AA 274: Principles of Robotic Autonomy Winter 2018

Lecture 9: Machine Learning and Modern Visual Recognition Techniques

9.1 Introduction

This lecture provides an introduction to neural networks and proceeds to discuss Convolutional neural networks, which are especially suited to process images. With the advance of technology the amount of data available for any given task has gone up exponentially in the last several decades. Methods such as logistic regression perform well when the training data set is large. But their performance plateaus out after a point. Neural networks have gained popularity because with huge amounts of data which is currently available, even small neural networks are able to outperform traditional learning algorithms by large margins.

Image recognition and image processing has widespread applications in robotics such as path identification, obstacle avoidance etc. If we were to use a standard neural network architecture to process images, we would need to learn parameters of the order of millions. Convolutional neural networks have significantly brought down this number by using special convolution layers which exploits the spatial structure of images.

The core learning objectives of this lecture are:

1. The basics of neural networks :
   - How neural networks mimic the architecture of human brain and how multiple layers of the network work together
   - How to train a Neural network to do a particular task and where they could go wrong
2. Convolutional neural networks:
   - This section covers in depth the different elements of a Convolutional neural network and some famous architectures commonly used in the industry

9.2 Neural Network Basics

Work on Artificial neural networks, generally known as just 'neural networks', has been inspired by the fact that the human brain processes information in a completely different way than computers. Due to this, human brains are much more efficient at certain tasks, such as vision, language processing, etc. than computers. Artificial neural networks aim to mimic this biological model by employing a large number of simple interconnected processing units or 'neurons'. We may thus offer the following definition of a neural network viewed as an adaptive machine[1]:

A neural network is a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:
1. Knowledge is acquired by the network from its environment through a learning process.
2. Inter-neuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.

9.2.1 Perceptron - Analogy to a Biological Neuron

The figure below shows a simplified diagram of a biological neuron:

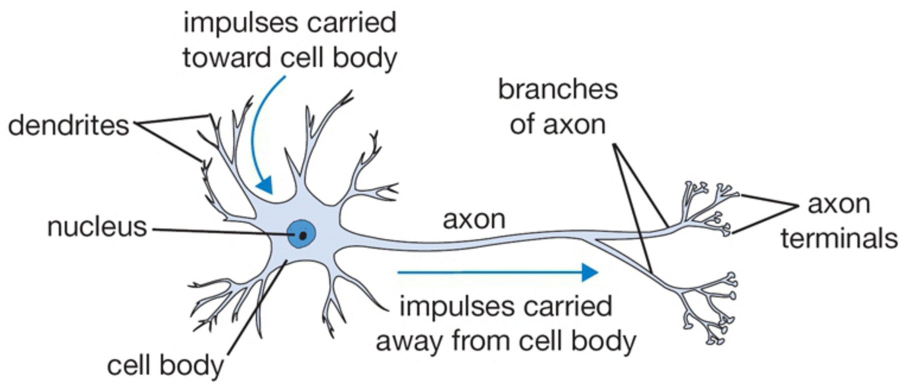

impulses carried toward cell body
dendrites
branches of axon
nucleus
axon
axon terminals
impulses carried away from cell body
cell body

Figure 9.1: Simplified model of a biological neuron[2]

There are approximately 86 billion neurons in the human nervous system and they are connected with approximately 10^{14} - 10^{15} synapses[2].

The figure below shows a commonly used mathematical model of a neuron:

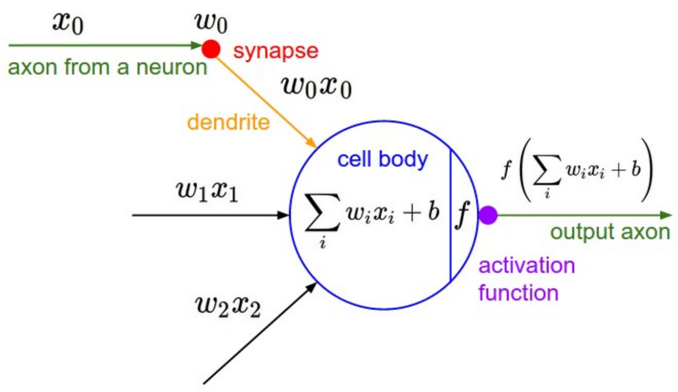
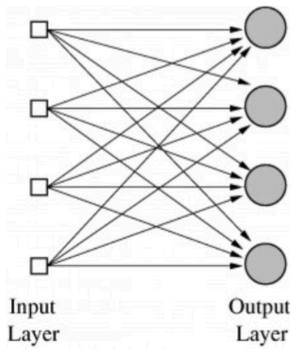


Figure 9.2: Mathematical model of a neuron[2]

Signals travel along the axons (e.g. x_0) and interact multiplicatively (e.g. $w_0 x_0$) with the dendrites of the other neuron based on the synaptic strength at that synapse (e.g. w_0). The idea is that the synaptic strengths (the weights w) are learnable and control the strength of influence (and its direction: excitatory (positive weight) or inhibitory (negative weight)) of one neuron on another. We can model the firing rate of the neuron with an activation function f, which represents the frequency of the spikes along the axon. Historically, a common choice of activation function was the sigmoid function, but the ReLU function is more commonly used nowadays. This is because the gradient of the sigmoid function becomes small as its value increases which reduces the training speed of the model.

9.2.2 Single Layer Network / Logistic Regression

Inspired by this model of the neuron, a single-layer neural net can be built which takes in binary inputs and provides binary output by taking a weighted sum and passing it through an activation function (as seen below). The theory is that if these weights are tuned perfectly, then it should be able to classify the inputs correctly.



$$y_1^i = f(x^i w_1 + b_1)$$
$$y_2^i = f(x^i w_2 + b_2)$$
$$y_3^i = f(x^i w_3 + b_3) \quad \Rightarrow \quad y = f(xW + b)$$
$$y_4^i = f(x^i w_4 + b_4)$$

Figure 9.3: Single layer neural net[?]

9.2.3 Multi-layer Neural Network / Deep Learning

We can get more resolution on our classifications by putting multiple of these layers together. In such regard, each layer can "learn" to classify a different part of the input. Take for an example the problem of classifying handwritten digits, in particular classifying the number 3. The first layer can be responsible for classifying a curved top, the second a curved bottom, etc.

