Introduction to Deep Learning

Convolutional Neural Network for Object Classification

John Luo

11/19/2018

Machine Learning?

- What does machine learning do?
- Tasks: **Classification, Regression, Transcription, Translation, Denoising**, etc.
- What is ML? Train a machine algorithm to do the above tasks.

1. Start with **representations** of training data and model parameter.
2. Training by **iterating** between mathematical **optimization** step and **evaluation** (minimizing training error).
3. Apply trained algorithm onto test data (hoping the test error is small enough to generalize)

Some Optimization Concepts Revisited

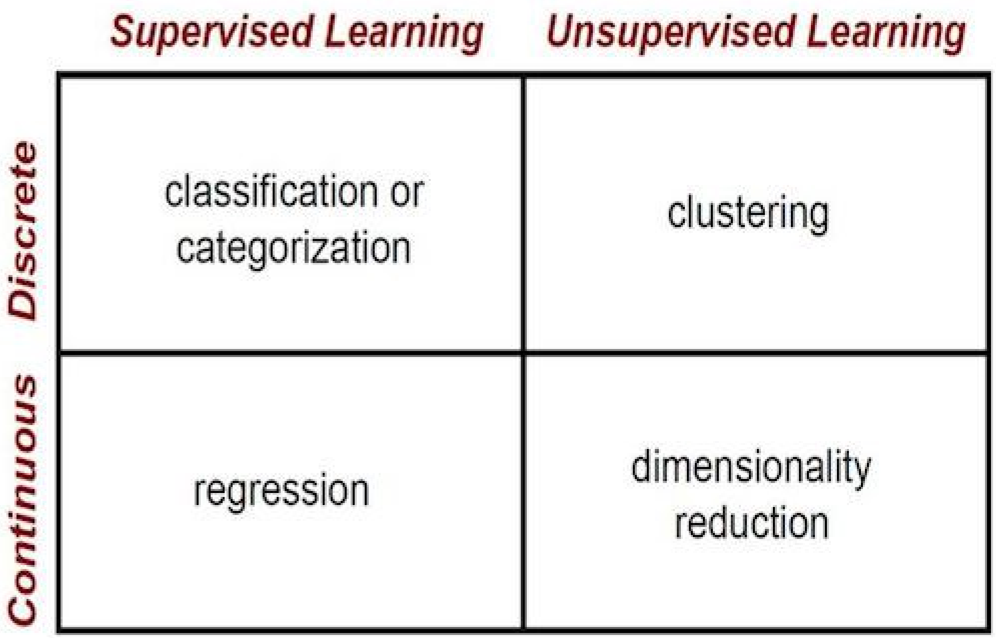
- Bayesian statistics/**Baye's Rule**
- Maximum a posteriori (MAP) estimation

$$\boldsymbol{\theta}_{map} = \arg\max \log p(\boldsymbol{\theta}|\boldsymbol{x}) = \arg\max \ \log p(\boldsymbol{x}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$$

- Minimizing cost/lost function: such as the negative log probability density function above or any arbitrarily chosen function with regularization
- Some Regularizations:
  1. L2 regularization = Gaussian Prior
  2. L1 regularization = Laplace Prior
- **Gradient Descent Optimization**

