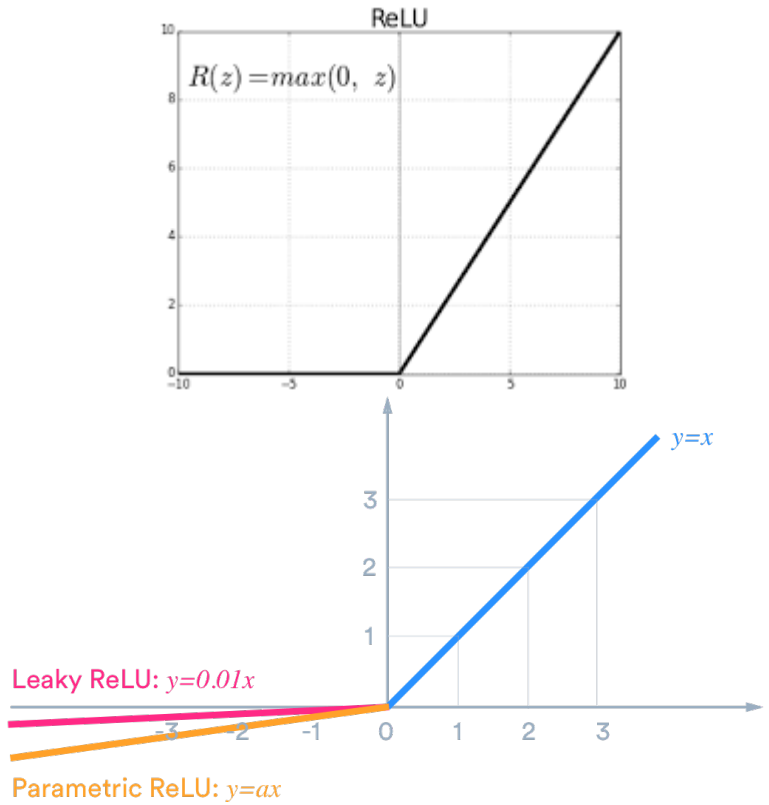


From LeCun 2015, “Deep Learning”, in *Nature*

Pooling takes outputs from the convolutions and reduces them in resolution – typically by either averaging or taking the max of neighboring filter responses

- Recall that the inputs to pooling are the results of applying kernels
- Thus, pooling has the effect of reducing run-time and keeping responses at the proper scale-space
- Most research shows that max pooling has better performance than average pooling on many types of images
  - Faster convergence, better invariance and improved generalization
  - Should be the default
- Other variations are used
  - Methods including randomness are especially interesting

ReLU (Rectified Linear Unit) Activation is currently the most common internal activation function



- ReLU is linear for positive outputs, but clamps negative outputs to zero
  - Has the "required" nonlinearity
  - Prevents all sorts of undesirable saturation and dilution in the net
- Some people favor "Leaky ReLU" or "Parametric ReLU"
  - To allow *some* contribution from negative neuron outputs

After one or more sequences of convolutional, pooling and activation layers, we often use a linear classifier output stage to discriminate between classes of images or objects

- Training is the phase of setting weights in all layers:
  - Convolutional: what kernels are most effective?
  - Pooling: how are feature maps downsampled and combined?
  - Summing/activation: how are features weighted and combined?

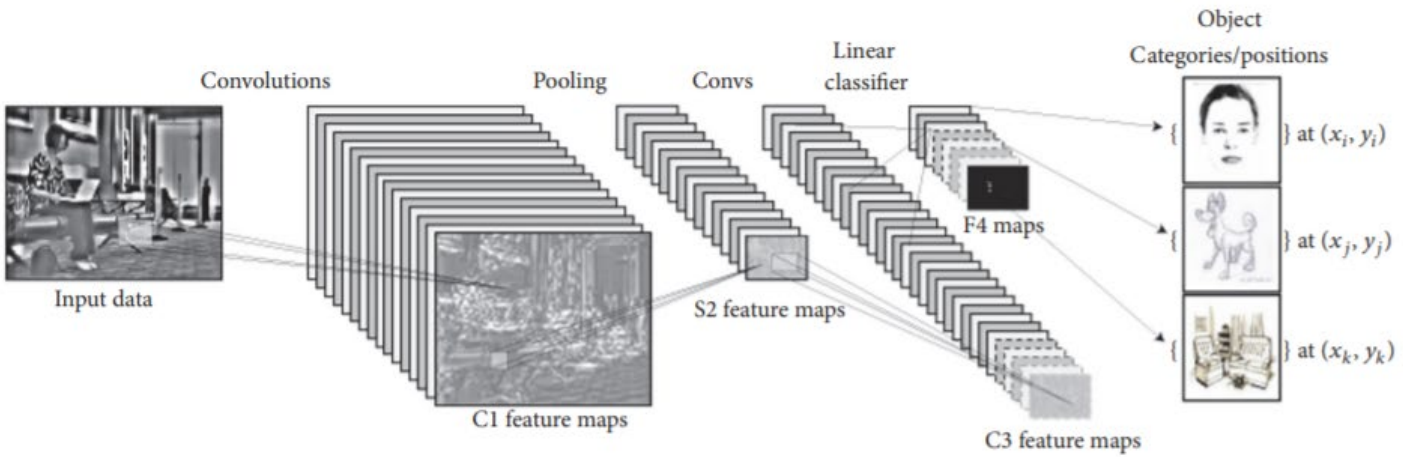
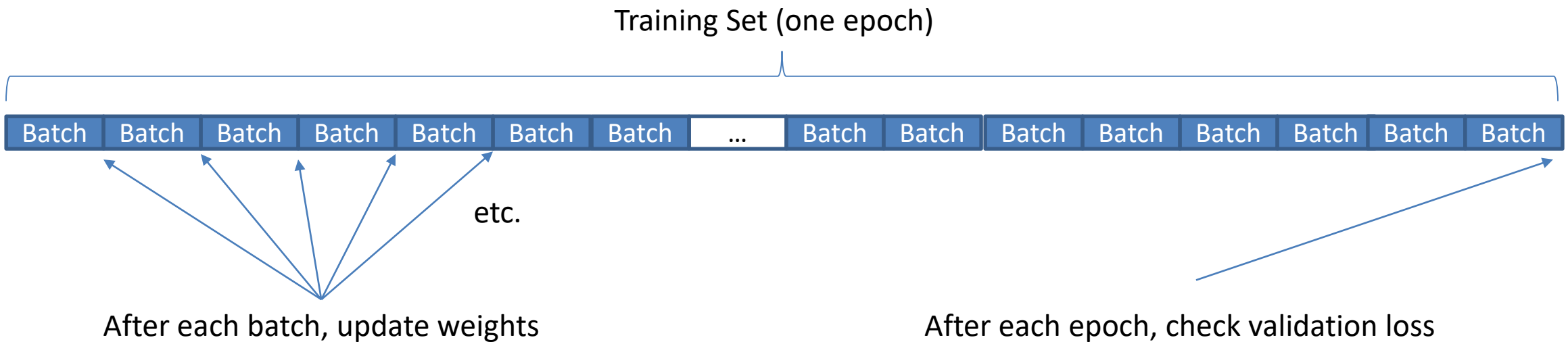


