Survey of Convolutional Neural Network

Chenyou Fan
Indiana University
Bloomington, IN
fan6@indiana.edu

Abstract

Convolutional Neural Network (CNN) was firstly introduced in Computer Vision for image recognition by LeCun et al. *in 1989. Since then, it has been widely used in image recognition and classification tasks. The recent impressive success of Krizhevsky* et al. *in ILSVRC 2012 competition demonstrates the significant advance of modern deep CNN on image classification task. Inspired by his work, many recent research works have been concentrating on understanding CNN and extending its application to more conventional computer vision tasks. Their successes and lessons have promoted the development of both CNN and vision science. This article makes a survey of recent progress in CNN since 2012. We will introduce the general architecture of a modern CNN and make insights into several typical CNN incarnations which have been studied extensively. We will also review the efforts to understand CNNs and review important applications of CNNs in computer vision tasks.*

1. Introduction

In spite of all the fancy names like "data-driven method" or "deep learning method", as well as the disputes like whether convolutional neural network is panacea or poison of machine learning researchers, we have to know convolutional neural network is essentially a type of neural network. The concept of neural network has already existed since 1950s when Frank Rosenblatt created the perceptron. Even convolutional neural network itself is not a new concept at all. The convolutional neural network was firstly introduced in [33] to recognize handwritten ZIP code in 1989, and later extended to recognition and classification of various objects such as hand-written digits (MNIST) [34], house numbers [49], traffic signs [51], Caltech-101 [19] and more recently 1000-category ImageNet dataset [32].

CNNs saw extensive use in the 90s of 20th century, but fell out of fashion with the emergence of SVM and Bayesian models. One important reason is, small datasets in 1990s and early 2000s such has MNIST ($\sim$70000 instances) and Caltech-101 ($\sim$10000 instances), were incapable of training a modern convolutional neural network with deep layers of hundreds of millions of parameters. In comparison, SVMs and Bayesian models have relative fewer parameters which can be well optimized with smaller contemporary datasets. Therefore, in a long time, convolutional neural networks had not been found to surpass other machine learning methods. However, with the advent of much larger datasets in the first decade of 21st century, training very deep convolutional neural networks has become feasible. The breakthrough was finally made in 2012. Krizhevsky *et al*. [32] achieved substantially higher image classification accuracy with convolutional neural network on ImageNet dataset in the ILSVRC 2012 competition. Their deep convolutional neural network was trained on ImageNet dataset with 1.2 million labeled images with data augmentation, and structured with modified layers such as ReLU and Dropout. They also used more powerful hardware to perform float computations. These ideas and practices have laid the foundation for modern convolutional neural networks. Since then, more and more research works have shown the significant advance of CNNs over contemporary state-of-art methods on large datasets. Today, convolutional neural networks have been successfully applied for object detection [50, 56, 17, 21, 20, 47], image classification [32, 55], image segmentation [36], motion detection [48] and interdisciplinary tasks of computer vision and natural language processing [60, 30, 31]. In the meanwhile, open source deep learning tools such as *Caffe* [28] have been emerging and aggregating functions from the most recent research works. This provides another propulsion of the development of deep learning in computer vision.

This survey aims to provide a profile of architecture of a modern convolutional neural network, and to introduce some of most important applications in broad topics in computer vision and natural language processing. This survey is organized as follows. In Section 2, we will review the overall structure of a typical convolutional neural network. In Section 3, we will discuss the functions and evolutions

of various types of layers in convolutional neural networks. In Section 4, we will dive into technique details of training a convolutional neural network. In Section 5, we will introduce the state of the art convolutional neural network and discuss the keynotes of its success. In Section 6, we will discuss the deep features extracted from convolutional neural network layers as generic descriptors to represent images. In Section 7, we will show an example that image deep features are transformed and concatenated to represent video motions. In Section 8, we will review the efforts of visualizing and understanding convolutional neural networks. In Section 9, we will introduce the state of the art objection detection method which utilizes the power of deep features. In Section 10, we will discuss recent efforts of combining deep neural networks to model joint distribution of images and texts.

2. Neural Network Architectures

Though it has been over 25 years after the first convolutional neural network was proposed, modern convolutional neural networks still share very similar architectures with the original one, such as convolutional layers, and share similar training scheme such as forward and backward propagation. In this section, we will give a profile of a typical convolutional neural network and introduce the most important ideas in design of a convolutional neural network. In rest of this survey, we will use the term CNN and convolutional neural network interchangeably.

Firstly, we will introduce three most import convolutional neural network incarnations that frequently referred in this paper: LeNet, ConvNet, and VGGNet. The LeNet was firstly introduced by LeCun *et al*. [33] in 1989 and developed to mature in [34] in 1998 to recognize hand-written digits. LeNet is often seen as the prototype of a modern CNN. Its structure is shown in Figure 1. The ConvNet was proposed by Krizhevsky *et al*. [32] in 2012 for ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) competition, and started the era of deep learning in computer vision. Those creative ideas brought by ConvNet such as data augmentation and rectified linear activation function have been proven critical to training a great scale modern CNN. Its structure is shown in Figure 2. The VGGNet is an improvement of ConvNet in 2014, and its practice of using deeper layers and smaller convolutional filters has been widely accepted to improve performance. It is also a standard CNN of generating image deep features in many deep learning related works.

By observing these three types of CNNs, the common factor is that the first several layers are consisted of convolutional layers, activation layers and sub-sampling layers. These layers form a hierarchical way to extract local features from input and combine them to higher level features. The last few layers in common are those fully connected layers which compress image feature maps to 1-D vectors to facilitate the classification task. The details of the layer connections and dimensions are also shown in Table 1.

<table>
<tr><td colspan="3">Convolutional Neural Networks Configurations</td></tr>
<tr><td>LeNet</td><td>ConvNet</td><td>VGGNet</td></tr>
<tr><td>5 weight layers</td><td>8 weight layers</td><td>16 weight layers</td></tr>
<tr><td>Input (28×28 gray image)</td><td colspan="2">Input (224×224 RGB image)</td></tr>
<tr><td>conv 6@5x5</td><td>conv 96@11x11</td><td>conv 64@3x3
conv 64@3x3</td></tr>
<tr><td colspan="3">pooling</td></tr>
<tr><td>conv 16@5x5</td><td>conv 256@5x5</td><td>conv 128@3x3
conv 128@3x3</td></tr>
<tr><td colspan="3">pooling</td></tr>
<tr><td rowspan="7">NA</td><td rowspan="2">conv 384@3x3</td><td>conv 256@3x3
conv 256@3x3
conv 256@3x3</td></tr>
<tr><td colspan="2">pooling</td></tr>
<tr><td rowspan="2">conv 384@3x3</td><td>conv 512@3x3
conv 512@3x3
conv 512@3x3</td></tr>
<tr><td>NA</td><td>pooling</td></tr>
<tr><td rowspan="2">conv 256@3x3</td><td>conv 512@3x3
conv 512@3x3
conv 512@3x3</td></tr>
<tr><td colspan="2">pooling</td></tr>
<tr><td>fc-120</td><td colspan="2">fc-4096</td></tr>
<tr><td>fc-84</td><td colspan="2">fc-4096</td></tr>
<tr><td>fc-10</td><td colspan="2">fc-1000</td></tr>
<tr><td>RBF</td><td colspan="2">softmax</td></tr>
</table>

Table 1: LeNet, ConvNet and VGGNet Architectures

2.1. Feature Maps and Weight Sharing

In a regular neural network, hidden units, also known as neurons, are organized as 1-D vectors. However, due to the nature of images, hidden units in a CNN are generally organized as 2-D planes which we call *feature maps*. A convolutional layer may have 10s to 100s of feature maps due to the width of this layer. A feature map is obtained by applying convolution operations upon input image or previous features with a linear filter, adding a bias term and then applying a non-linear function. In another word, each unit in a feature map receives inputs from the combination of a $p \times p$ area of a subset or all of the features maps of previous layers. The combined areas are called the *receptive fields* of this unit. A simple 1D example is shown in Figure 3 to illustrate this connectivity. The interval of the receptive fields