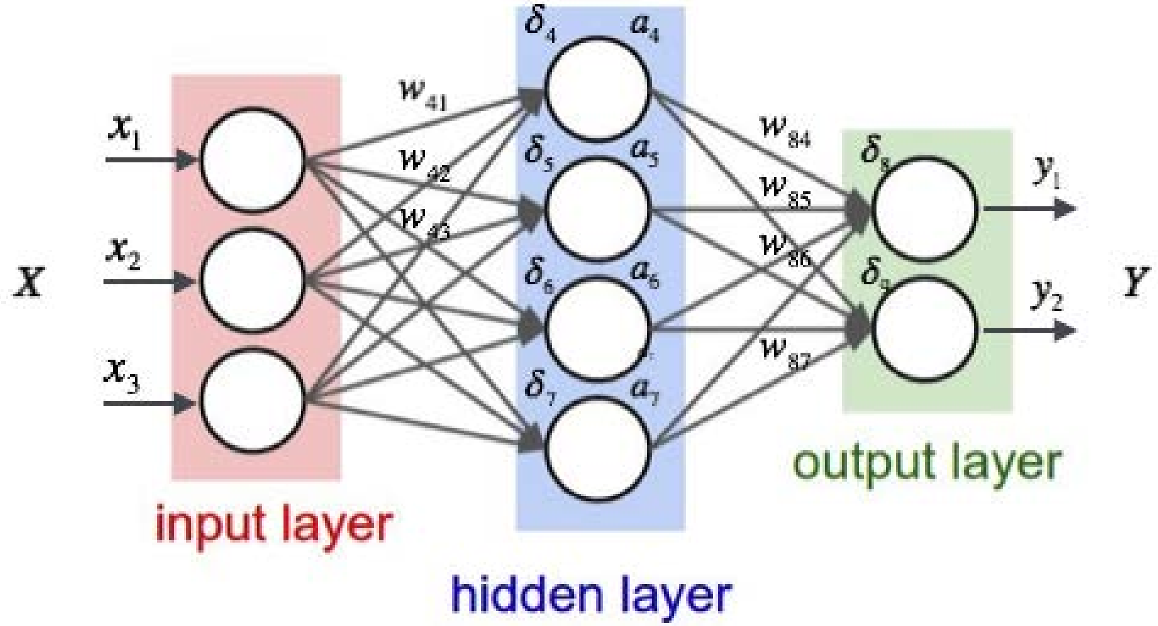
Some ML Concepts

- **Supervised** Learning: training data is already tagged by human supervisors (classification and regression tasks).
- **Unsupervised** Learning: some underlying insight identified directly by machine learning programs (clustering, principal component analysis).
- We will focus on object classification using deep neural network as a case study on how deep learning works



Simple Neural Network Architecture

- On the right, NN of 1-hidden layer
- $\boldsymbol{x}^i$ has 3 inputs (3 x1), $\boldsymbol{W}^h$ is 3 x 4 matrix of weights connecting layers of neural nodes
- ${\boldsymbol{W}^i}^T \boldsymbol{x}^i = \boldsymbol{\delta}^h$
- Some activation function, g^h gives a pointwise operation: $g^h(\boldsymbol{\delta}^h) = \boldsymbol{\alpha}^{\mathrm{h}}$
- To compute output layer, we have a matrix $\boldsymbol{W}^o$ which is 4x2
- ${\boldsymbol{W}^o}^T \boldsymbol{a}^h = \boldsymbol{\delta}^o$
- Another activation function, g^o, a pointwise operation: $g^o(\boldsymbol{\delta}^o) = \boldsymbol{y}^{\mathrm{o}}$



Neural Network Basics

- Feedforward neural networks (often called multilayer perceptrons (MLPs)), each layer has linear transformation followed by an activation, 1st and 2nd layers:

$$\boldsymbol{h}^1 = g^1\left(\boldsymbol{W}^{1^T}\boldsymbol{x} + \boldsymbol{b}^1\right)$$
$$\boldsymbol{h}^2 = g^2(\boldsymbol{W}^{2^T}\boldsymbol{h}^1 + \boldsymbol{b}^2)$$