A sample is shown below (from : http://www.cs.cmu.edu/ aharley/vis/conv/flat.html)

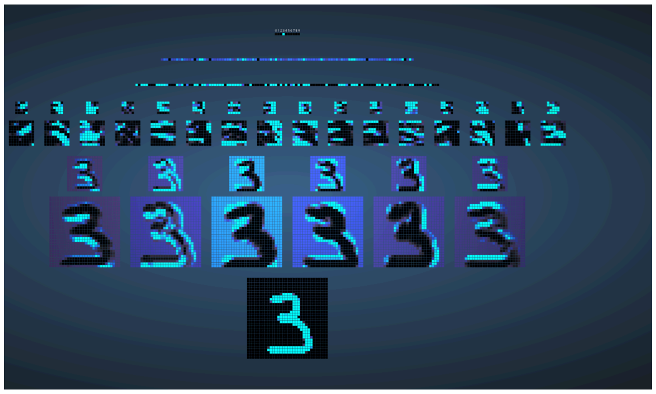
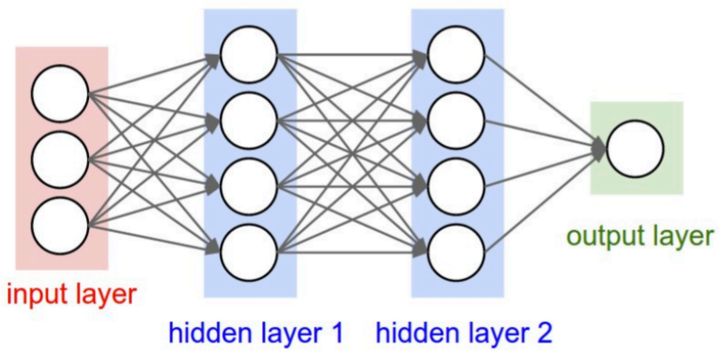


Figure 9.4: Deep Neural Net for Classifying Handwritten Numbers[]

Representing this is simply a bunch of single layer neural networks linked together where the output of 1 layer is the input of the second layer.



$$h_1 = f_1(xW_1 + b_1)$$
$$h_2 = f_2(h_1W_2 + b_2)$$
$$y = f_3(h_2W_3 + b_3)$$

Figure 9.5: Deep Learning[]

9.2.4 Activation Functions

If we didn't use non-linear activation functions, the output of the neural network would just be a linear function of the input. Such a network is basically just a Linear regression Model and and no matter how many layer and nodes we add to such a network, it is equivalent to having a network with a single node.

Hence, we use Activation functions to make the network more powerful give it the ability to represent non-linear complex functional mappings between inputs and outputs.

Another important feature of an Activation function is that it should be differentiable. This is because we perform back-propagation to calculate the gradients of the Loss function with respect to the weights and then accordingly optimize weights using gradient descend or any other Optimization technique to decrease the Loss Function.

The following are the most commonly used activation functions:

1. Sigmoid Function: The Sigmoid Function takes a real-valued number and maps it to a range between 0 and 1.

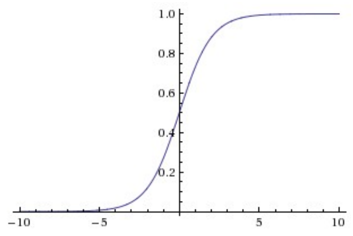


Figure 9.6: The sigmoid function[2]

$$f(x) = \frac{1}{1+e^{-x}}$$

Sigmoid Activation functions have historically been used extensively but have now fallen out of favor due to the following drawbacks:

   (a) They are prone to saturate as the gradients are very small away from the center. When this occurs, almost no signal will flow through the neuron to its weights and recursively to its data.

   (b) They are not zero-centered.

2. Tanh function: The Tanh function looks quite similar to a sigmoid function. It maps a real-valued number to a range between -1 and 1.

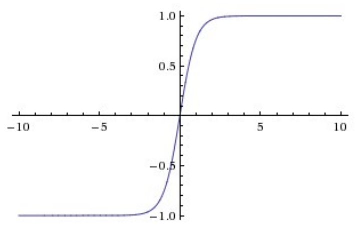


Figure 9.7: The tanh function[2]

$$f(x) = \tanh(x)$$

   It saturates in the same way as a sigmoid does but it has the advantage of being zero-centered. Hence, in practice, it is always preferred to a sigmoid function.

3. ReLU Function: The Rectified Linear Unit is simply a threshold at zero and has become very popular in the last few years.

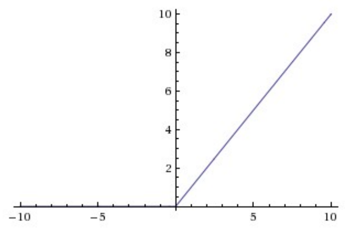


Figure 9.8: The ReLU function[2]

$$f(x) = max(0, x)$$

   It has the following advantages:

   (a) It greatly accelerates the convergence of stochastic gradient descent compared to tanh/sigmoid.
   (b) It is computationally cheaper to implement than tanh/sigmoid as it can implemented by simply thresholding a matrix of activations at zero.

   The main disadvantage of the ReLU is that ReLU units can irreversibly die during training since they can get knocked off the data manifold.

4. Leaky ReLU Function: The Leaky ReLU attempts to fix the problem of dying nuerons by providing a small negative slope in the $x < 0$ region.

$$f(x) = max(0.1x, x)$$

9.3 Training Neural Networks

We now need a way to teach these neural networks weights to make correct predictions. This is done by running the network on known data points and updating the weights of the neurons based on the correctness

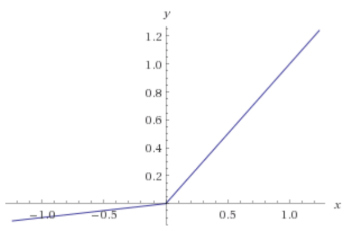


Figure 9.9: The Leaky ReLU function[2]

of the model in classifying the input. In order to update the weights, we need to select a loss function – a metric that tells us how well our network is doing.

9.3.1 Cross Entropy Loss

In classification settings, it is common to use the cross-entropy loss.

$$\ell_i = -\sum_{c=1}^{M} y_{i,c} \log p_{i,c} \tag{9.1}$$

ℓ_i is used to denote the loss on the ith training example. M is the number of classes, $y_{i,c}$ is an indicator which is 1 if image i belongs to class c, and p_c is our predicted probability of image i belong to class c. Note that this function is convex with respect to the class probabilities $p_{i,c}$.

In practice, you won't have to compute the cross entropy loss by hand – packages like Tensorflow and Pytorch have it built in.

9.3.2 Optimization Methods

Neural networks are trained using Stochastic Gradient Descent and a family of related algorithms. We denote the weights of our model as W and the input of the model as X. The total loss on the dataset is denoted as $\ell = \sum_i \ell_i(X, W)$ where the loss is a function of both the parameters and the data, and α is a parameter called the learning rate, which determines how much we update the weights at each step. The gradient descent update is given by:

$$W \leftarrow W - \alpha \frac{1}{N} \sum_{i=1}^{N} \nabla \ell_i(X_i, W) \tag{9.2}$$

In the case of Vanilla Gradient Descent the gradient $\nabla \ell(X, W)$ is computed by averaging the gradient across each individual training example before updating the weights. In practice, iterating over the whole dataset before making any updates to the weights is slow, and so we use Stochastic Gradient Descent, which performs an update of the weights for each training example:

$$W \leftarrow W - \alpha \nabla \ell_i(X_i, W) \tag{9.3}$$

Yet another variant of this method is called Mini-Batch Gradient Descent, in which we sample a small batch of the data, and average the gradient over this subset of the data before updating the weights.

There are several extensions to Gradient Descent which have been shown to converge faster in practice. They all use more or less the same idea: perform a weighted average over the gradient across several iterations. Of these, a method called Adam has become the preferred standard for neural network training. See [2] for information about why Adam outperforms Gradient Descent.