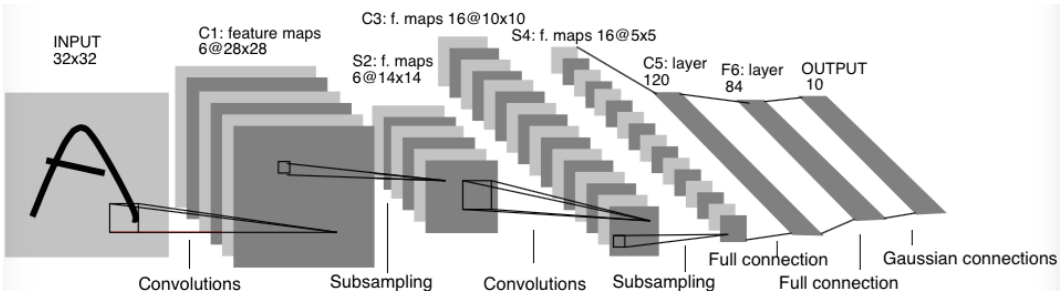


Figure 1: Architecture of LeNet taken from [34]. Two convolutional layers and two sub-sampling layers are interleaved to form the first four layers. Activation layers (not shown) are appended after every layer up to F6. Two fully connected layers are attached after the last sub-sampling layers to vectorize image representations. The last and output layer is composed of Euclidean Radial Basis Function (RBF) units which output the Euclidean distance between the network outputs and ground truth labels for 10 classes.

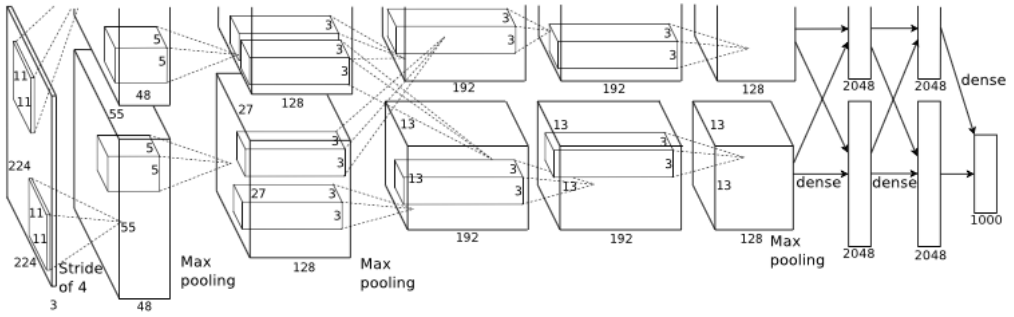


Figure 2: Architecture of ConvNet taken from [32]. Five convolutional layers and three sub-sampling (shown as max-pooling) layers are interleaved to form the first eight layers. Activation layers (not shown) are appended after convolutional layers. Two fully connected layers are attached after the last sub-sampling layers to vectorize image representations as in LeNet but with much higher dimensions. The last and output layer outputs the softmax loss of the network predictions for 1000 classes.

of neighboring units is called *stride*. The receptive fields of two neighboring units in a same feature map may overlap if stride is less than receptive field width. Figure 4 from [1] illustrates another important fact that units in a feature map share same weights since the same convolutional filter is used to scan over all possible receptive fields of previous feature maps.We will call these shared weights as convolutional filters or convolutional kernels interchangeably.

Shift invariance is achieved from weight sharing for the reason that if the input image is shifted the feature maps would shift to a same amount. Considering an image as input, elementary visual features such as edges and corners will be extracted at first, and then combined in upper layers to form high-level features. Since those elementary features could appear in any part of an image, shift invariance is important for being able to capture them.

Weight sharing has another practical benefit of greatly reducing the number of free parameters. In LeNet as shown in Figure 1, convolutional layer $C3$ has 16 feature maps, each unit in each feature map is connected to a 5×5 neighborhood in each or some of the feature maps in previous layer $S2$. So the upper bound of the number of free parameters is (kernel size + bias) $\times |S2| \times |C3| = (5 \times 5 + 1) \times 6 \times 16$ which is of asymptotic complexity $O(|S2||C3|)$, given the kernel size is a fixed small number. That means the number of free parameters to learn will not explode with the input size.

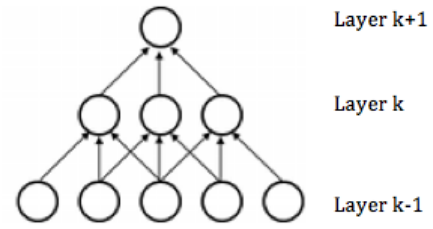


Figure 3: A 1D example of sparse connectivity between adjacent layers. Each unit in layer k receives input from 3 units in layer $k-1$, and thus has a receptive field of 3. Similarly, each unit in layer k+1 also has a receptive field of 3 with respective of layer k, but has a receptive field of 5 with respective of layer k-1. Layer k and k-1 are sparsely connected, while layer k+1 and k are fully connected. Also notice that units at layer k have overlapping receptive fields of stride 2.

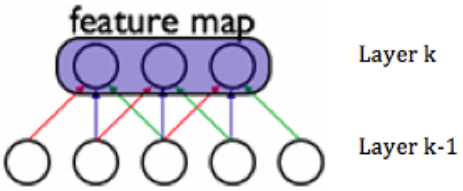


Figure 4: A 1D example of weight sharing of a feature map. The feature map in layer m contain three units. Each unit has a receptive field of 3 units in layer m-1, which get convoluted by a shared filter of size 3. The filter is shared across units in feature map, and thus the connections of same color in figure have the same weight value.

2.2. Sub-sampling

Sub-sampling, also known as pooling, is another important concept in CNN. The benefits of doing sub-sampling is to reduce feature dimensions, and thus computational costs by discarding non-maximum local values. The typical example is that the input of ConvNet has dimension 224×224 while its last fully connection layer only has 1×4096 after three pooling layers. Moreover, sub-sampling could provide robustness against noise and distortions in feature maps. The spatial resolution of feature maps is reduced and the coarse- resolution feature maps retain more distinctive patterns we are interested in. We will discuss sub-sampling layers in detail in Section 3.2.

2.3. Loss Functions

One of the most important questions of designing a neural network is how can we evaluate the performance of the whole system, or, how can be measure the errors. For a neural network of classification or regression task, we can define a loss function l given output of a neural network and ground truth of input data x. Given loss function l for a single training data, we can define loss function L over dataset $\boldsymbol{x}$ as

$$L(\boldsymbol{x}, W) = \frac{1}{N} \sum_{i=1}^{N} l(x_i, W) \tag{1}$$

The minimization of loss function with respective to model parameters turns out to be a optimization problem represented as

$$W^* = \arg\min_{W} L(\boldsymbol{x}, W) \tag{2}$$

In following two subsections, we will introduce two types of loss functions for different tasks.

2.3.1 Sum-of-Squares

The Sum-of-Squares loss function computes the sum of squares of differences between the predicted output of the neural network and the ground truth label of data under a supervised learning scheme.

$$L = \frac{1}{2N} \sum_{i=1}^{N} \|o_i - y_i\|^2 \tag{3}$$

Here we assume that the output is $\mathbf{o} = \{o_1, o_2, ..., o_N\}$, while the ground truth label is $\mathbf{y} = \{y_1, y_2, ..., y_N\}$. Sum-of-squared errors are usually used as measure of fitness between the model and data for a regression task.