FIGURE 1: Example architecture of a CNN for a computer vision task (object detection).

Voulodimos 2018

To attempt to accelerate learning of a Deep Model, Batch learning is often used

- A batch is a subset of the training set
  - 1/20 of the whole? 1/50?
- Rather than waiting for the completion of the epoch to adjust weights, we calculate loss and update weights after each batch



How do we implement a deep learning neural network?

- Are you going to train a model or apply existing models?
- Python/Java/C++/others
- Packages like:
  - PyTorch
  - TensorFlow
  - DL4J
  - Caffe

As one example, the TensorFlow math library has a strong set of deep learning capabilities

- Supports training on GPUs
- Python, JavaScript, Swift; CPU, GPU, Mobile (deployment)

- Originally developed for internal use at Google
  - Released to the public in 2015 or so

- Very popular
- Lots of examples available

Training a Deep Learning network is a long process

- As with traditional neural networks, the base approach is generally backpropagation
  - Each weight is changed in proportion to the partial derivative of the output error with respect to that weight
  - It's a multivariate optimization problem
- Many weights, lots of data – slow process
- Much work has gone into the effort to speed up training
  - Parallel processes on GPUs
  - Cluster computing
  - Algorithm parallelization

Keep in mind the difference between *training* and *deploying* a DL model

- Very often, when we say we are doing Deep Learning, we are actually only deploying a pre-taught DL model
  - Google's TPU (tensor processing unit) is designed for deployment, not training

In my example below, you will see:

```
# load our serialized model from disk
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
```

- This model was trained by someone at UC Berkeley

We often decide to *augment* the training set, to compensate for limited data or to provoke the model to overcome expected data corruptions

Data set size

- It's often difficult to obtain enough labeled data to permit training
- We explore ways of creating synthetic samples that are "like" new data

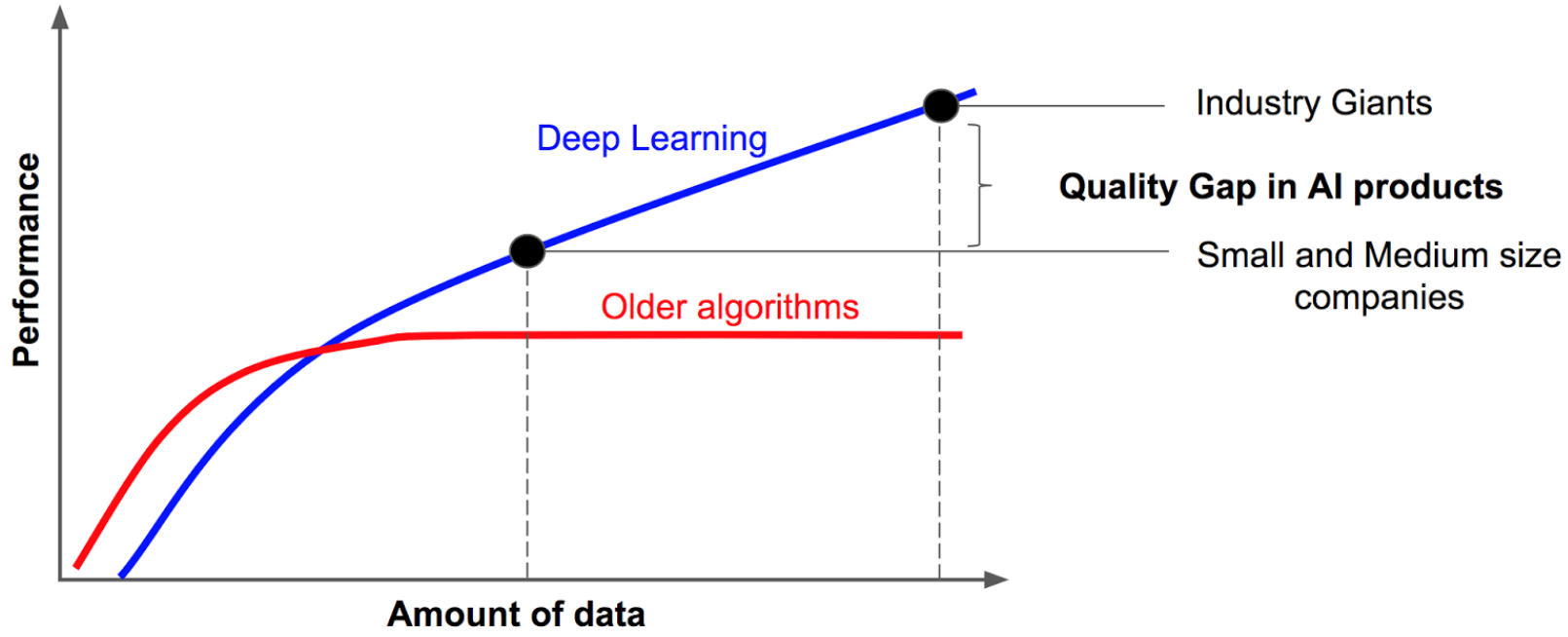
Expected data corruptions

- Noisy images, changes in contrast, etc.
- Occlusions
- Change in location and pose of image content

- A nice library called Albumentations provides utilities for augmenting image data sets

DL networks have the ability to reap increased accuracy from large amounts of data, without losing generalization – this is a significant technical advantage

- Deep Learning researchers are not yet common
- Developing solutions can be expensive in CPU time and skilled labor



OpenCV 4.5.2
Open Source Computer Vision

Main Page | Related Pages | Modules | Namespaces | Classes | Files | Examples | Java documentation | Search

OpenCV Tutorials

Deep Neural Networks (dnn module)

- Load Caffe framework models
- How to enable Halide backend for improve efficiency
- How to schedule your network for Halide backend
- How to run deep networks on Android device
- YOLO DNNs
- How to run deep networks in browser
- Custom deep learning layers support
- How to run custom OCR model
- High Level API: TextDetectionModel and TextRecognitionModel

PyTorch models with OpenCV

In this section you will find the guides, which describe how to run classification, segmentation and detection PyTorch DNN models with OpenCV.