9.3.3 Overfitting

Creating complex models sometimes have the disadvantage of being extremely accurate only on the test data. That is, the model is able to classify its entire training data with high accuracy however is unable to generalize and fails with test / real data.

On the other hand too simple models are unable to learn the more subtle trends in the data.

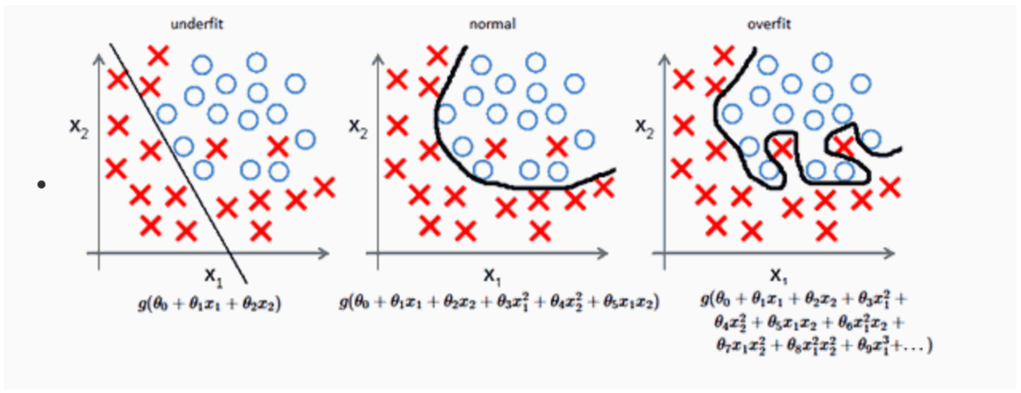

underfit
normal
overfit
x_2
x_1
$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$
$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$
$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2 + \theta_6 x_1^2 x_2 + \theta_7 x_1 x_2^2 + \theta_8 x_1^2 x_2^2 + \theta_9 x_1^3 + \dots)$

Figure 9.10: Overfitting[]

http://mlwiki.org/index.php/Overfitting

In the example above the left model was too simple and was unable to classify some of the values correctly. On the other the right model despite perfectly classifying on this training dataset is unlikely to perform as well in real-world situations. The middle offers a good classification.

To combat overfitting, regularization is usually introduced in models. Regularization is a technique that penalizes complex models or prevents them from being made. Some of these techniques include

Dropout Randomly select neurons that will not be activated at each pass during training.

L2 Regularization Penalize the square magnitude of all parameters which forces neuron size to be smaller.

Max Norm Constraints Limit the maximum weight of a neuron.

9.4 Convolutional Neural Networks

A simple Convolutional Neural Networks is a sequence of layers, where every layer of a Convolutional Neural Networks transforms one volume of activations to another through a differentiable function. There are four main types of layers to build ConvNet architectures: Convolutional Layer, Nonlinearity Layer, Pooling

Layer, and Fully-Connected Layer. These layers are stacked to form a full ConvNet architecture. In this way, Convolutional Neural Networks transform the original image layer by layer from the original pixel values to the final class scores.

In this section, we mainly discuss the three types of layer in Convolutional Neural Networks: the Convolutional layers, the Pooling layers and the Fully-connected layers.

9.4.1 Convolutional Layer

Parameter Sharing Parameter sharing scheme is used in Convolutional Layers to control the number of parameters. It is hard to make perceptrons scalable when the input image size is large given the fact the number of parameters grow quickly with the input size. However, we can dramatically reduce the number of parameters by making one reasonable assumption. If we know the input is image data, we can assume some spatial symmetries. In other word, if one feature is useful to compute at some spatial position (x, y), then it should also be useful to compute at a different position (x_2, y_2), so same parameters could be used.

Convolution Details The convolution operation essentially performs dot products between the filters and local regions of the input. In other words, each element is computed by element-wise multiplying the local regions of the input (blue part in Figure 9.11) with the filter (red part in Figure 9.11), summing it up, and then offsetting the result by the bias. The output is the green part in Figure 9.11.

Noticing we give the demo in 2D format. In real cases, the input would be a 3D tensor while the depth of the output volume is a hyperparameter. It corresponds to the number of filters we would like to use, each learning to look for something different in the input. Take Figure 9.12 as an example, if we have 6 5×5 filters, we will get 6 separate activation maps as output.

Spatial arrangement Besides depth, there are two hyperparameters to control the size of the output volume: the stride and zero-padding

- stride Stride specifies how we slide the filter. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2 (or uncommonly 3 or more, though this is rare in practice) then the filters jump 2 pixels at a time as we slide them around.

- zero-padding In some cases, the size of filters and stride don't "fit" neatly and symmetrically across the input. For example, if the input size $H = W = 10$ while the filter size is $F = 3$ and it takes stride $S = 3$, it would be impossible to use stride $S = 2$, since

$$\begin{aligned}(H - F)/S + 1 = (10 - 3)/3 + 1 = 3.33 \text{ (Not an integer)}\\ (W - F)/S + 1 = (10 - 3)/3 + 1 = 3.33 \text{ (Not an integer)}\end{aligned} \tag{9.4}$$

  Therefore, we need to use zero-padding. We add additional zero-paddings in the contour of the image. If we take zero-padding $P = 1$, we would have

$$\begin{aligned}(H - F + 2 * P)/S + 1 = (10 - 3 + 2 * 1)/3 + 1 = 4 \text{ (An integer)}\\ (W - F + 2 * P)/S + 1 = (10 - 3 + 2 * 1)/3 + 1 = 4 \text{ (An integer)}\end{aligned} \tag{9.5}$$

In general, setting zero padding to be $P = (F1)/2$ when the stride is $S = 1$ ensures that the input volume and output volume will have the same size spatially. It is very common to use zero-padding in this way.

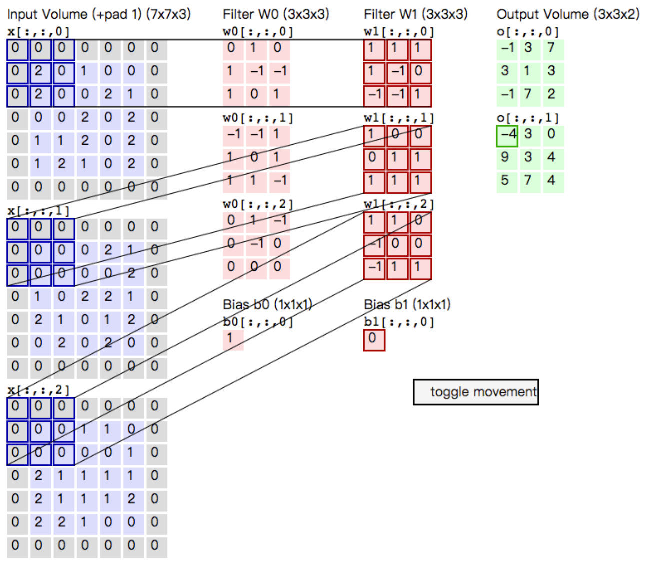


Figure 9.11: Convolution Demo[2]