2.3.2 Softmax

The Softmax loss layer computes the cross entropy of the softmax of outputs of neural network, and is usually used for classification task. The softmax loss function is

$$L = -\sum_{j} y_j \log p_j \tag{4}$$

where

$$p_j = \frac{e^{o_j}}{\sum_k e^{o_k}} \tag{5}$$

Here we assume output vector is $\mathbf{o} = \{o_1, o_2, ..., o_N\}$, softmax of outputs is $\mathbf{p} = \{p_1, p_2, ..., p_N\}$, and the ground truth label is $\mathbf{y} = \{y_1, y_2, ..., y_N\}$ which is a 1-of-N encoding, with constraints $\sum_k o_k = 1$ and $\sum_k y_k = 1$. One benefit of using softmax output is that the values of all output units

sum to 1, so the output can be interpreted as probability distribution over all classes under a classification problem domain. For example, ConvNet has an output layer of 1000 units corresponding to 1000 class labels of the ImageNet classification data set, and softmax of outputs are naturally interpreted as probability of each class. Another benefit of using softmax activation function and cross-entropy error function is that the derivative of loss L with respect to outputs $\mathbf{o}$ has a surprisingly succinct form. Firstly, we take derivative of p_j with respect to o_i,

$$\frac{\partial p_j}{\partial o_i} = \begin{cases} p_i(1-p_i) & i = j \\ -p_i p_j & i \neq j \end{cases} \tag{6}$$

The derivative of L with respect to o_i is,

$$\begin{aligned} \frac{\partial L}{\partial o_i} &= -\sum_j y_j \frac{\partial \log p_j}{\partial o_i} \\ &= -\sum_j y_j \frac{1}{p_j} \frac{\partial p_j}{\partial o_i} \\ &= -\left(y_i(1-p_i) + \sum_{k \neq i} -p_i y_k \right) \\ &= -y_i + \sum_k p_i y_k \\ &= p_i - y_i \end{aligned} \tag{7}$$

If we write this result in vector form of $\mathbf{o}$, we get

$$\frac{\partial L}{\partial \mathbf{o}} = \mathbf{p} - \mathbf{y} \tag{8}$$

where $\mathbf{p} = \{p_1, p_2, ..., p_N\}$ This nice property of softmax loss layer greatly facilitates error back-propagation during training of a neural network. It is also noteworthy that with softmax activation function and cross-entropy error function, the output layer of neural network is equivalent to a logistic regression model in which inputs are image features from last fully-connected layer and parameters are estimated by maximum likelihood [24].

2.4. Gradient-Based Parameter Optimization

The learning goal of CNN is to minimize the loss function with respect to the network parameters of each layer. This goal is at the heart of many other machine learning methods such as Bayesian and SVM models. Gradient learning method is based on the fact loss function Eq(1) could be minimized by estimating effects of small variations of parameter values on loss function. The quantities of the effects are measured by the gradient of the loss function with respect to the parameters. Let's consider at each training step a *mini-batch* of training data $x_{1...m}$ of size m as input. The mini-batch is used to approximate the gradient of the loss function with respect to parameters by

$$W^* = \arg\min_W \frac{\partial L(x_{1...m}, W)}{\partial W} \tag{9}$$

In following two subsections, we will discuss two widely used gradient-based methods. In Section 4, we will discuss how to take gradient of loss function with respective to parameters of each CNN layer.

2.4.1 Gradient Descent

Let's recall that the loss function $L(W)$ of Eq(1) is a function of model parameters W. The basic gradient-based updating method, which is called *gradient descent*, is to iteratively update W by negative gradient of loss function on a mini-batch:

$$W_{t+1} = W_t - \epsilon \frac{\partial L(W_t)}{\partial W}$$

Here, ϵ is a small fixed value which is called learning rate or learning step. Learning rate value is usually problem dependent and sometimes critical to performance.

2.4.2 Stochastic Gradient Descent

An improved gradient-based method, which is called *stochastic gradient descent*, updates the weights W by a linear combination of the negative gradient $\nabla L(W)$ and the previous weight update ΔW_t such that

$$\Delta W_{t+1} = \mu \Delta W_t - \epsilon \nabla L(W_t)$$

$$W_{t+1} = W_t + \Delta W_{t+1}$$

Here, learning rate ϵ is the weight of the negative gradient; momentum μ is the weight of previous update. Sometimes, an additional term called *weight decay* is added as a regularizer:

$$\Delta W_{t+1} = \mu \Delta W_t - \epsilon \nabla L(W_t) - \eta \cdot \epsilon \cdot W$$

Krizhevsky *et al*. [32] reported that weight decay term could reduce training error of a ConvNet. Stochastic gradient descent is more robust to noise in training data, since a sudden change in gradient is diluted and thus puts less affects on changes of parameters.

2.4.3 Discussion

The drawback of gradient descent method is that the learning rate ϵ is fixed, but a smaller learning rate is usually needed in later stage of training. To overcome this, manually setting learning rate ϵ at different training stages is a common practice, usually after a certain amount of

epochs or after validation error stops improving. For example, in training of ConvNet, Krizhevsky *et al*. initialized ϵ at 0.01, and divided it by 10 when validation error stops improving [32]. Also in some deep learning package like Caffe [28], dropping learning rate is an option of training protocol.

Gradient descent optimization methods only use first-order derivative information of target function, which is following the steepest descent direction at each step. Another iterative method commonly used to find optimal solution of a target function is Newton's method (or Newton-Raphson method). Newton's method is to find root of $f'(x) = 0$ by

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

In neural network scenario, loss function is $f(x)$, and optimal configuration of parameters is found by setting $f'(x) = 0$. If we want to utilize full Newton's methods, the inverse of Hessian $f''(x)$ has to be built. This is considered too costly in a complex model, not to say for a deep neural network. Gradient descent method is equivalent to Newton's method with inverse of Hessian matrix to be identical matrix. However, we generally want to preserve second-order information. Traditional optimization method which takes advantage of second-order information is to approximate inverse Hessian directly without explicitly calculating Hessian matrix. Such a class of optimization method is called Quasi-Newton method [63], including BFGS, LBFGS, etc. The advantage of using second-order information is that it converges super-linearly and it is less sensitive to initial values [6]. Its disadvantage is that it is much slower than using only first-order derivative at a single step, and it has to store more information and thus consumes more memory. There is little research of applying Quasi-Newton method to CNN, and thus the trade-off between convergence rate and per-step time remains unknown. What's more, it is a question whether second-order information collected on mini-batch is stable enough. Research of applying Quasi-Newton methods on logistic regression [6] shows its fast-convergence and better performance on middle-scale model. Future work on large-scale model such as CNN is expected to be shown in next few years, in the urgent need of speeding up training of a large-scale CNN not only in system perspective like using GPU, but in algorithm perspective with theoretic foundations.

2.5. Regularization

Regularization is widely used in statistical methods to control the complexity of the model in avoid of overfitting. One phenomenon of overfitting is that the training error decreases consistently during training stage, while error on validation set starts to increase at certain point. The more number of parameters of a model, the more risk the model is overfitting training data. In a deep CNN, overfitting is even more likely to happen with the exploding number of parameters of its layers. In ConvNet, for example, there are nearly 2,500,000 trainable parameters.

One method of reducing overfitting is to introduce a regularization term to the error measure in order to control the model complexity. The loss function with regularization term could have the form

$$L_{reg}(W) = L(W) + \eta r(W) \tag{10}$$

where $L(W)$ is the loss functions defined in Section 2.3, $r(W)$ is the regularization term of parameters W we want to regularize, and η controls the level of regularization. This regularized loss function is also seen in lasso and ridge regression, as well as many other statistical methods.

The gradient of this regularized loss function $L_{reg}(W)$ is nothing more than the gradient of original loss function $L(W)$ plus the gradient of regularization term $\eta r(W)$

$$\nabla L_{reg}(W) = \nabla L(W) + \eta \nabla r(W) \tag{11}$$

A commonly used regularization term is in form of

$$r(W) = \|W\|_F^2 = \sum_{i,j} |W_{ij}|^2 = Tr(WW^T) \tag{12}$$

Its gradient is easily taken as [43]

$$\frac{\partial\, r(W)}{\partial\, W} = \frac{\partial\, Tr(WW^T)}{\partial\, W} = 2W \tag{13}$$

Bishop [4, Chapter 5.5.5] has a discussion which shows equivalence between using data augmentation technique and adding regularization term to error function.

2.6. Dropout

Dropout is another regularization technique which randomly skips certain hidden neurons at each training iteration with probability of p, say 0.5. Those skipped neurons which are so-called "dropped out" will not contribute to either forward pass or backward pass of this iteration. Dropout is usually applied to fully-connected layers of CNNs at training stage. At testing stage, dropout is turned off, and each neuron output is given reduced weight by a factor of $(1 - p)$. This technique ensures the sparsity of learned features and thus reduces model complexity. In another point of view, dropout forces to break potential co-occurrences of certain features, and therefore makes the network to learn more distinctive and robust features [32]. Dropout technique is not limited to CNNs. It is a common technique could be applied to various kinds of neuron networks. For example, in recurrent neural networks (RNNs), dropout could be applied on non-recurrent connections to achieve better results [64].