- Conversion of PyTorch Classification Models and Launch with OpenCV Python
- Conversion of PyTorch Classification Models and Launch with OpenCV C++
- Conversion of PyTorch Segmentation Models and Launch with OpenCV

TensorFlow models with OpenCV

In this section you will find the guides, which describe how to run classification, segmentation and detection TensorFlow DNN models with OpenCV.

- Conversion of TensorFlow Classification Models and Launch with OpenCV Python
- Conversion of TensorFlow Detection Models and Launch with OpenCV Python
- Conversion of TensorFlow Segmentation Models and Launch with OpenCV

```
### from https://www.pyimagesearch.com/2017/08/21/deep-learning-with-opencv/

import numpy as np
import argparse
import time
import cv2

pathname = "C:\\Data\\DLimages\\"

# load the input image from disk
image = cv2.imread(pathname + args["image"])
print(image.shape)

# load the class labels from disk
rows = open(args["labels"]).read().strip().split("\n")
classes = [r[r.find(" ") + 1:].split(",")[0] for r in rows]

# our CNN requires fixed spatial dimensions for input image(s)
# so we need to ensure it is resized to 224x224 pixels while
# do mean subtraction (104, 117, 123) to normalize the input;
# after executing this command our "blob" now has the shape:
# (1, 3, 224, 224)
blob = cv2.dnn.blobFromImage(image, 1, (224, 224), (104, 117, 123))

# load our serialized model from disk
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

# set blob as input to the network and perform a forward pass to
# obtain our output classification
net.setInput(blob)
start = time.time()
preds = net.forward()
end = time.time()
print("[INFO] classification took {:.5} seconds".format(end - start))

# sort the indexes of the probabilities in descending order (higher
# probabilitiy first) and grab the top-5 predictions
idxs = np.argsort(preds[0])[::-1][:5]

# loop over the top-5 predictions and display them
for (i, idx) in enumerate(idxs):
    # draw the top prediction on the input image
    if i == 0:
        text = "Label: {}, {:.2f}%".format(classes[idx],
            preds[0][idx] * 100)
        cv2.putText(image, text, (5, 25),
cv2.FONT_HERSHEY_SIMPLEX,
            0.7, (0, 0, 255), 2)
    # display the predicted label + associated probability to the
    # console
    print("[INFO] {}. label: {}, probability: {:.5}".format(i + 1,
        classes[idx], preds[0][idx]))

# display the output image
cv2.imshow("Image", image)
cv2.waitKey(0)
```

7/08/21/deep-

```
# set blob as input to the network and perform a forward pass to
# obtain our output classification
net.setInput(blob)
start = time.time()
preds = net.forward()
end = time.time()
print("[INFO] classification took {:.5} seconds".format(end -
start))

# sort the indexes of the probabilities in descending order
(higher
# probabilitiy first) and grab the top-5 predictions
idxs = np.argsort(preds[0])[::-1][:5]

# loop over the top-5 predictions and display them
for (i, idx) in enumerate(idxs):
```

n")

```
classes = [r[r.find(" ") + 1:].split(",")[0] for r

# our CNN requires fixed spatial dimensions f
# so we need to ensure it is resized to 224x2
# do mean subtraction (104, 117, 123) to no
# after executing this command our "blob" nc
# (1, 3, 224, 224)
blob = cv2.dnn.blobFromImage(image, 1, (22
123))

# load our serialized model from disk
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["protot
args["model"])
```

```
{'image': 'jemma.png', 'prototxt': 'bvlc_googlenet.prototxt', 'model':
'bvlc_googlenet.caffemodel', 'labels': 'synset_words.txt'}
(375, 500, 3)
[INFO] loading model...
[INFO] classification took 0.076728 seconds
[INFO] 1. label: beagle, probability: 0.73989
[INFO] 2. label: Labrador retriever, probability: 0.083672
[INFO] 3. label: soccer ball, probability: 0.016035
[INFO] 4. label: dalmatian, probability: 0.012536
[INFO] 5. label: Walker hound, probability: 0.011524
```