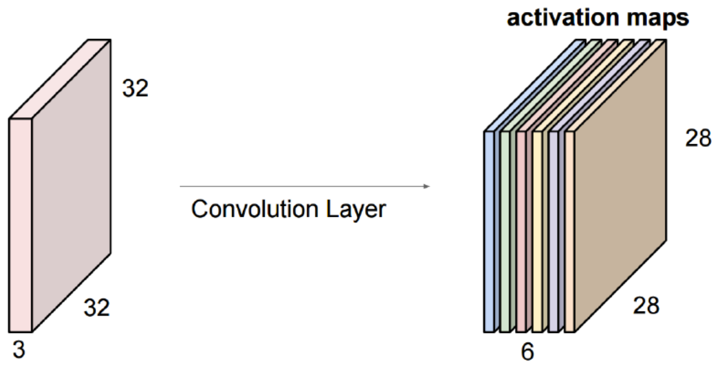
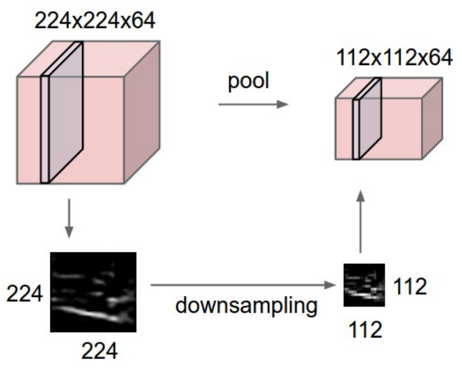


Figure 9.12: Number of filter and depth of output[2]

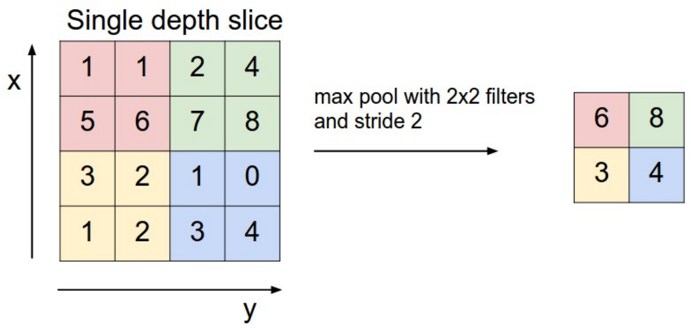
9.4.2 Pooling Layer

In practice, it is common to periodically insert a Pooling Layer in-between successive Conv layers in a ConvNet architecture. It is mainly used to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation load, and hence to also control overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation if max pooling, AVG if average pooling, etc. The most common form is a pooling layer with filters of size 2×2 applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations. In this case, every pooling operation would be taking a max or average over 4 numbers (little 2×2 region in some depth slice). The depth dimension remains unchanged. In a nutshell, pooling layer downsamples the volume spatially, independently in each depth slice of the input volume.

The Fig.9.13a shows how pooling works generally. A 2×2 pooling filter with stride 2 is applied on the $224 \times 224 \times 64$ input volume, yielding the $112 \times 112 \times 64$ output volume. Note that the width and height of



(a) general pooling



(b) max pooling

Figure 9.13: Pooling Layer[2]

the input volume shrinks to half while the depth is preserved. The most common downsampling operation is MAX. The Fig.9.13b here shows max pooling with a stride of 2. That is, each max is taken over 4 numbers in a 2×2 square. For instance, the top-left red square squashes to a single value, `6`, which is `max{1, 1, 5, 6}`.

9.4.3 Fully-Connected Layer

The last few layers of a ConvNet architecture are typically Fully-Connected (FC) Layers. As seen in regular neural networks, FC Layers serve as a linear classifier for classification or regression. The FC Layers takes the ”summary vectors” of the images, which are vectors with only the basic information needed to recreate the image, and feed them into the image classifier to obtain the final classification.

Layer.jpg Layer.bb

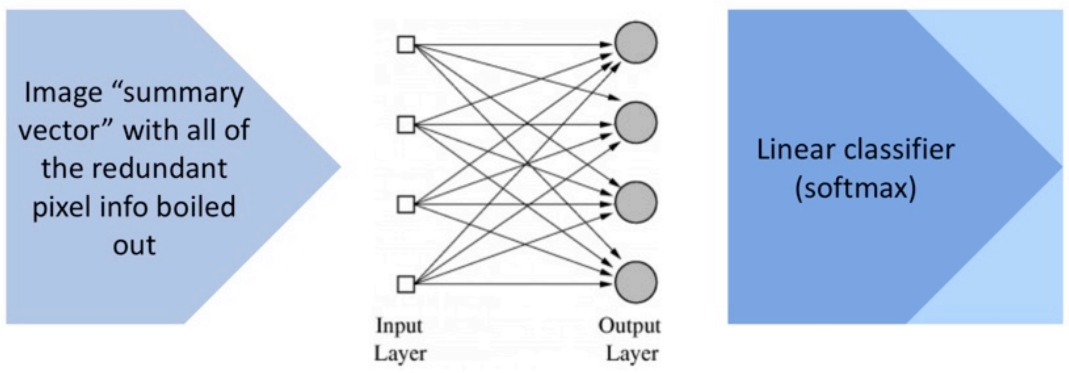


Image “summary vector” with all of the redundant pixel info boiled out
Input Layer
Output Layer
Linear classifier (softmax)

Figure 9.14: Fully-Connected Layer[2]

The activations of the classifier are computed with a matrix multiplication followed by a bias offset, i.e. $f = Wx + b$. Where the weights, W, of the biases are formed while classifying the known training dataset. The output of the FC Layers is typically a vector representing the probablitities calculated of each classification, with the final classification being the highest calculated probability. See the Neural Network Basics section of the note for more information.

9.4.4 Mainstream Model Architectures

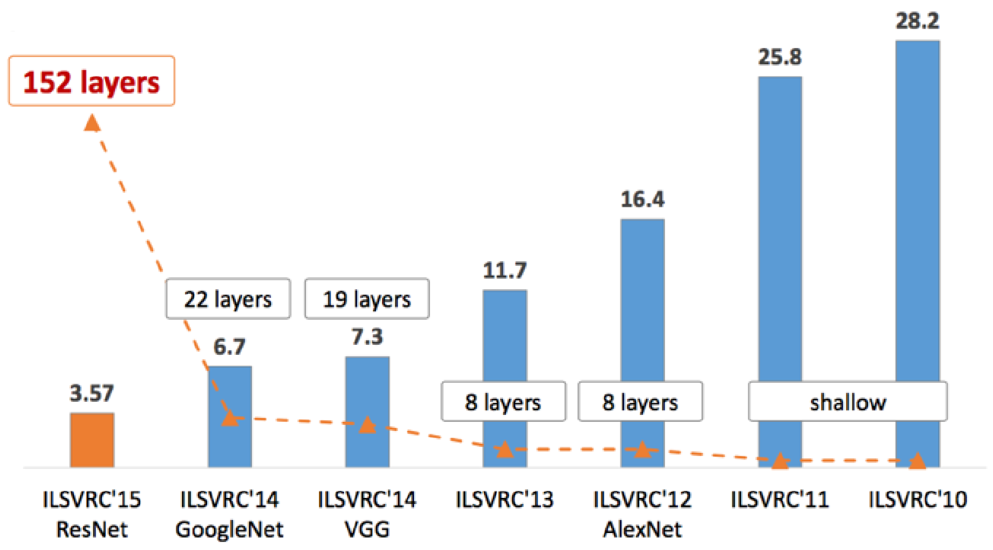


Figure 9.15: Comparison of performance of different image classification models in the ImageNet Large Scale Visual Recognition Competition (ILSVRC). Values indicate top-5 classification error on test datasets (how often the correct class did not appear in the top 5 outputted classes for new images).