2.7. Data Augmentation

Data augmentation is such a technique that can reduce overfitting by artificially increasing the size of training set at very low cost. It is usually done by making label-preserving transformations which produce multiple copies of each original data instance without changing the label. Cropping and flipping are two widely used forms of data augmentation in most CNN implementations [32, 7]. Flipping is to produce horizontal reflections of training images. Cropping is to randomly extract multiple slightly smaller frames from original images. For example, Krizhevsky *et al*. firstly down-sample and crop all ImageNet images to a fixed resolution of 256×256, and extract random 224×224 patches and their horizontal reflections to enlarge the original training set by a factor of 2048 [32]. At test time, four corner patches and the center patch (along with their reflections) of size 224×224 are extracted from each testing image and predictions are made by averaging the predictions of all ten patches. Szegedy *et al*. [55] propose cropping a same image in different scales instead of a fixed scale being down-sampled to. Specifically, they resize images to four scales where each image has a shorter dimension of256, 288, 320 and 352 respectively. Then they take left, right and center (or top, bottom and center) square patches of these resize images. Finally they take the four corners and center crop of size 224×224, together with the squares, and their mirrors as input images. Other data augmentation methods include contrast and color jittering, and adding Gaussian noise.

3. CNN Layers

As fundamental building blocks of CNNs, CNN layers have shown their variety and flexibility both in their designed structures and the connections. The efforts in modifying layer structures and connections have led to the possibility of training a CNN faster and of making it perform better. In following subsections, we will introduce common layers of modern CNNs and their functions.

3.1. Activation Layer

Activation layer is usually in the form of an activation function on top of a layer's outputs. The main purpose of activation functions is to introduce non-linearity into neural networks. Without activation functions, the whole neural network would be a linear transformation from input to output. The output of activation layer l is the application of activation function upon feature maps of layer l-1. Mathematically,

$$\boldsymbol{x}^l = f\left(\boldsymbol{x}^{l\text{-}1}\right) \tag{14}$$

There are several types of activation functions which are widely used: ReLU (Rectified-Linear Unit), Sigmoid and TanH (Hyperbolic Tangent), etc.

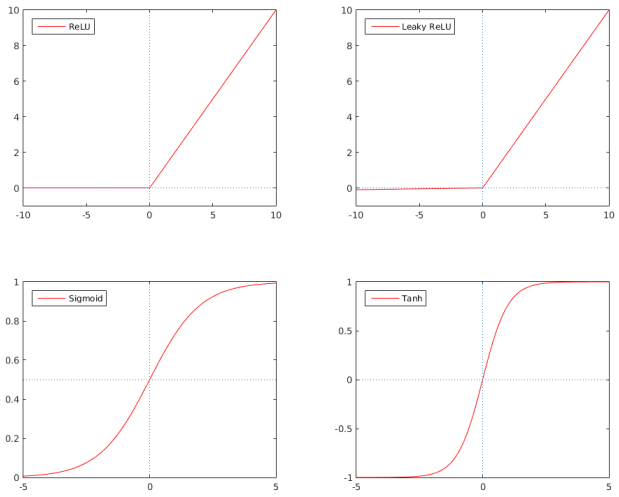


Figure 5: ReLU, Leaky ReLU, Sigmoid and Tanh functions

3.1.1 Sigmoid

The definition of sigmoid function is

$$\sigma(z) = \frac{1}{1 + \exp(-z)} \tag{15}$$

Sigmoid function will squash the input onto interval $[0, 1]$, as shown in Figure 5. The derivative of a sigmoid function is

$$\sigma(z)' = \sigma(z)(1 - \sigma(z)) \tag{16}$$

3.1.2 Tanh

Tanh function, also known as hyperbolic tangent function, can be viewed as a linear transformation of sigmoid onto interval $[-1, 1]$, as shown in Figure 5. The definition of tanh function is

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{17}$$

The derivative of a tanh function is

$$\tanh(z)' = 1 - \tanh(z)^2 \tag{18}$$

3.1.3 ReLU

In a conventional neural network before 2012, sigmoid or tanh were the default activation functions. The problem of sigmoid and tanh is their outputs can easily get saturated to 0 or 1 for sigmoid and $+1$ or -1 for tanh, as shown in Figure 5. The saturated outputs will make gradients vanish, which can be easily seen by putting saturated values to

Eq(16) and Eq(18). This greatly slows down training a neural network, and makes it more possible to trap into local minimum.

ReLU, short for Rectified-Linear Unit, was introduced in 2010 by Nair and Hinton [40] to speedup the converge of training neural networks. ReLU has the benefit that it does not get saturated, also it's efficient to compute a rectified linear function. ReLU gives the possibility to train much larger neural networks like CNN. The definition of a rectified function is

$$f(\mathbf{x}) = \max(0, \mathbf{x}) \tag{19}$$

In CNN, layer outputs are usually feature maps with two dimensions. In this case, $\mathbf{x}$ is a matrix and max is elementwise applied on each matrix element. A variant of ReLU, which is called Leaky ReLU, allows a small, non-zero value even if the unit is not active. The definition is

$$f(x) = \begin{cases} x & x > 0 \\ 0.01x & \text{otherwise} \end{cases} \tag{20}$$

ReLU and Leaky ReLU functions are shown in Figure 5.

3.2. Pooling Layer

Pooling layer, also referred as sub-sampling layer or down-sampling layer, is to sub-sample the feature maps obtained by previous layers. Pooling layers bring benefits to CNNs in two ways: it reduces computations for upper layers by reducing dimensions of feature maps; it provides robustness to noise and small distortions [27] and achieves translation invariance [34] by reducing spatial resolution, and thus increases the richness of the representation of the features.

There are two types of pooling layer widely used in CNNs. **Average pooling** is to compute the average value within each pooling window across feature maps of layer l-1 as one unit of corresponding feature map of layer l. **Max pooling** is to compute the maximum value within each pooling window across each of the feature map of layer l-1 as one unit of corresponding feature map of layer l. A good illustration of max pooling operation from Zeiler [65] is shown in Figure 6. The right part of the figure shows pooling operation during forward propagation. In a pooling window, i.e. 2×2 in this case, maximum value is retained, and its position in pooling window is also recorded in a "Switch Map", or we call it position mask. The left part of the figure shows unpooling operation during backward propagation, which we will discuss in Section 4.1. In another point of view, pooling operation is a special case of convolution. Average pooling operation is equivalent to applying a average filter, while max pooling is similar to applying a varying filter which has value 1 at maximum value while 0 everywhere else.

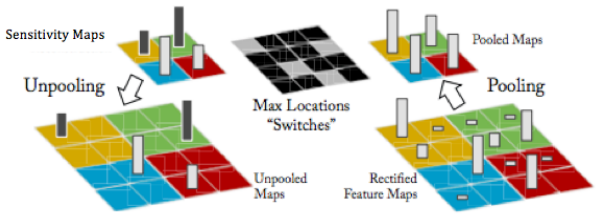


Figure 6: An illustration of max pooling operation from Zeiler [65]. The right part shows pooling operation during forward propagation. Local maximum values in each 2×2 pooling windows are placed to pooled feature maps and their positions are recorded in mask. The left part shows unpooling operation during backward propagation. Sensitive maps are up-sampled and values placed back to positions recorded in mask.

Overlapping and non-overlapping pooling windows are both used in different architectures of CNNs. A pooling window of typical size 2×2 in layer l-1, with stride specified as 2, will result in a non-overlapping subs-ampling of the feature map and reduce its width and height to half each. However, if the stride is 1 instead, each pooling window is overlapping: all pixels except those at borders are subsampled twice, and the size of the feature map is almost unchanged after sub-sampling.

3.3. Fully Connected Layer

In a regular neural network, each neuron in a fully connected layer receive inputs from all neurons in previous layer. Their activations can be computed by a matrix multiplication followed by a bias offset addition during forward propagation. In a typical CNN, the feature maps of last convolutional layer are vectorized and fully connected with output units, which are followed by a softmax loss layer. As discussed in Section 2.3, this structure turns convolutional structure to traditional neural network classifiers. In ConvNet, two fully connected layers of size 4096 and the final fully connected layer with 1000 output units that compute the class scores are attached to the last convolutional layer, as shown in Figure 2.