A SAMPLE DEEP LEARNING APPLICATION

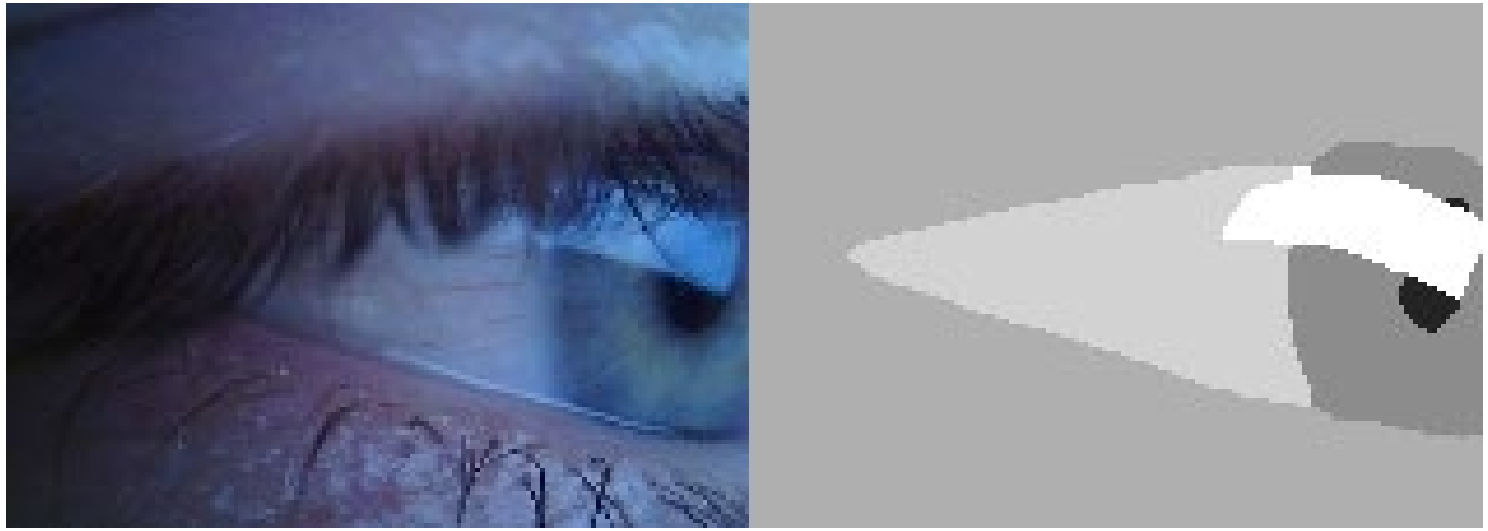
I’d like to present a very simple deep learning application from some work that I am doing at Globe Biomedical

- There are some restrictions on what I can show or discuss
- The application is to perform *semantic segmentation* on images of the eye, taken with Globe Biomedical’s wearable imaging devices
- Semantic image segmentation is the process of classifying image pixels, to separate the image into regions representing different objects or materials

Here, we want to separate eye images into regions corresponding to different parts of the eye and surrounding

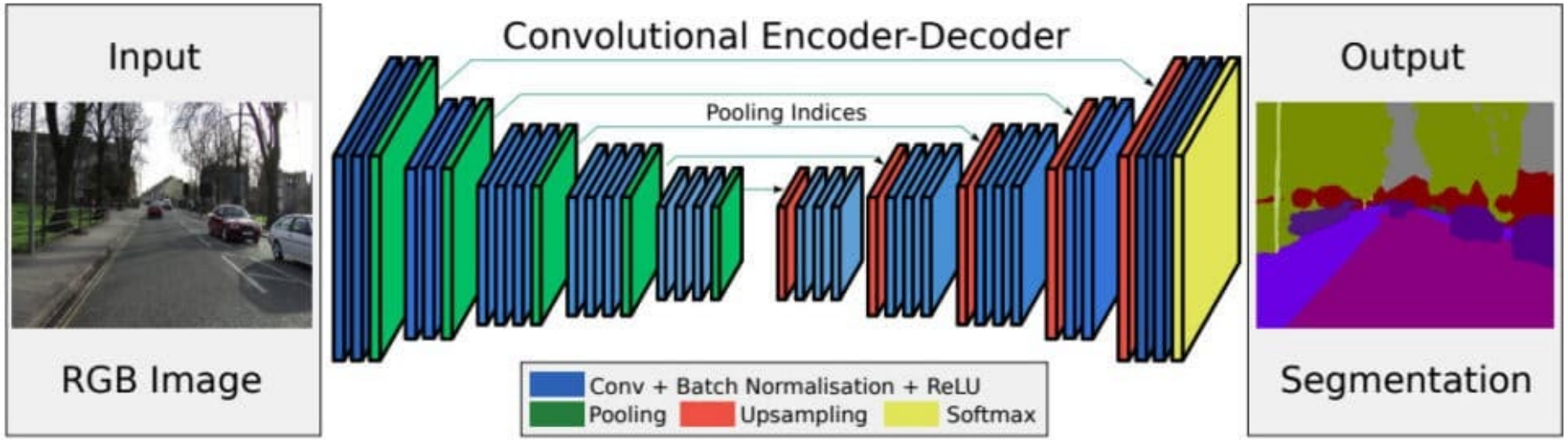
Eye images contain regions corresponding to:

- Pupil
- Eyebrow
- Eyelash
- Iris
- Skin
- Sclera (white of the eye)
- Glare (usually from a screen)



Model Architecture

- I can’t tell you all of the specifics, but I used a fairly conventional CNN for semantic image segmentation



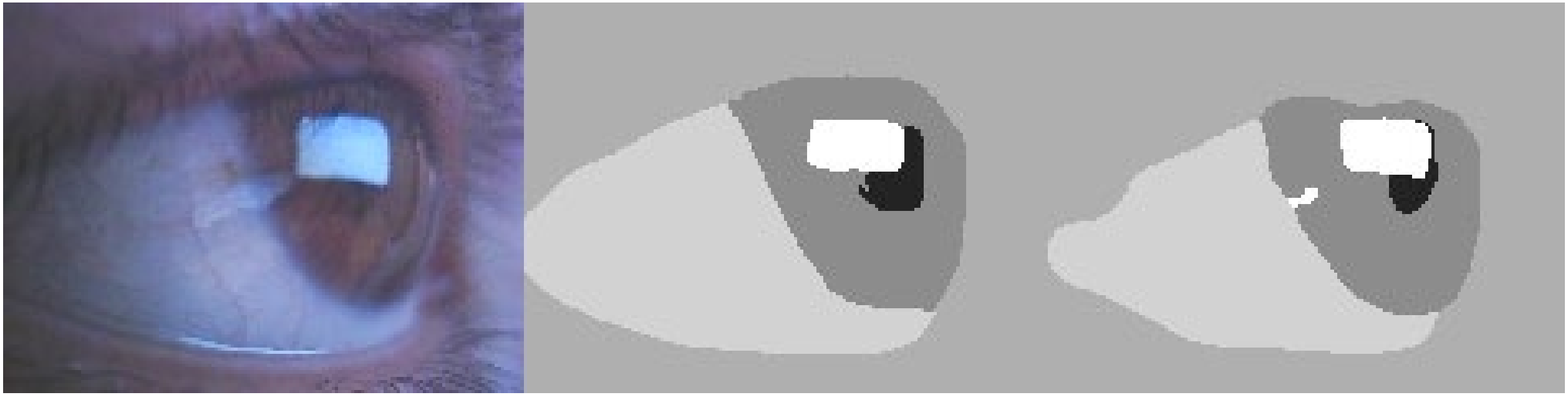
- A good reference to the general field is at https://cnvrg.io/semantic-segmentation/