Empowered with impressive ability of feature extraction and optimization, deep convolutional neural networks has become the cornerstone of data-driven visual recognition techniques nowadays. Since AlexNet's overwhelming success in the ImageNet Large Scale Visual Recognition Competition (ILSVRC) 2012, a number of milestone model architectures have been proposed. Some common ones are described below:

- AlexNet[3]: Developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton. The AlexNet significantly outperformed the second runner-up (top 5 error of 16% compared to runner-up with 26% error) ILSVRC challenge in 2012. It is the first work that popularized Convolutional Neural Networks in Computer Vision.

  AlexNet had a similar structure to a pivotal network proposet by LeCun et al in 1998 for classifying digits[4]. It invokes convolutional, max pooling, and fully connected layers, also invoking ReLU activations with dropout (Figure 9.16).

- VGGNet[5]: Proposed by Karen Simonyan and Andrew Zisserman. The VGGNet was the runner-up in ILSVRC 2014. Its main contribution was in showing that the depth of the network is a critical component for good performance. Researchers and engineers tend to design deeper rather than shallower models ever since.

  VGGNet (Figure 9.17 is similar to AlexNet, only with more filters and more layers. The weight configuration of the VGGNet is publicly available and can be used in other applications as a feature extractor.

- GoogLeNet[6]: As its name shows, it is authored by Szegedy et al. from Google. GoogLeNet (Figure 9.19) was the ILSVRC 2014 winner with main contribution in developing Inception Module that dramatically reduced the number of parameters in the network. The Inception Modules (Figure 9.18) serve to approximate a sparse CNN with a normal dense construction. This greatly reduces the number

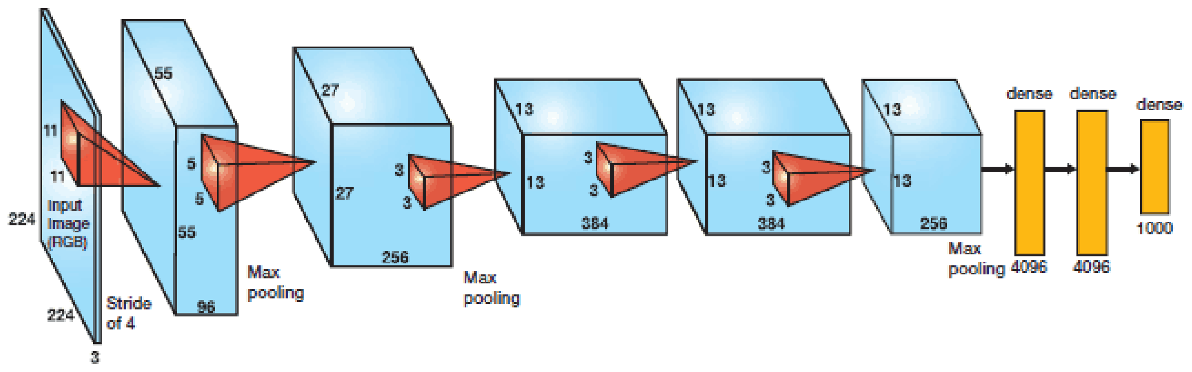


Figure 9.16: AlexNet model architecture

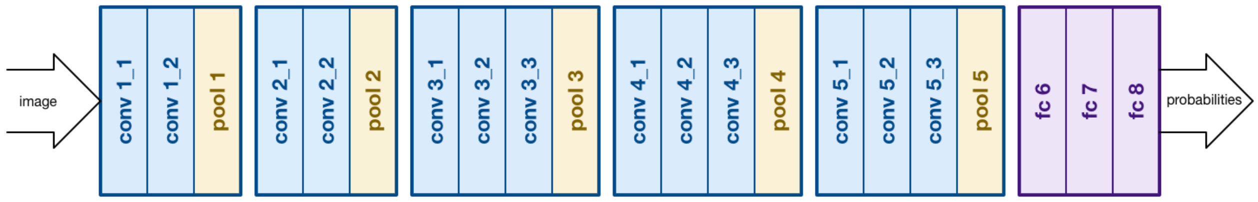


Figure 9.17: VGGNet model architecture

of parameters required to optimize over (noting that many activations hence parameters will be heavily correlated). These Inception Modules massively reduced computation requirements.

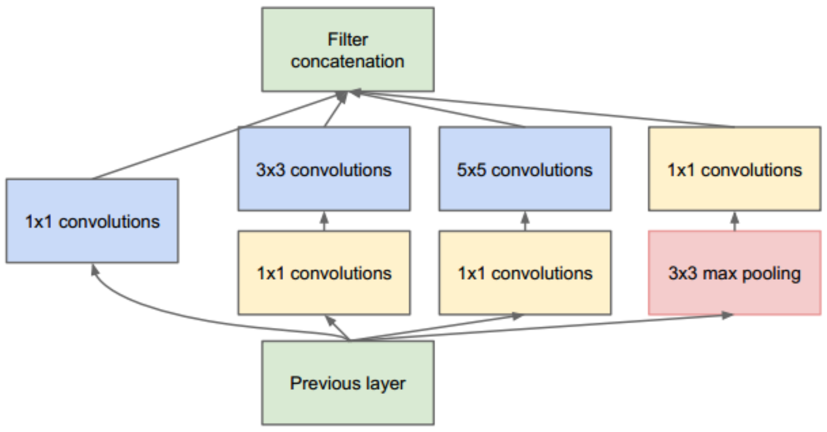


Figure 9.18: Single Inception Module

- ResNet[7]: Developed by Kaiming He et al. ResNet was the winner of ILSVRC 2015. When multiple models were ensembled together, ResNet achieved a top-5 error of 3.57%, which is astounding. ResNet implemented heavy use of batch normalization, and special skip connections now known as Residual Modules (Figure 9.20). CNNs have a big problem of vanishing gradients, that is, as we move farther away from our output/loss end and more towards our data, less and less gradient information gets