It is also interesting to find that a fully connected layer is a special case of a densely connected convolutional layer with filter size of the same as the feature maps. We can illustrate this with ConvNet example. The last convolutional layer of ConvNet yields 256 feature maps of size 7×7. Instead of vectorizing them into a $7 \times 7 \times 256$ vector and have full connections with 4096 neurons as in [32], we replace fully connected layer with a convolutional layer of 4096×256 filters of size 7×7 which yields 4096 feature maps of size 1×1. We also replace second fully connected

layer with a convolutional layer of 4096×4096 filters of size 1×1. Finally, we replace the last fully connected layer with a convolutional layer of 4096×1000 filters of size 1×1. This conversion yields a "full convolutional network" equivalent with ConvNet but without any fully connected layer. This allows us to "slide" the original CNN over many spatial positions in a larger image, in a single forward pass. The following example is given by Karpathy [2]. ConvNet has a 224×224 image as input and gives feature maps $7 \times 7 \times 512$ at its fifth convolutional layer - i.e. a size reduction by 32. By converting ConvNet's fully connected layers to convolutional layers, we can forward an image of size 384×384 through the converted architecture and get an equivalent volume of size $12 \times 12 \times 512$, with same size reduction by 32. Following through with the next 3 convolutional layers converted from fully connected layers would give the final volume of size $1 \times 1 \times 1000$ for 224×224 input and volume of size $6 \times 6 \times 1000$ for 384×384 input image, with calculation $(12 - 7)/1 + 1 = 6$. By feeding a larger image into converted ConvNet, we are now getting 6×6 array of 1000 class scores across the 384×384 image. In another word, forwarding converted ConvNet with a 384×384 image just one time is equivalent to evaluating the original ConvNet with fully connected layers independently across 224×224 crops of the 384×384 image in strides of 32 pixels, but much more efficiently.

3.4. Convolutional Layer

At a convolutional layer l, previous layers' feature maps are convolved with learnable kernels to form the output feature maps at this layer. Let the i-th feature map at layer l denoted as $\boldsymbol{x}_i^l$, and j-th feature map at layer l-1 denoted as $\boldsymbol{x}_j^{l\text{-}1}$. $\boldsymbol{x}_i^l$ is computed as

$$\boldsymbol{x}_i^l = \sum_{j \in M_i} \boldsymbol{k}_{ij}^l * \boldsymbol{x}_j^{l\text{-}1} + b_i^l \tag{21}$$

for $i = 1, 2, ..., |M|$. Here, b_i^l is a bias term shared across all connections to i-th feature map, $|M|$ is number feature maps at layer l. M_i represents a subset of feature maps at layer l-1 that connected to units i at layer l. In LeNet, a group of manually selected subsets were specified, while in many other CNNs layer l and l-1 are fully connected. As we can see, there will be at most $|M| \times |N|$ learnable kernels at convolutional layer l, assuming $|N|$ feature maps at previous layer and feature maps in l and l-1 are fully connected. Figure 7 illustrates an example of fully connections between convolutional layer l and l-1.

The size of kernel $\boldsymbol{k}$, which decides the size of the receptive field, is usually set to a small value. For example, LeNet sets the size to 5×5 for all three of its convolutional layers, while ConvNet and VGGNet have even smaller kernels. The benefit of making the kernel size small is that kernels can scan over overlapping pixels and the feature maps can retain more local details [65].

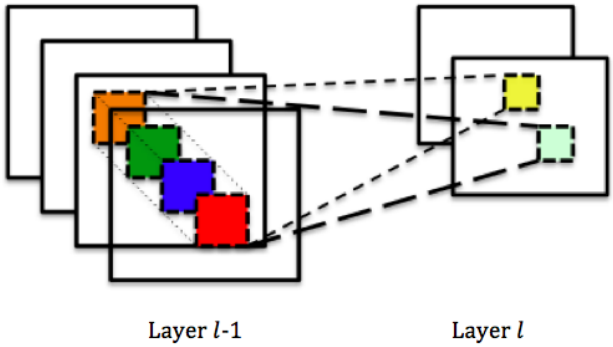


Figure 7: Convolutional layer l is fully connected with layer l-1. Units in feature maps in layer l are computed from their receptive fields of units in layer l-1. Since these two layers are fully connected, receptive field of unit in convolutional layer l spans all feature maps in layer l-1.

4. Backpropagation Through Layers

In Section 2.4, we have discussed that the key step of updating model parameters by using gradient-based methods is to take gradient of loss function with respective to parameters. Back-propagation is such an efficient way to calculate gradients of parameters in each layer of a neural network.

The basic idea of back-propagation is that gradients can be efficiently computed by tracing from output layer back to input layer in sequence. In another word, derivatives of loss function L with respect to the network parameters can be computed from higher layers to lower layers by using chain rule. This is done by storing the error message arriving at one layer l as $\boldsymbol{\delta}^l$. $\boldsymbol{\delta}^l$ could be viewed as the "sensitivities" of parameters at this layer, i.e., how much perturbations would it undertake when the loss function changes one unit.

Let superscript l denote the current layer, L denote the output layer. For clarity, we use E to denote the loss function. Let's consider a general neural network. The output of each layer can be defined as

$$\boldsymbol{x}^l = f(\boldsymbol{u}^l), \quad \boldsymbol{u}^l = \boldsymbol{W}^l \boldsymbol{x}^{l\text{-}1} + \boldsymbol{b}^l \tag{22}$$

where f is activation function. Define sensitivities at layer l as $\boldsymbol{\delta}^l$

$$\boldsymbol{\delta}^l = \frac{\partial E}{\partial \boldsymbol{u}^l} \tag{23}$$

Use chain rule and (22) to get $\boldsymbol{\delta}^{l\text{-}1}$

$$\begin{aligned}\boldsymbol{\delta}^{l\text{-}1} &= \frac{\partial E}{\partial \boldsymbol{u}^{l\text{-}1}} \\ &= \frac{\partial E}{\partial \boldsymbol{u}^{l}}\frac{\partial \boldsymbol{u}^{l}}{\partial \boldsymbol{u}^{l\text{-}1}} \\ &= \frac{\partial E}{\partial \boldsymbol{u}^{l}}\frac{\partial \boldsymbol{u}^{l}}{\partial \boldsymbol{x}^{l\text{-}1}}\frac{\partial \boldsymbol{x}^{l\text{-}1}}{\partial \boldsymbol{u}^{l\text{-}1}} \\ &= (\boldsymbol{W}^{l})^{T}\boldsymbol{\delta}^{l} f'(u^{l\text{-}1})\end{aligned} \tag{24}$$

By reusing $\boldsymbol{\delta}^{l}$ we could compute $\boldsymbol{\delta}^{l\text{-}1}$, and thus compute all $\boldsymbol{\delta}^{l}$ from L-1 to 1 in a cascade way. For the output layer with Softmax loss function as described in Section 2.3.2, $\boldsymbol{\delta}^{L}$ is given by

$$\begin{aligned}\boldsymbol{\delta}^{L} &= \frac{\partial E}{\partial \boldsymbol{u}^{L}} \\ &= \frac{\partial E}{\partial \boldsymbol{y}}\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{u}^{L}} \\ &= (\boldsymbol{y}-\boldsymbol{t}) f'(\boldsymbol{u}^{L})\end{aligned} \tag{25}$$

With help of cached $\boldsymbol{\delta}^{l}$, we can define the derivative with respect to actual weights in an elegant manner.

$$\frac{\partial E}{\partial \boldsymbol{W}^{l}} = \frac{\partial E}{\partial \boldsymbol{u}^{l}}\frac{\partial \boldsymbol{u}^{l}}{\partial \boldsymbol{W}^{l}} = \boldsymbol{x}^{l\text{-}1}(\boldsymbol{\delta}^{l})^{T} \tag{26}$$

In a regular neural network, Eq(22) could well define the layer connections, so that Eq(22, 25, 26) give the complete rules to update parameters. However, in spite of fully connected layers, CNNs have more distinctive layers such as convolutional layer, pooling layer, and also have special activation functions such as ReLU. Therefore, how to back-propagate derivatives through such layers and how to apply Eq(25, 26) to calculate the gradients of weight parameters deserve more details. We will discuss them in following subsections.