The model output is a image-sized map of the regions to which each pixel is estimated to belong (gray levels indicate the regions)

original image

true segments (target)

model output

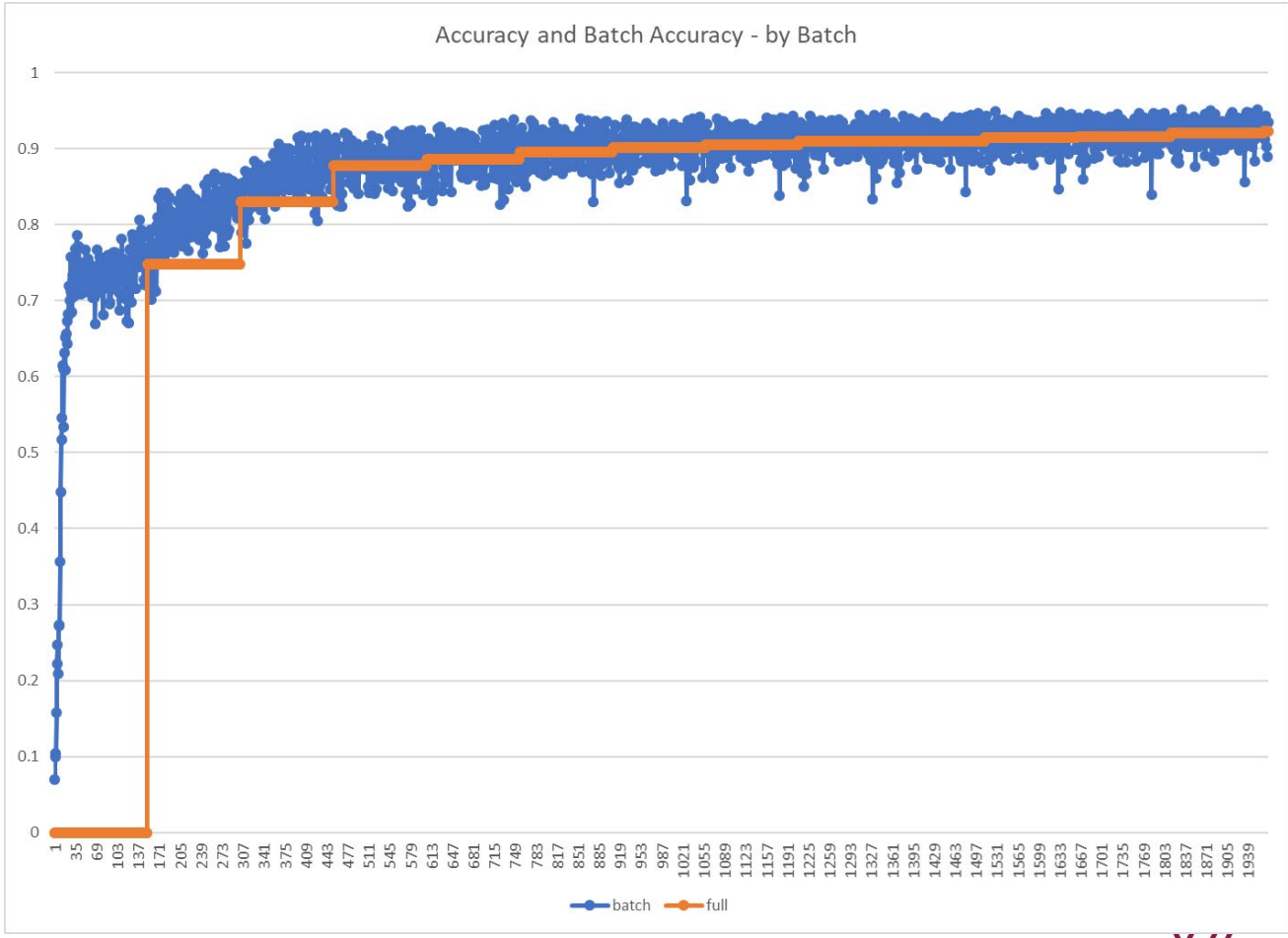


I chose to use PyTorch for this model – it supports Deep Learning on CNNs, and has support for GPUs