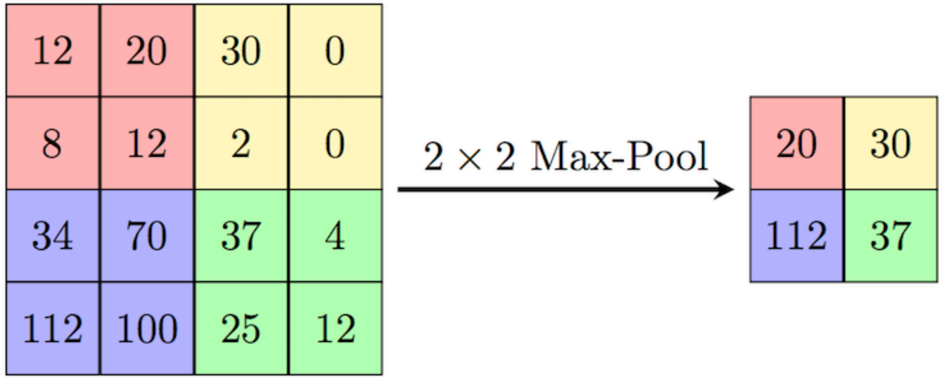
- **W** represents **linear weights** assigned to each input, **b** represents **bias**, g is an **activation** function (can be nonlinear, see ReLU).
- Commonly used g are:
  - **Sigmoid** for binary classification (for Bernoulli output distributions):
  $$g(z) = \frac{1}{1 + e^{-z}}$$
  - **Softmax** for multiple classification (for multinoulli output distributions):
  $$g(z)_i = \frac{\exp z_i}{\sum_j \exp z_j}$$
  - **Rectified linear units (ReLU)**: $g(z) = \max\{0, z\}$, most widely used
  - Tangent activation function: $g(z) = \tanh(z)$

Chain Rule & Backpropagation

- Chain rule is the key to **backpropagate error** and **update each layer's parameter $\boldsymbol{W_i}$ and $\boldsymbol{b_i}$**
- The iterative update on W_i and b_i normally use a **gradient descent approach** with hyperparameters such as regularization (added to the gradients calculated) where needed and learning rate (step size of the gradient descent)
- Stochastic gradient descent (each sample assumed I.I.D, update parameters one sample at a time) is the most widely used, also full GD (use all samples) and batch GD (use a batch of sample to update)
- Regularization: L1, L2 or arbitrary
- Also check out enhanced gradient descent algorithms such as Nestorov momentum and RMSprop (for example, stochastic GD with RMSprop)
- Optimal hyperparameters can be determined using **grid search** or **random search** (a time-consuming guessing game)