4.1. Gradient of Pooling Layer

In Section 3.2, we have discussed two different types of pooling layers: average pooling and max pooling. Pooling layers generally don't have learnable parameters themselves, so the only important procedure during back-propagation is to compute the error messages passed through pooling layers. By following convention of [5], we introduce two operators *down* and *up*: *down* represents down-sampling operation of pooling layers during forward propagation, and *up* represents up-sampling operation upon error signals from above layers during backward propagation. Mathematically,

$$\begin{cases}\boldsymbol{x}^{l} = down\left(\boldsymbol{x}^{l\text{-}1}\right) & \text{pooling} \\ \boldsymbol{\delta}^{l\text{-}1} = up\left(\boldsymbol{\delta}^{l}\right) & \text{unpooling}\end{cases} \tag{27}$$

where $down(\cdot)$ down-samples a small neighborhood of pooling window size to the max value and average value for max pooling and average pooling respectively in forward propagation; $up(\cdot)$ up-samples one unit of sensitivity map $\boldsymbol{\delta}$ to its origin neighborhood by placing the value at the position where the max value was for max pooling, and setting all sampled units to this value for average pooling in backward propagation. An efficient implementation of max pooling is to record the positions of all local maxima of each neighborhood as a binary mask matrix (or switches) during forward propagation. During backward propagation, as shown in left part of Figure 6, we firstly up-sample the sensitivity map from above layer and then apply mask matrix upon it to recover sensitivities at locations of local maxima. The up-sampling operation can be mathematically defined by Kronecker product [5]

$$up(\boldsymbol{x}) = \boldsymbol{x} \otimes \mathbf{1}_{w\times w}$$

where w is window size. A non-overlapping pooling layer will reduce the feature map by a factor of window size at each dimension. Sometimes we may desire keeping the feature map size unchanged. In this case, we can either use overlapping pooling window or pad units to feature maps before pooling [53].

4.2. Gradient of Convolutional Layers

In convolutional layers, feature maps from the lower layers are convolved with learnable kernels, as we discussed in Section 3.4. Now we discuss how to back-propagate errors through convolutional layers and how to update kernels and biases. The content of this subsection is mostly discussed in [5].

As shown in Eq(21), kernel $\boldsymbol{k}_{ij}^{l}$ is shared across all patches in the j-th feature map $\boldsymbol{x}_{j}^{l\text{-}1}$ at layer l-1. Let $(\boldsymbol{x}_{j}^{l\text{-}1})_{uv}$ be one patch in $\boldsymbol{x}_{j}^{l\text{-}1}$ of kernel size. By nature of convolution, $(\boldsymbol{p}_{j}^{l\text{-}1})_{uv}$ is multiplied by $\boldsymbol{k}_{ij}^{l}$ element by element and summed to compute (u,v)-th element of i-th feature map at layer l. To illustrate this, we rewrite each element of Eq(21) individually as

$$\begin{aligned}\boldsymbol{x}_{i}^{l} &= \sum_{j\in M_i} \boldsymbol{k}_{ij}^{l} * \boldsymbol{x}_{j}^{l\text{-}1} + b_{i}^{l} \\ (\boldsymbol{x}_{i}^{l})_{uv} &= \sum_{j\in M_i} \left(\boldsymbol{k}_{ij}^{l} * (\boldsymbol{x}_{j}^{l\text{-}1})_{uv}\right) + b_{i}^{l}\end{aligned} \tag{28}$$

Let $(\boldsymbol{\delta}_{i}^{l})_{uv}$ be (u,v)-th element of the i-th sensitivity map at layer l during back-propagation. Let's introduce an operator $flip(\cdot)$ first. $flip(\cdot)$ flips input horizontally and vertically, which is corresponding to the first step of discrete convolution. For one patch, the gradient for $\boldsymbol{k}_{ij}^{l}$ is $flip\left((\boldsymbol{\delta}_{i}^{l})_{uv}(\boldsymbol{x}_{j}^{l\text{-}1})_{uv}\right)$. Since $\boldsymbol{k}_{ij}^{l}$ is shared across all patches in this feature map, we sum the gradients from all

patches as total gradient for $\boldsymbol{k}_{ij}^l$

$$\begin{aligned}\frac{\partial E}{\partial \boldsymbol{k}_{ij}^l} &= flip\left(\sum_{u,v}(\boldsymbol{\delta}_i^l)_{uv}(\boldsymbol{x}_j^{l\text{-}1})_{uv}\right)\\ &= flip\left(corr(\boldsymbol{\delta}_i^l, \boldsymbol{x}_j^{l\text{-}1})\right)\\ &= flip\left(flip(\boldsymbol{\delta}_i^l) * \boldsymbol{x}_j^{l\text{-}1}\right)\\ &= \boldsymbol{\delta}_i^l * flip(\boldsymbol{x}_j^{l\text{-}1})\end{aligned} \tag{29}$$

where $corr(\cdot,\cdot)$ is cross-correlation operator, and it is equivalent to flipping one of the element and convoluting each other. Eq(29) shows that the derivative of kernel is another convolution. The bias b_i^l is added to every unit of feature map $\boldsymbol{x}_i^l$, thus the partial derivative of E with respective to b_i^l is given by

$$\frac{\partial E}{\partial b_i^l} = \sum_{u,v}(\boldsymbol{\delta}_i^l)_{uv} \tag{30}$$

To pass sensitivity map $\boldsymbol{\delta}_i^l$ through convolutional layer, we use Eq(28) again and have

$$\begin{aligned}\boldsymbol{\delta}_j^{l\text{-}1} &= \frac{\partial E}{\partial \boldsymbol{x}_j^{l\text{-}1}} = \sum_{j\in M_i}\frac{\partial E}{\partial \boldsymbol{x}_i^l}\frac{\partial \boldsymbol{x}_i^l}{\partial \boldsymbol{x}_j^{l\text{-}1}}\\ (\boldsymbol{\delta}_j^{l\text{-}1})_{uv} &= \sum_{j\in M_i}\left(\frac{\partial E}{\partial \boldsymbol{x}_i^l}\right)_{uv}\frac{(\partial \boldsymbol{x}_i^l)_{uv}}{(\partial \boldsymbol{x}_j^{l\text{-}1})_{uv}}\\ &= \sum_{j\in M_i}(\boldsymbol{\delta}_i^l)_{uv}\cdot flip(\boldsymbol{k}_{ij}^l)\end{aligned} \tag{31}$$

or simply

$$\boldsymbol{\delta}_j^{l\text{-}1} = \sum_{j\in M_i}\boldsymbol{\delta}_i^l * \boldsymbol{k}_{ij}^l \tag{32}$$

4.3. Gradient of Activation Layers

At activation layer l, feature maps at layer l-1 is applied with activation function and passed on to higher layer l+1. Since activation layers generally have no learnable parameters, the only important procedure during back-propagation is to compute the error messages $\boldsymbol{\delta}^{l\text{-}1}$ from $\boldsymbol{\delta}^l$ and pass to its lower layer. By using Eq(14), we can derive the error messages $\boldsymbol{\delta}^{l\text{-}1}$ in the following way

$$\begin{aligned}\boldsymbol{\delta}^{l\text{-}1} &= \frac{\partial E}{\partial \boldsymbol{x}^l}\frac{\partial \boldsymbol{x}^l}{\partial \boldsymbol{x}^{l\text{-}1}}\\ &= \boldsymbol{\delta}^l f'(x^{l\text{-}1})\end{aligned} \tag{33}$$

When activations function is sigmoid or tanh, substitute $f'(\cdot)$ with Eq(16) or Eq(18) will complete Eq(33). When activation function is ReLU, as shown in Eq(19), in addition to storing the rectified result $\boldsymbol{x}^l$, we should better cache a mask matrix $\mathbf{M}$ to store positions of elements of $\boldsymbol{x}^{l\text{-}1}$ which are positive and not rectified to 0 in forward propagation. Mask matrix $\mathbf{M}$ is used for efficient back-propagation since only at non-zero positions error messages are allowed to pass, otherwise error messages are zeroed out. Mathematically, Eq(33) is reduced to

$$\boldsymbol{\delta}^{l\text{-}1} = \boldsymbol{\delta}^l \cdot \boldsymbol{M} \tag{34}$$

4.4. Discussion

One problem of gradient-based methods is that the solution could be local minima instead of global minima. But this seems to be not a problem of neural networks. LeCun *et al*. conjectured that an oversized network compared to the difficulty level of the task is not likely to trap into local minima due to the extra dimensions of parameter space [34].