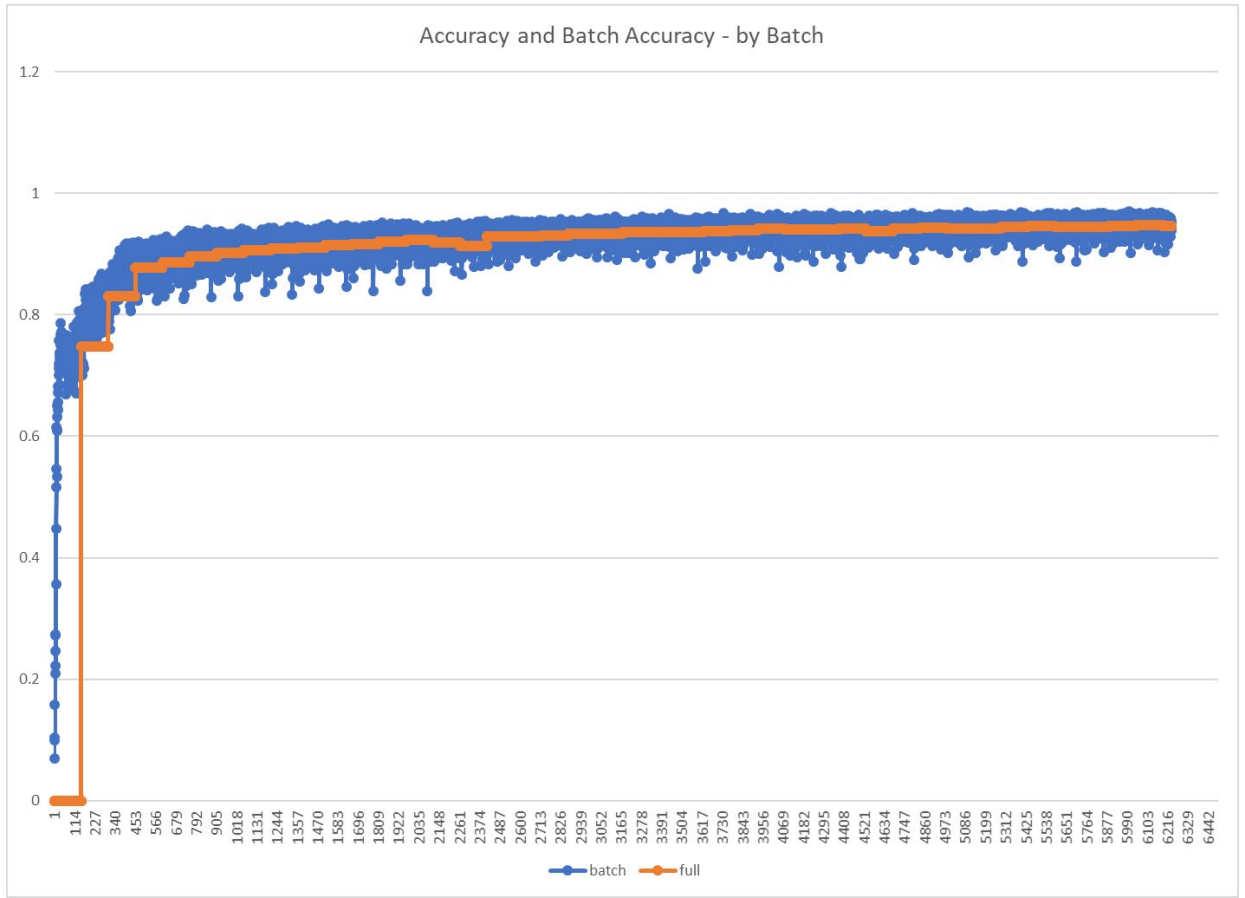
- Training can take a long time
  - Days is not uncommon
- If one has a good graphics card, the CUDA cores on the graphics processor can be used by the training algorithms
  - PyTorch has nice support for NVIDIA GPUs
  - Trains maybe 5x to 10x faster
- A convolutional neural network on PyTorch is implemented as a sequence of modules
  - torch.nn.Conv2d() is a convolutional layer
  - torch.nn.MaxPool2d() performs max pooling of inputs
  - torch.nn.BatchNorm2d() is a normalizing layer
  - torch.nn.ReLU() is a rectified linear unit activation function
  - etc.

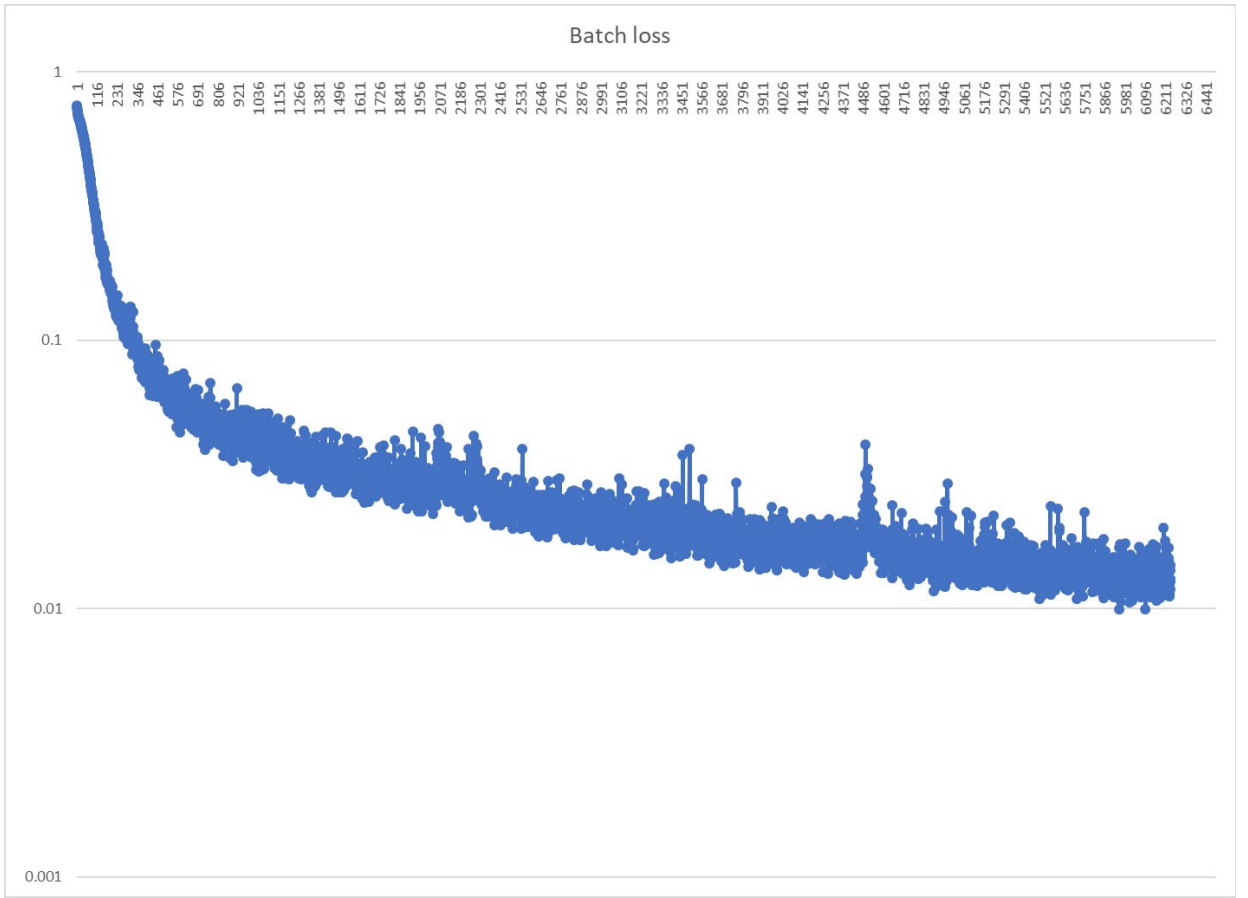
I am using both batch learning and augmentations