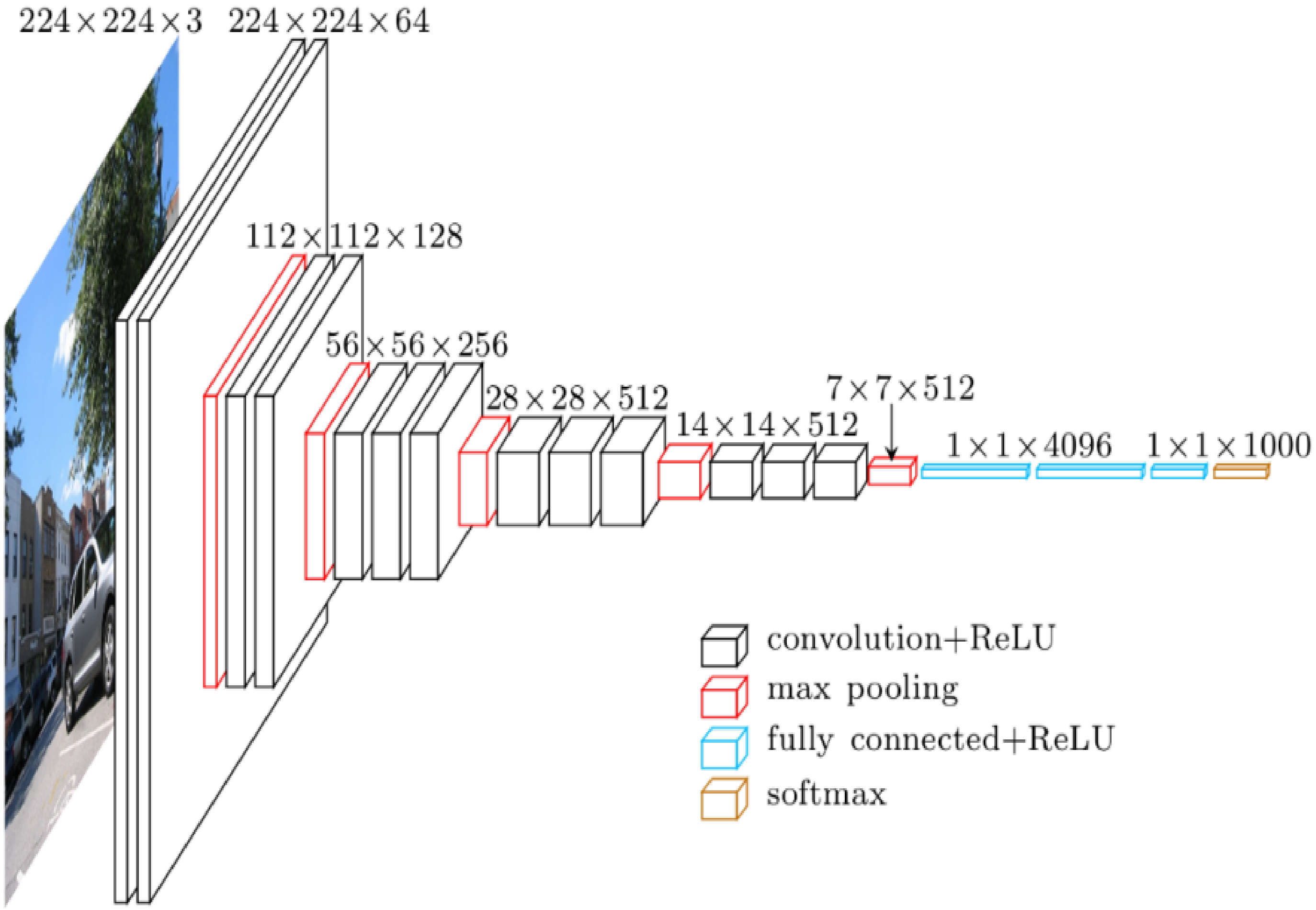
Convolutional Neural Network

- Convolution operation is very useful in processing 2D images because it is **translation invariant** and **computationally inexpensive** compared to a matrix operation
- Convolution is a linear transformation, replacing $\boldsymbol{W}x + \boldsymbol{b}$ in regular feedforward neural networks (often use "same" size option to specify convolution so output size = input size)
- Nonlinearity step such as **Rectified Linear Unit** still used
- **New building block: Pooling** such as max pooling:



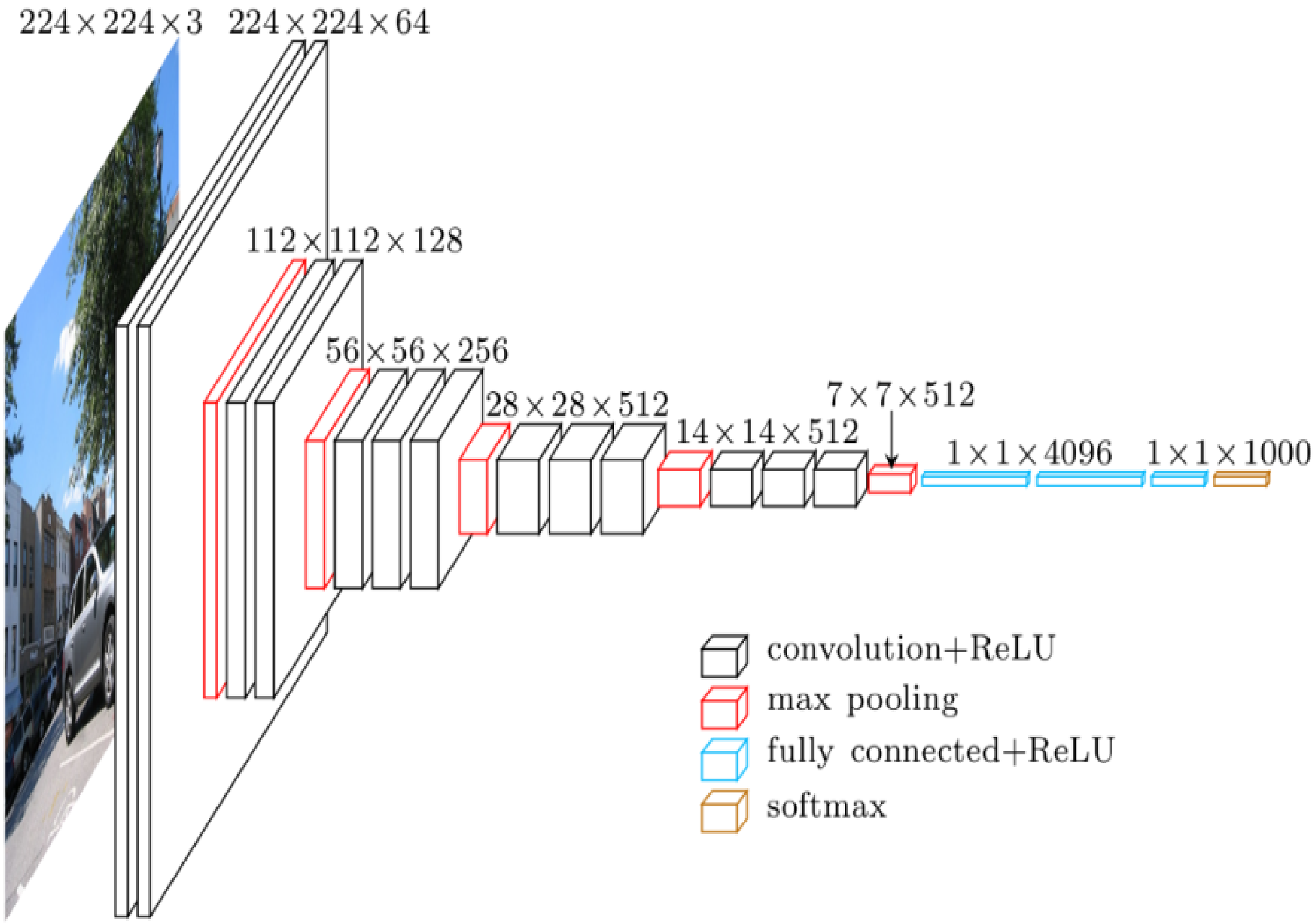
A Simple CNN Architecture

1. RBG image **H x W x 3** (224x224x3), we want to know what is in the input image
2. First **convolution** filter stack of **3 x K x K x F**: a stack of F filters each detect a feature (for example, detecting horizontal and vertical edges will requires a stack of 3 x K x K x 2).
3. Then **ReLU** is applied pointwise.
4. Second **convolution** filter stack is 64 x K x K x 64**,** third one is **64 x K x K x 128,** then **ReLU** again.
5. followed by **max pooling of 2x2** to reduce 224x224x128 to 112x112x128, repeat Steps (2 to)
6. Repeat from Step 2 to 5 with different matrices' size
7. After all Convo+ReLU layers, **stack the 7x7x512 to 1x1x4096, called "Flatten"**
8. In the end, a **softmax** operation can be used to do **classification** of objects, here identifying possibly 1000 objects