5. State of the Art CNN

Until ILSVRC 2014 (Imagenet Large Scale Visual Recognition Challenge), the state of the art CNN architecture in classification task is called GoogLeNet, proposed by Szegedy *et al*. [55]. Their success lies in introducing a special dimension reduction module which explores the sparsity inside convolutional layer, and thus allows increase in both depth and width of GoogLeNet without exploding computations. Later, Ioffe and Szegedy [26] proposed adding a Batch Normalization step before every activation layer of CNN, and batch-nomalized GoogLeNet has further boost its performance. We will discuss GoogLeNet architecture in 5.1 and batch normalization technique in 5.2

5.1. GoogLeNet

The importance of the size of a CNN architecture to its performance has been demonstrated in [53]. The size of a CNN has two dimensions, the depth – number of layers, and the width – number of units at each layer. Two main reasons prevent unlimitedly increasing size of a CNN. Firstly, a network with a large number of parameters is prone to overfitting when labeled training data is limited. Secondly, a practical training time and usage of memory is always the concern.

The above question can be abstracted to "how to increase the size of a CNN while keep the parameters in limited number". In a densely connected CNNs, any linearly increased layer size will result in quadratically increased number of connections between adjacent layers. However, recent works [13, 35] have shown there is heavy redundancy of CNN parameter. Szegedy *et al*. proposed to use sparsely connected architectures instead of current fully connected architectures of CNN inside convolutional layers. Let's take a brief review of Section 3.4. Parameter M_i of Eq(21) indicates the subset of feature maps at layer l-1 connected to units i at layer l. In Krizhevsky's ConvNet [32], this

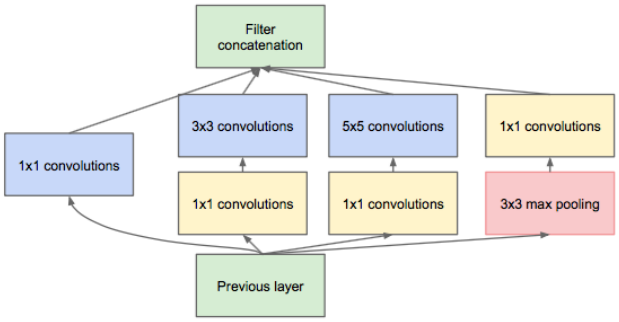


Figure 8: Inception layer of GoogLeNet. Two 1×1 convolutions stacked before expensive 3×3 and 5×5 convolutions are used to reduce dimensions, and also combined with ReLU activation.

inter-layer connection is dense and M_i includes all units of layer l-1. However, the sparse connection concept can be traced back to LeNet [33]. A manually set mapping table determines subsets of connections between the third convolutional layer feature maps with previous ones, in order to break symmetry. This inspires Szegedy *et al*. to design a new layer type called *Inception* layer which can find optimal local sparse structure automatically by using existing CNN components. Figure 8 shows the structure of an inception layer. 1×1 convolutions are used for two purposes: to reduce number of feature maps by linearly combing them before costly 3×3 and 5×5 convolutions, and to apply ReLU activation function upon reduced feature maps. Those 1×1 convolutional kernels are of actual size $M \times 1 \times 1 \times N$, if we denote M as input feature maps and N as output feature maps number. When $M > N$, dimension reduction is achieved. In GoogLeNet, nine such inception layers are stacked together after the first two regular convolutional layers, with five pooling layers inserted to halve the resolution of feature maps. In ILSVRC 2014 classification task, Szegedy *et al*. trained seven versions of GoogLeNet independently and averaged their softmax probabilities over all individual classifiers to obtain final prediction. The ensemble of GoogLeNet achieved the 1st place in top-5 error which was 6.67% and slightly better than the 7.32% of 2nd place VGGNet.

5.2. Batch Normalization

As discussed in Section 2.4, the subset of training data on which CNN is trained at each forward-backward propagation is called mini-batch. The general practice of generating mini-batches is to shuffle training data first and divide into batches of equal size. Each mini-batch has different distribution with the whole dataset, which could be distinctive for small batch size. The consequence is the distribution of layers' input are constantly changing, thus layers have to adapt to new distribution at each iteration. The change in input distribution is called *covariate shift* [52], and the subsequent change in neural network layers' activations is called *internal covariate shift*. Internal covariate shift slows down the training of CNNs. Ioffe and Szegedy [26] propose to use a technique called *Batch Normalization* to reduce internal covariate shift and accelerate the training of deep neural nets.

Normalization of inputs is not a new concept. In LeNet, all input images are linearly transformed to have zero mean and unit variance in order to achieve faster training speed [34]. Ioffe and Szegedy call this operation *whitening*. The basic idea of batch normalization is to perform whitening on inputs of all layers in sequence. Let's consider a feature vector $x = (x^{(1)} \dots x^{(d)}) \in \mathcal{X}$ of d-dimension, where $\mathcal{X}$ is whole layer input set. Normalizing x mathematically is

$$\begin{aligned} \mathrm{Cov}[x] &= \mathcal{E}_{\mathcal{X}}[xx^T] - \mathrm{E}_{\mathcal{X}}[x]\mathrm{E}_{\mathcal{X}}[x]^T \\ \hat{x} &= \frac{x - \mathrm{E}_{\mathcal{X}}[x]}{\mathrm{Cov}[x]^{-1/2}} \end{aligned} \tag{35}$$

Calculating covariance matrix $\mathrm{Cov}[x]$ and its square root and inverse with all layer inputs over whole training set is too costly to perform in practice. Ioffe and Szegedy make two simplifications to make it feasible. Firstly, they assume that each dimension of feature vector is independent, and thus they can normalize each feature dimension individually. This avoids computing covariance matrix completely since they only have to make each scalar value of a single dimension have zero mean and unit variance. Secondly, they estimate mean and variance over a mini-batch instead of the whole training set. Now the normalization procedure simplifies to

$$\begin{aligned} \mathrm{Var}[x^{(k)}] &= \mathrm{E}_{\mathcal{B}}[(x^{(k)})^2] - (\mathrm{E}_{\mathcal{B}}[x^{(k)}])^2 \\ \hat{x}^{(k)} &= \frac{x^{(k)} - \mathrm{E}_{\mathcal{B}}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}} \quad \text{for } k = 1 \dots d \end{aligned} \tag{36}$$

where $\mathcal{B} = x_{1 \dots m}$ is mini-batch. In practice, $\mathrm{E}_{\mathcal{B}}[x]$ and $\mathrm{Var}[x]$ are estimated by

$$\begin{aligned} \mu_{\mathcal{B}} &= \frac{1}{m} \sum_{i=1}^{m} x_i \\ \sigma_{\mathcal{B}}^2 &= \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \end{aligned} \tag{37}$$

Furthermore, another linear transformation is added to normalized inputs to enhance representation power

$$y = \eta \hat{x} + \beta \tag{38}$$

where η and β are learnable parameters.

Ioffe and Szegedy [26] call the normalization step *Batch Normalizing Transform*, or BN transform for short. BN transform can be viewed as adding a normalization layer to inputs and feeding the normalized inputs $\hat{x}$ to a linear activated sub-network. Introducing BN transform has the following benefits. A high learning rate will increase the scale of parameters and may lead to either exploding or vanishing error signals [25]. By using BN transform, the scale of parameters have no effect on back-propagation, and larger weights get smaller gradients, both lead to stable update of model parameters. Moreover, as a side effect of BN transform, a training instance is affected by other instances in same mini-batch and thus no longer produces deterministic effect on update of the network. This increases the diversity of training data and reduces the risk of overfitting, and as a find in experiment, reduces the need of using dropout [26].