- I only have 108 labeled images
- For each image, I create 20 copies with random:
  - Added image noise
  - Rotation and translation
  - Change in brightness, contrast and color
- So, the training set is 108x21=2268
  - One epoch is 2268 images
  - Batches are size 20

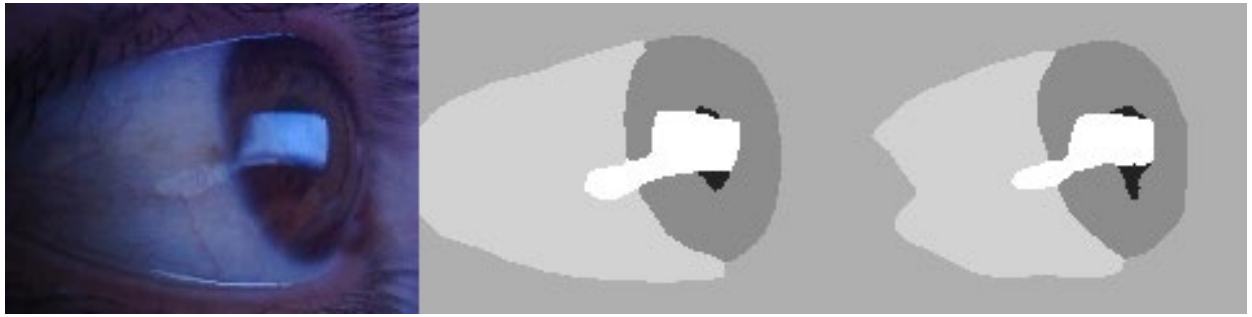


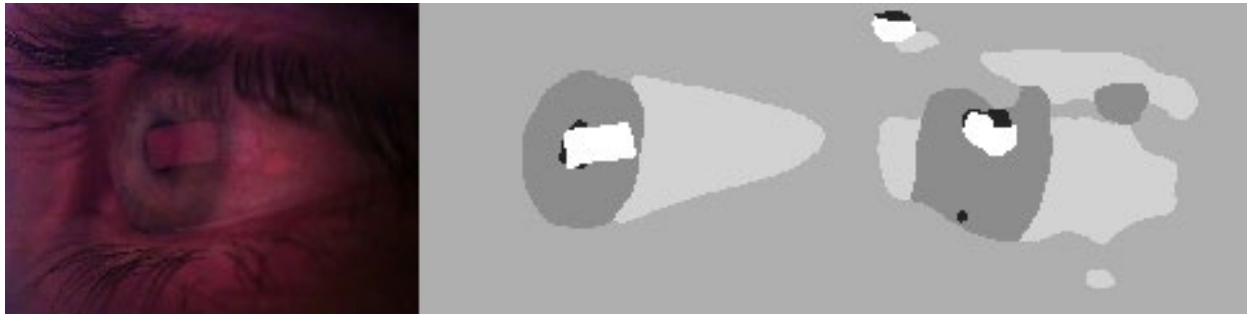
Learning curves – accuracy (batch and epoch) on the left, loss on the right





Overall accuracy is around 98% at the pixel level; sometimes it works great – other times, not as well (working on acquiring more real training data)





APPLICATIONS OF DEEP NEURAL NETWORKS

A woman is throwing a **frisbee** in a park.

A **dog** is standing on a hardwood floor.

A **stop** sign is on a road with a mountain in the background

A little **girl** sitting on a bed with a teddy bear.

A group of **people** sitting on a boat in the water.

A giraffe standing in a forest with **trees** in the background.

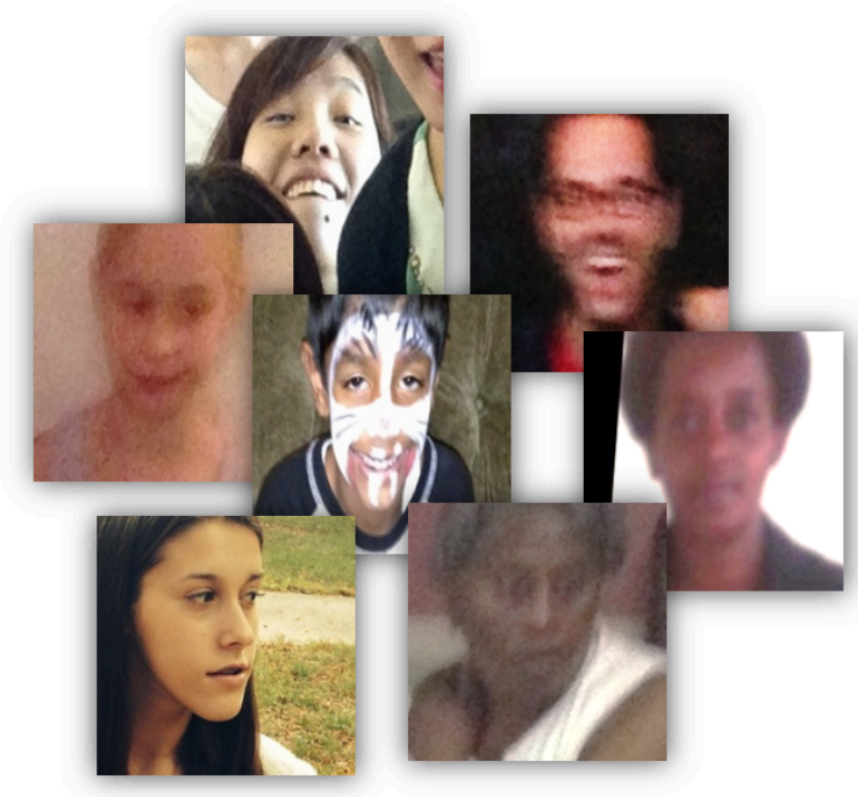
Figure 3 | From image to text. Captions generated by a recurrent neural network (RNN) taking, as extra input, the representation extracted by a deep convolution neural network (CNN) from a test image, with the RNN trained to 'translate' high-level representations of images into captions (top). Reproduced with permission from ref. 102. When the RNN is given the ability to focus its attention on a different location in the input image (middle and bottom; the lighter patches were given more attention) as it generates each word (bold), we found[86] that it exploits this to achieve better 'translation' of images into captions.