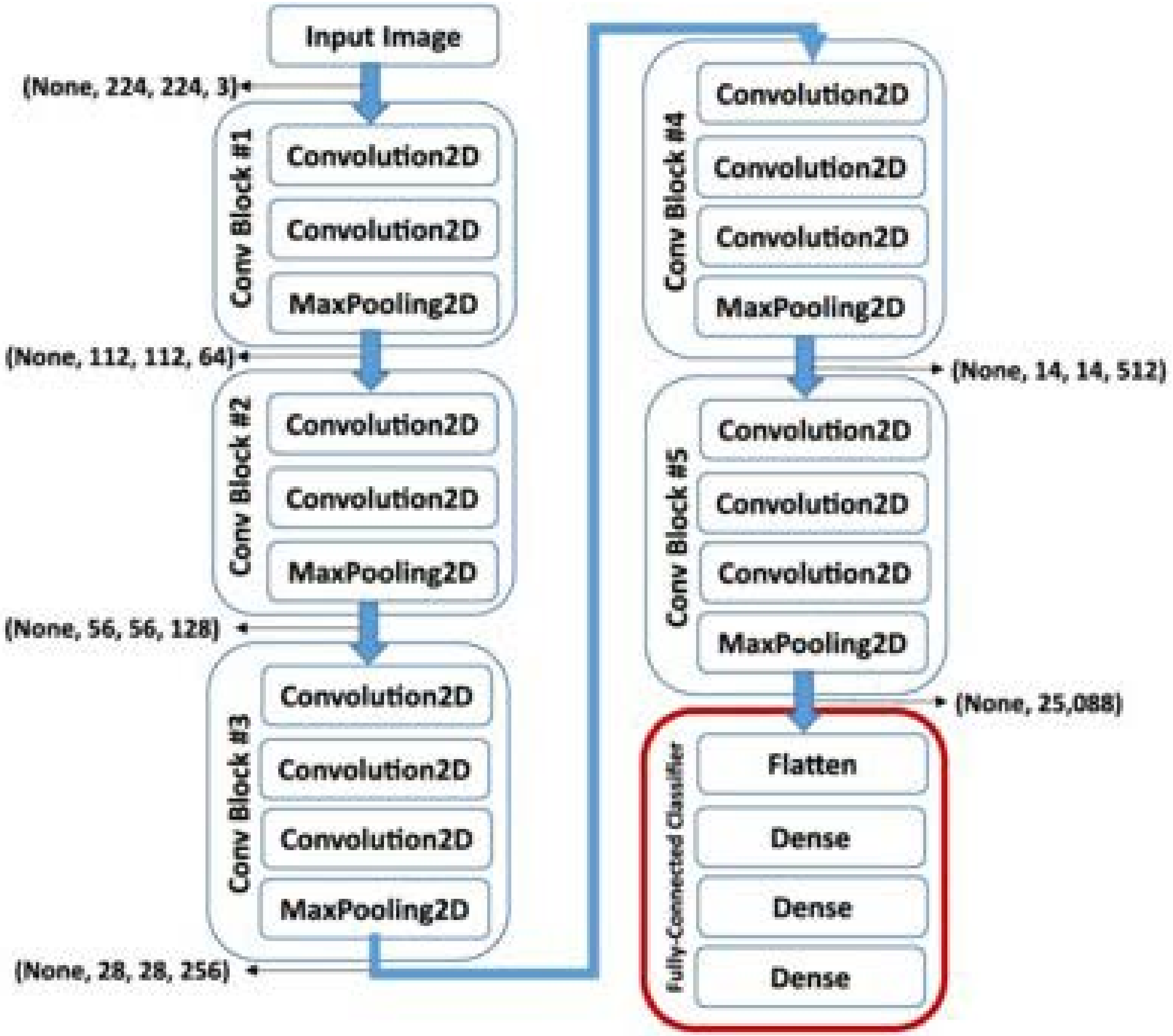
How to Interpret this?

- The first convolution 3x K x K x **F** extract **F number of features**, these features are simple features such as lines and edges.
- The output of the convolution operation indicates where the feature is found, max pooling retain that info while keeping the data size small (down sampling).
- Here the 1st convolution 3 x K x K x **64** extract **64 simple features** (shapes) from the RBG image
- The 3rd convolution 64 x K x K x **128** extracts more complicated features (128 of them) on top of the simple ones
- The deepest convolution stack (512 x K X K x **512**) **extracts very sophisticated feature representation** such as a car, a tree
- The **fully connected + ReLU layer** maps the stacked output of convolutional layers to an identification matrix: logits indicating if a car or a tree is present in the input