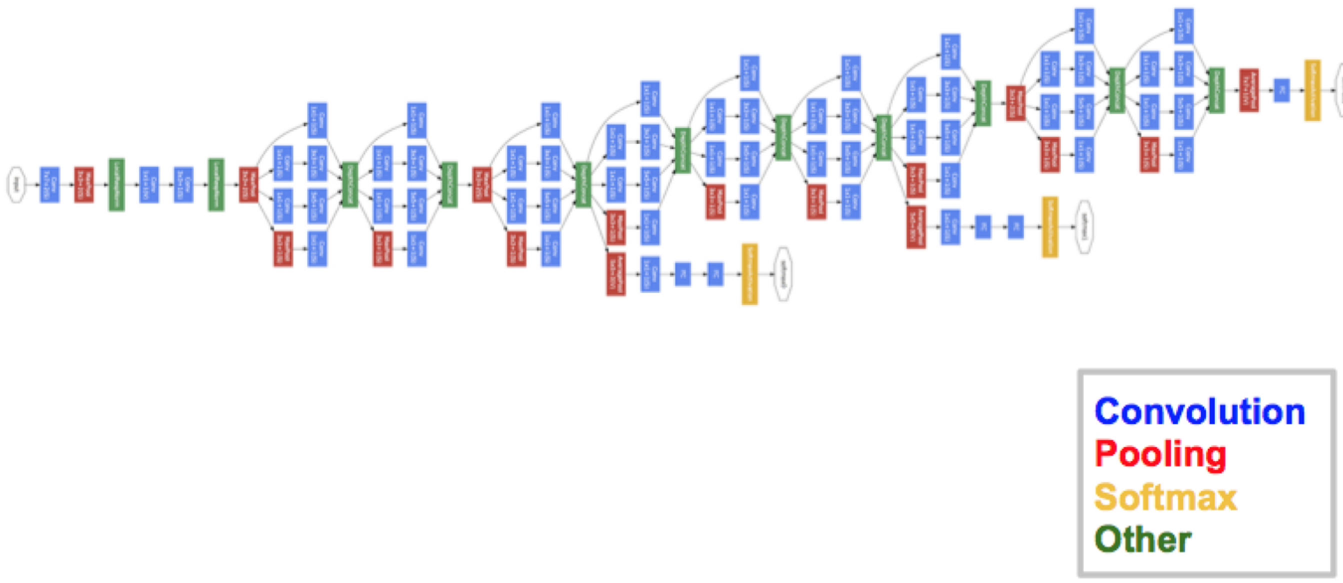


Figure 9.19: GoogLeNet model architecture

propagated. Residual modules allay this by appending lower layers with higher ones, so that the gradients can skip operations.

When multiple models are ensembled together, ResNet achieves a top-5 classification error of 3.57%, which is astounding (some say this to be better-than-human accuracy). Still, the top-1 (true) classification accuracy for ResNet was roughly 80% and has been slightly outperformed by later Inception-based models. Analysis of the evolution of these models in terms of their top-1 classification accuracy can be seen in Figure 9.21.

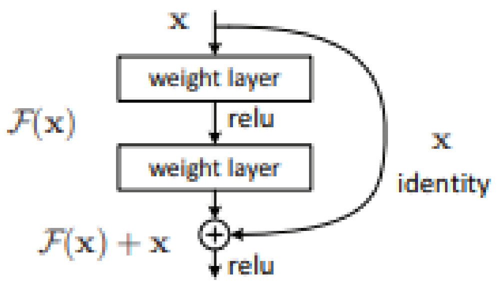


Figure 9.20: Residual Module[7]

There also exist more model architectures proposed for specific tasks, such as YOLO[8] and Faster R-CNN[9] in the field of object detection and localization.

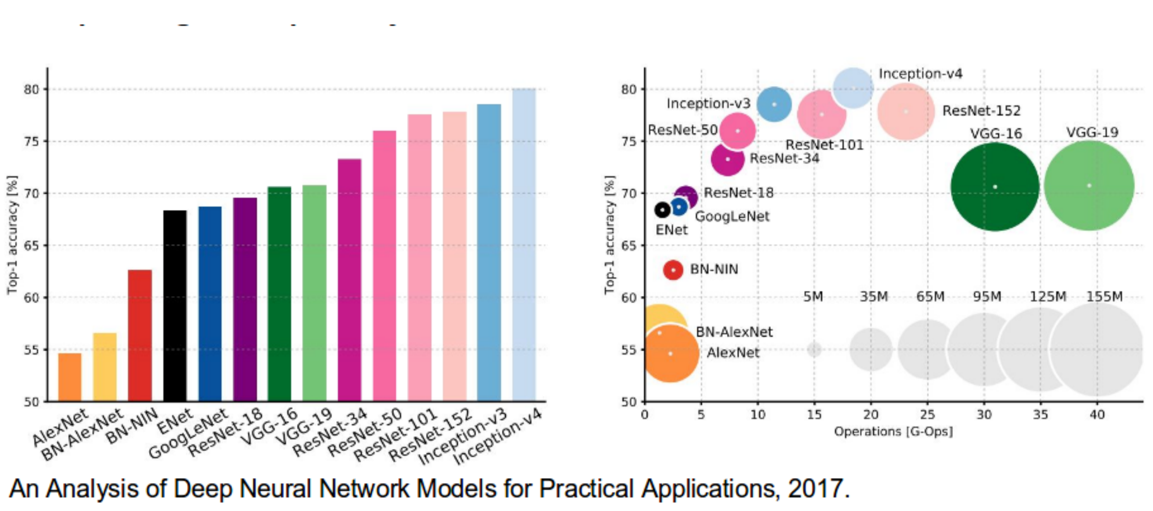


Figure 9.21: Top-1 classification accuracy of different model architectures

9.5 Object Detection Algorithms

9.5.1 R-CNN[10]

The goal of R-CNN is to take in an image, and correctly identify where the main objects (via a bounding box) in the image. The network input is image and the outputs are bounding boxes + labels for each object in the image. In summary, R-CNN is follows the steps below:

1. Generate a set of proposals for bounding boxes.
2. Run the images in the bounding boxes through a pre-trained AlexNet and finally an SVM to see what object the image in the box is.
3. Run the box through a linear regression model to output tighter coordinates for the box once the object has been classified.

In terms of the bounding boxes selection, Ross Girshick et al. proposed a method[10] where we use selective search to extract just 2000 regions from the image and he called them region proposals. Therefore, now, instead of trying to classify a huge number of regions, you can just work with 2000 regions. These 2000 region proposals are generated using the selective search algorithm which is written below.

1. Generate initial sub-segmentation, we generate many candidate regions
2. Use greedy algorithm to recursively combine similar regions into larger ones
3. Use the generated regions to produce the final candidate region proposals

So far, we have seen how RCNN can be helpful for object detection. But this technique comes with its own limitations. Training an RCNN model is expensive and slow thanks to the below steps:

1. Extracting 2,000 regions for each image based on selective search

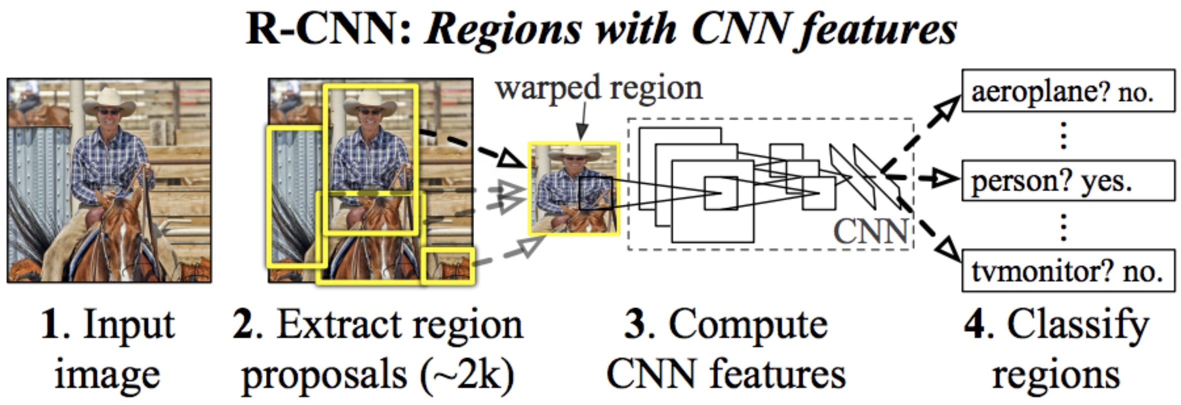


Figure 9.22: R-CNN Algorithm

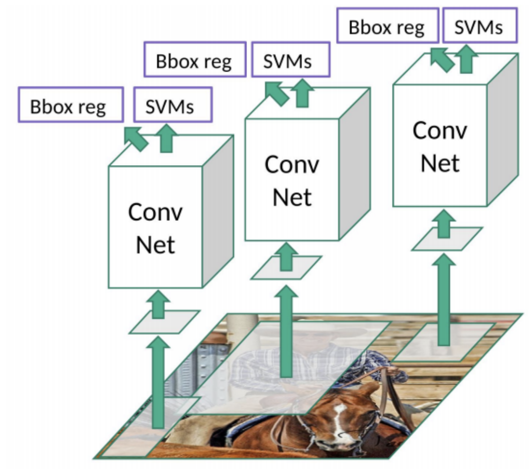


Figure 9.23: R-CNN Algorithm

2. Extracting features using CNN for every image region. Suppose we have N images, then the number of CNN features will be N*2,000

3. The entire process of object detection using RCNN has three models:

   (a) CNN for feature extraction

   (b) Linear SVM classifier for identifying objects

   (c) Regression model for tightening the bounding boxes.

9.5.2 YOLO[8]

YOLO (You Only Look Once), is a network for object detection. The object detection task consists in determining the location on the image where certain objects are present, as well as classifying those objects. Previous methods for this, like R-CNN and its variations, used a pipeline to perform this task in multiple steps. This can be slow to run and also hard to optimize, because each individual component must be trained separately. YOLO, does it all with a single neural network. A image is passed through a neural network that looks similar to a normal CNN, but you get a vector of bounding boxes and class predictions in the output. The output is shown in figure 9.24 and described below:

- center of a bounding box (bx, by)
- width (bw)
- height (bh)
- value c is corresponding to a class of an object (f.e. car, traffic lights,...).

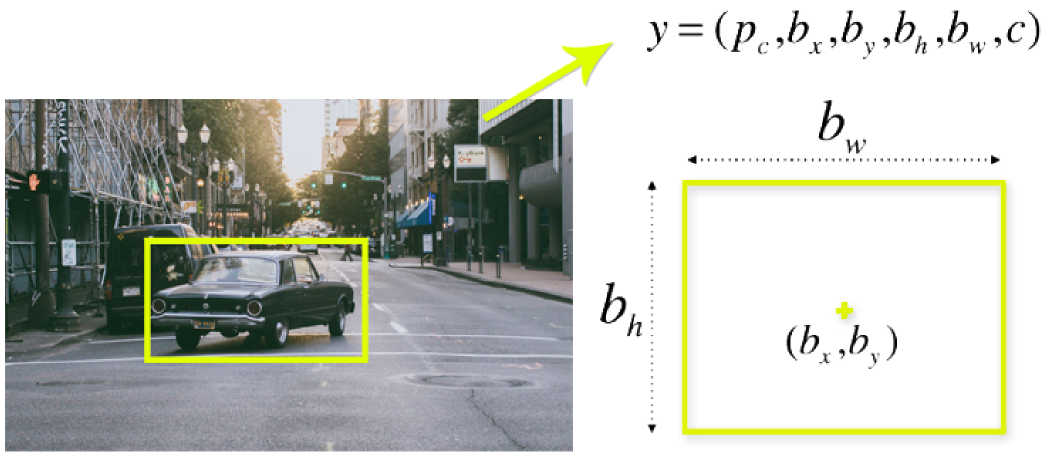


Figure 9.24: YOLO Output

Firstly, we are splitting our image into cells, typically its 19×19 grid. Each cell will be responsible for predicting 5 bounding boxes in case there are more than one object in this cell. This will give us 1805 bounding boxes for an image. This is illustrated in figure 9.25.
Then we have a deep CNN taking in the input image and spit out the output in the dimension of $(19, 19, 5, 5 + 80)$, where 80 is the number of classes. This is illustrated in figure 9.25.
Majority of those cells and boxes won't have an object inside and this is the reason why we need to predict p_c. In the next step, we're removing boxes with low object probability and bounding boxes with the highest shared area in the process called non-max suppression, which is shown in figure 9.26.
YOLO has several advantages over classifier-based systems. It looks at the whole image at test time so its predictions are informed by global context in the image. It also makes predictions with a single network evaluation unlike systems like R-CNN which require thousands for a single image. This makes it extremely fast, more than 1000x faster than R-CNN and 100x faster than Fast R-CNN.

9.6 Applications of Machine Learning in Industry

To the reader that is being exposed to machine learning for the first time, it might be helpful to know what types of problems have been solved through the use of machine learning. Understanding the existing problem domain in which machine learning operates may inspire the reader to extend the use of methods to other domains. This section highlights some examples from recent literature.