Age and Gender Classification using Convolutional Neural Networks

Gil Levi and Tal Hassner
Department of Mathematics and Computer Science
The Open University of Israel
gil.levi100@gmail.com hassner@openu.ac.il

Abstract

Automatic age and gender classification has become relevant to an increasing amount of applications, particularly since the rise of social platforms and social media. Nevertheless, performance of existing methods on real-world images is still significantly lacking, especially when compared to the tremendous leaps in performance recently reported for the related task of face recognition. In this paper we show that by learning representations through the use of deep-convolutional neural networks (CNN), a significant increase in performance can be obtained on these tasks. To this end, we propose a simple convolutional net architecture that can be used even when the amount of learning data is limited. We evaluate our method on the recent Adience benchmark for age and gender estimation and show it to dramatically outperform current state-of-the-art methods.



SPECIAL SECTION ON SOFT COMPUTING TECHNIQUES FOR IMAGE ANALYSIS IN THE MEDICAL INDUSTRY CURRENT TRENDS, CHALLENGES AND SOLUTIONS

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 22, 2017, accepted December 19, 2017, date of publication December 29, 2017, date of current version March 13, 2018.

Digital Object Identifier 10.1109/ACCESS.2017.2788044

Deep Learning Applications in Medical Image Analysis

JUSTIN KER[1], LIPO WANG[2], JAI RAO[1], AND TCHOYOSON LIM[3]

[1]Department of Neurosurgery, National Neuroscience Institute, 308433 Singapore
[2]School of Electrical and Electronic Engineering, Nanyang Technological University, 639798 Singapore
[3]Department of Neuroradiology, National Neuroscience Institute, 308433 Singapore

Corresponding author: Lipo Wang (elpwang@ntu.edu.sg)

This work was supported by the National Neuroscience Institute–Nanyang Technological University Neurotechnology Fellowship.

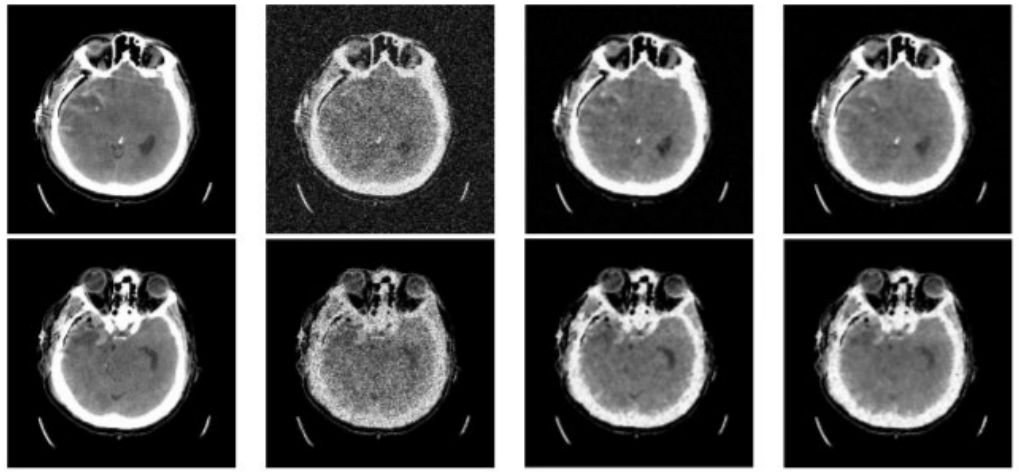
ABSTRACT The tremendous success of machine learning algorithms at image recognition tasks in recent years intersects with a time of dramatically increased use of electronic medical records and diagnostic imaging. This review introduces the machine learning algorithms as applied to medical image analysis, focusing on convolutional neural networks, and emphasizing clinical aspects of the field. The advantage of machine learning in an era of medical big data is that significant hierarchal relationships within the data can be discovered algorithmically without laborious hand-crafting of features. We cover key research areas and applications of medical image classification, localization, detection, segmentation, and registration. We conclude by discussing research obstacles, emerging trends, and possible future directions.

Adaptive Multi-Column Deep Neural Networks with Application to Robust Image Denoising

Forest Agostinelli **Michael R. Anderson** **Honglak Lee**
Division of Computer Science and Engineering
University of Michigan
Ann Arbor, MI 48109, USA
{agostifo,mrander,honglak}@umich.edu

Abstract

Stacked sparse denoising autoencoders (SSDAs) have recently been shown to be successful at removing noise from corrupted images. However, like most denoising techniques, the SSDA is not robust to variation in noise types beyond what it has seen during training. To address this limitation, we present the *adaptive multi-column stacked sparse denoising autoencoder* (AMC-SSDA), a novel technique of combining multiple SSDAs by (1) computing optimal column weights via solving a nonlinear optimization program and (2) training a separate network to predict the optimal weights. We eliminate the need to determine the type of noise, let alone its statistics, at test time and even show that the system can be robust to noise not seen in the training set. We show that state-of-the-art denoising performance can be achieved with a single system on a variety of different noise types. Additionally, we demonstrate the efficacy of AMC-SSDA as a preprocessing (denoising) algorithm by achieving strong classification performance on corrupted MNIST digits.



(a) Original (b) Noisy (c) Mixed-SSDA (d) AMC-SSDA

Figure 2: Visualization of the denoising performance of the Mixed-SSDA and AMC-SSDA. Top: Gaussian noise. Bottom: speckle noise.

ECE6524 – Deep Learning

- Advanced concepts in Machine Learning and Deep Learning. Models (multi-layer perceptrons, convolutional neural networks, recurrent neural networks, long short-term memory networks, memory networks), learning algorithms (backpropagation, stochastic sub-gradient descent, dropout), connections to structured predictions (Boltzmann machines, "unrolled" belief propagation), and applications to perception and Artificial Intelligence (AI)problems (image classification, detection, and segmentation; image captioning; visual question answering; automatic game playing).

- Students will:
  1. Analyze and contrast broad classes of deep learning models (multilayer perceptrons vs ConvNets vs RNNs)
  2. Derive and implement backprogation-based parameter learning and modern optimization techniques in such models
  3. Summarize and review state-of-art approaches in deep learning
  4. Discuss and critique research papers on these topics
  5. Identify open research questions in these areas