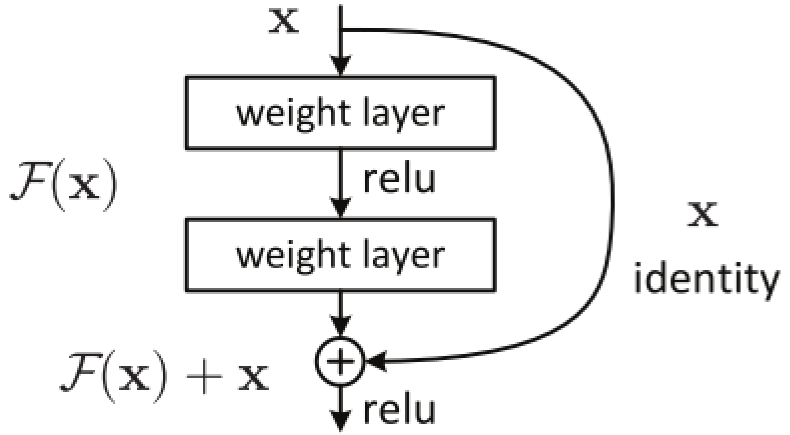
Some Widely Used CNN Architectures

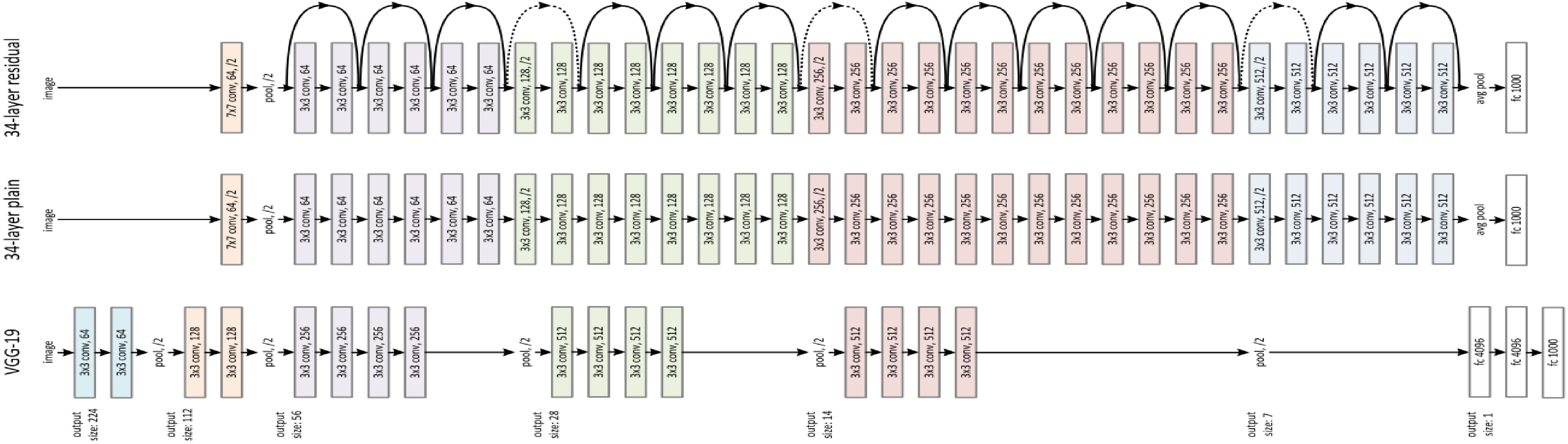
- The training of CNN uses chain rule as basis again
- Basically the deeper the better up to some extent
- All are combo of Conv and ReLU and Pooling with the last few layers as feedforward neural layers
- **VGG16 architecture on the right**
- Can find pre-trained model online



ResNet

- Deeper is not always better, for example identity operation $f(x) = x$ is notoriously difficult to replicate
- Introducing residual block





Practical Implementation/Software Tools

- Python with **sklearn**, **Theano**, **TensorFlow**(google) libraries. **Keras**, written using TensorFlow is a high-level neural network library (very simple to set up complicated neural networks with)
- C++ with **Caffe** library
- MATLAB with **MatConvNet** library
- All the above can achieve **GPU integration** to take advantage of its faster computational speed
- Pre-trained weights of an image classifier (for example a pretrained VGG16 model) can be downloaded from ImageNet

Onto Computer Vision

- We also want to identify the location of the object that we identified.
- Modify the final layers of fully connected neural networks (only used to determine if an object is there) and incorporate linear regression to get center coordinates and height and width
- One of the state-of-art algorithm is called **Single-Shot MultiBox Detector (SSD)** which adds different box predictors along with the orignal class predictors at different scales/default boxes to detect and locate objects