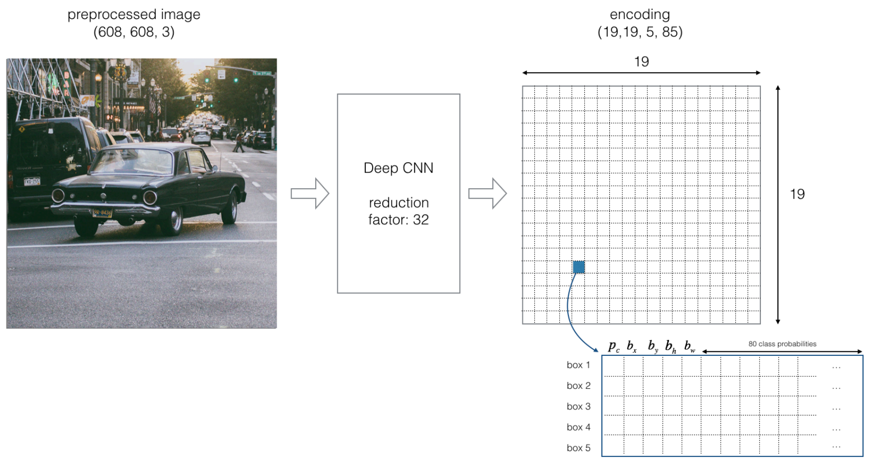


Figure 9.25: YOLO: Split the Image into cells

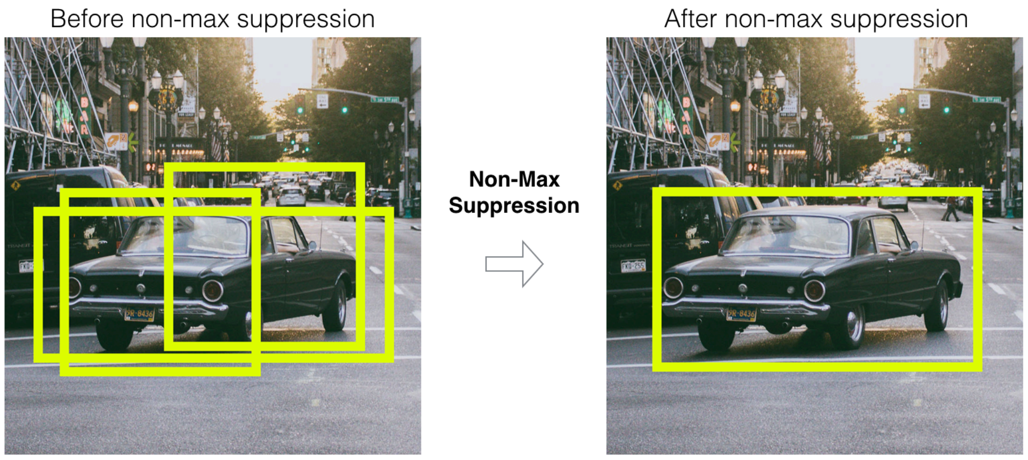


Figure 9.26: YOLO: Non-max suppression

- On-board down-selection of data for spacecraft telemetry. One of the challenges faced by space mission designers is the volume of remote data (telemetry) that can be downlinked for examination by a human operator on the ground. Given this relatively hard physical constraint, it is useful to perform data processing on-board the spacecraft and filter the entire dataset to a (predictably) useful set of telemetry packages for downlink. The Intelligence Payload Experiment (IPEX), a CubeSat that operated on Low Earth Orbit from 2013 to 2015, utilized machine learning to accomplish this on-board autonomy task [11]. The satellite performed on-board feature extraction and image classification over the Earth images obtained from its camera. The system was trained with a small sample of data from a suborbital flight, and classified orbital images based on its predictions of snow, water, land, haze, or cloud content. The system utilized random forest classifiers (from the TextureCam suite), which advanced previous work through enhanced analysis on spatial neighborhoods and increased regularization due to using multiple ensembles of decisions trees. This is a notable example for the use of machine learning in robotic systems with limited computational resources.

- Unsupervised learning algorithm to estimate scene depth and robot pose from monocular videos. As studied earlier in this course, the estimation of scene depth is one of the most important outputs of computer vision in the context of autonomous robotics. Furthermore, a designer may be interested in reducing the number or complexity of sensors required to obtain a depth estimation. Zhou et al. [12] provides a framework for using unsupervised learning from a monocular video to predict depth and pose. The implementation utilizes two parallel networks (one for ego-motion modelling and another one for depth modelling), which are coupled during the calculation of the training loss, but can operate independently during test time. Empirical testing of these algorithms has demonstrated results on par, or higher in the case of pose modelling, than similar supervised methods.

- Reduced dynamic models for early earthquake prediction. While machine learning is typically introduced in the context of computer vision and image feature extraction, it is important to remember that the framework can be used to analyze other kinds of signals. In 2018, a group of researchers at Los Alamos National Lab utilized machine learning to create a model to analyze correlations between two distinct types of geological measurements [13]. They constructed a machine learning model to explain readily observable data (like GPS displacement rates) as a function of deeper seismic dynamics, which typically require more sensors and time to describe accurately. To train this random forest prediction model, statistical characteristics of the seismic signal dataset were used as features and measured GPS displacement rate data as labels. According to the authors, this model could provide indirect real-time access to fault physics on deeper portions of the the Earth, thus proving useful in early detection of seismic events that could lead to a major earthquake.
- Enhanced catalog classification in the Mars rover image catalog. Mars rovers and orbiters have produced more than 22 million images of Mars over the course of history. Previously, classification of these images on public databases has been limited to a-priori knowledge of image properties (e.g. camera angle, date, time, etc.), from which the user would have to infer the set of classifiers that would produce a set of features (i.e. if an engineer is looking for images of rover wheels to analyze wheel deterioration, select images that had camera targets pointed at wheels). In 2018, Wagstaff et al. [14] described the use of machine learning for feature extraction from images in databases. This resulted in expanding the set of feature class samples that would have been otherwise too costly to examine and classify through human means. Of particular interest, this model showed that a convolutional neural network trained with Earth images can be retrained to successfully classify images from Mars. It used transfer learning to adapt the AlexNet image classifier (trained on Earth dataset) via removal of the final fully connected layer, redefinition of output classes, and retraining using Caffe on the Mars image dataset.

9.7 Additional Resources

1. CS231n (CNNs for Visual Recognition) class notes : http://cs231n.github.io/convolutional-networks/
2. Live Demo - Inner Workings of a CNN : http://scs.ryerson.ca/ãharley/vis/conv/
   2D version : http://scs.ryerson.ca/ãharley/vis/conv/flat.html
3. CS230 : https://cs230-stanford.github.io/
4. Tensorflow documentation : https://www.tensorflow.org/api_docs/python/tf

References

[1] Simon S. Haykin. Neural Networks and Learning Machines. Prentice Hall, 2009.

[2] Andrej Karpathy. Stanford university cs231n course notes, May 2016. Accessed: 2018-02-07.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 25, pages 1097–1105. Curran Associates, Inc., 2012.

[4] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[5] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556, 2014.

[6] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In Computer Vision and Pattern Recognition (CVPR), 2015.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. CoRR, abs/1512.03385, 2015.

[8] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. CoRR, abs/1506.02640, 2015.

[9] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, Advances in Neural Information Processing Systems 28, pages 91–99. Curran Associates, Inc., 2015.

[10] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. CoRR, abs/1311.2524, 2013.

[11] David R. Thompson Kiri L. Wagstaff John Bellardo Craig Francis Eric Baumgarten Austin Williams Edmund Yee Eric Stanton Steve Chien, Joshua Doubleday and Jordi Piug-Suari. Onboard autonomy on the intelligent payload experiment cubesat mission. Journal of Aerospace Information Systems, 14, 2017.

[12] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G. Lowe. Unsupervised learning of depth and ego-motion from video. In CVPR, 2017.

[13] Bertrand Rouet-Leduc, Claudia Hulbert, and Paul A Johnson. Continuous chatter of the cascadia subduction zone revealed by machine learning. Nature Geoscience, 12(1):75, 2019.

[14] Kiri L Wagstaff, You Lu, Alice Stanboli, Kevin Grimes, Thamme Gowda, and Jordan Padams. Deep mars: Cnn classification of mars imagery for the pds imaging atlas. In Conference on Innovative Applications of Artificial Intelligence., 2018.

Contributors

Chi Zhang, Honghao Wei, Matthew Tan, Taylor Howell, Brian Jackson, Saifan Rafiq, Kevin Okseniuk