Ioffe and Szegedy applied BN transform on GoogLeNet and trained three versions to compare with original GoogLeNet on ILSVRC 2014 dataset: BN-Baseline, BN-x5 and BN-x30. BN-Baseline simply inserts BN transform before each activation layer of GoogLeNet; BN-x5 increases learning rate by 5 times upon BN-Baseline, and BN-x30 increases learning rate by 30 times upon BN-Baseline. Their experiment results show that BN-Baseline could match the original GoogLeNet performance in accuracy (72.7%) with only half number of training steps. BN-x5 achieves a better accuracy (73.0%) with 14 times fewer steps, while BN-x30 achieves the best accuracy (74.8%) with 5 times fewer steps. They also trained a 6 BN-x30 GoogLeNet ensemble and made classification by the average class probabilities. Their ensemble network achieves the lowest top-5 error (4.9%) among all reported results [26] .

6. CNN Features

Features extracted from CNN layers have become preferred generic descriptors to represent images [46, 53]. The convenience and effectiveness of obtaining deep image features with deep learning package like Caffe [28] has made deep features even more popular. A commonly accepted practice of obtaining good task specific deep features is to pre-training CNN on large-scale dataset and fine-tune with relatively small task dataset . In this section, we will review experiment results that have proven the distinctiveness of deep features obtained with CNNs.

Image feature, or image descriptor, is a low-dimensional vector used to represent image in a compressed form. It is not a new concept at all. Before thrive of deep learning in computer vision, many types of image features have been developed for specific tasks or general purpose. SIFT [37], GIST [41], HOG [11] and Bag-of-Words features are among those popular ones. In tasks such as two-view geometry and 3D-reconstruction when strong geometric constraints exist, SIFT features are incomparable in locating exact points. This is because SIFT features are highly optimized and engineered vectors in pixel grained level. But for tasks such as object recognition, image classification and instance retrieval, when course-grained generalizations are needed, CNN deep features have its own advantage over SIFT, GIST, HOG and Bag-of-Words features. Girshick *et al*. [21] gave an explanation why deep features could be better than shallow features in perspective of human recognition:

> "SIFT and HOG features are blockwise orientation hisograms, a representation we could associate roughly with complex celss in V1, the first cortical area in the primate visual pathway. But we also know that recognition occurs several stages downstream, which suggests that there might be hierarchical, multi-stage processes for computing features that are even more informative for visual recognition".

Razavian *et al*. [46] have conducted an experiment which compares the performance of using different image features to classification tasks. Here we briefly reviewed their work and results. ***CNN model*** The CNN model used by Razavian *et al*. is named OverFeat [50], which follows ConvNet with some small modifications like larger feature maps and smaller strides. OverFeat was pre-trained for classification task on data set of ImageNet [12] ILSVRC 2013. Here is the outline of the new classification tasks we want to perform. ***Datasets*** Two recognition datasets are used to compare deep features and traditional ones: Pascal VOC 2007 [18] for object classification and MIT-67 indoor scenes [45] for scene recognition. Pascal VOC 2007 contains about 10000 images of 20 classes. MIT-67 contains 15620 images of 67 indoor scene classes. ***Features and Classifiers*** The deep features are 4096-dimensional vectors extracted from last fully-connected layer of OverFeat, which represent the most distinctive information learned by CNN. The corresponding classifier was simply one-against-all linear SVMs (CNN-SVM), Baseline classifiers were GHM [9], AGS [15] and NUS [54]. GHM learns bag-of-words features of images. AGS learns features from subcategories by clustering images. NUS learns a codebook for SIFT, HOG and LBP descriptors from images. ***Experiment Results*** On Pascal VOC 2007 dataset, CNN-SVM, which gains benefits of using deep features, has better average precisions on 10 out of 20 classes. Razavian *et al*. [46] also compared the average precisions achieved by using different OverFeat CNN layer features as deep representations of images to train SVMs. The result is shown in Fig 9. Obviously the highest level deep feature, which is the last fully-connected layer of OverFeat, produces the best generalizations of images for classification tasks. On MIT-67 dataset, CNN-SVM out-

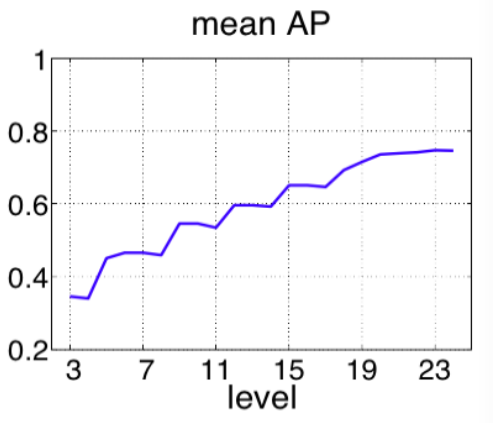


Figure 9: Mean image classification average precision on Pascal VOC 2007 classes [46] . The level indicates which layer of OverFeat CNN features are used.

performed most highly optimized methods for this particular dataset, like MLrep [14] and IFV [29]. Zeiler and Fergus have conducted similar experiment on Caltech-101 [19] and Caltech-256 [23] datasets. They keep layers 1-7 of ImageNet-trained ConvNet fixed and train a new softmax classifier on top using Caltech dataset. This is equivalent to extracting image features by ConvNet and training another linear classifier separately. They find on both datasets their deep features significantly outperform previously best reported result [65].

Discussion CNN features, as a continuous distributed representation of images, has been proved to surpass traditional hand-selected interpretable features such as HOG and SIFT in classification and recognition tasks. It is still hard to interpret such a large-dimensional deep feature which obvious limits theoretical analysis of its high abstraction power.

7. Motion Features

We now discuss a successful application of image features on classification of motions in videos. Video is a temporal sequence of images. Video is frame sequence is generally described by representation of certain visual words, such as bag-of-words representation of histogram of optical flows. Ryoo *et al*. [48] introduce a new feature representation called *pooled time series* (PoT) to describe motion information in first-person video with fixed-length *motion features*. PoT representation of motion videos is formed by pooling per-frame image features in a set of temporal windows and concatenating them together. They compared different per-frame feature descriptors: histogram of optical flows (HOF) [8], motion boundary histogram (MBH) [61], CNN and Overfeat deep features, as well as three feature representations: Bag-of-Words (BoW), Improved Fisher Vector (IFV) [42] and PoT.

The pipeline of generating PoT representation is briefly discussed here. At first image descriptors mentioned above are extracted from each video frame to form a sequence of n-dimensional vectors. Then PoT keeps track of value changes of each dimension over time. Two questions need to be addresses at this step: how can we know the time interval of a certain motion, and how do we record value changes. Their solution to first question is, a set of temporal filters is applied to each time series to capture information in a certain time segment. Their solution to second question is to perform pooling operations. Several types of pooling operations are defined: max, sum, count or sum of gradient changes. Finally, pooling results are concatenated to form final representation of videos. By following this pipeline, PoT converts video frames to a single finite-dimensional vector as input to classifiers for motion classification task.

Suppose there are m video frames $\{V^t\}_{t=1}^m$, each has n dimensions; k temporal filters, each covers a certain time window $\{[t_u^s, t_u^e]\}_{u=1}^k$ to form a temporal pyramid [10] in order to capture diverse temporal structures. Multiple pooling operators are applied to each filter and results are concatenated to form the final PoT feature representation $x = [x_i^{op_j}[t_u^s, t_u^e]]$, each element is interpreted as applying jth pooling operator to the uth time window of ith component of image frame descriptor. Finally, a SVM classifier is trained with PoT features to perform classification task. The design of PoT features brings several benefits of recognizing actions. By pooling from a pyramid of temporal filters, PoT can capture either short-term or long-term motions, and enhance representing power for high-level actions. Moreover, by using various pooling operators, diverse patterns of actions can be captured.

One particular interesting work of [48] is that multiple pooling operators are used to apply on temporal filters. Traditional max pooling and sum pooling operators (Σ) select the maximum value or sum over all values of a particular dimension of per-frame image feature vector over temporal window of this filter. Another two new pooling operators (Δ_1 and Δ_2) in [48] either counts the number of positive and negative gradient changes of values within temporal window or sums the values. These two new operators capture more information of changes of each feature dimension during time interval. Their experiment shows that best classification performance is achieved by PoT representation with sum, Σ and either Δ_1 or Δ_2 operators applied on temporal pyramid of filters.

Experiment result shows, by using any type image features, PoT representations outperforms BoW and IFV in classification accuracy. By using high-dimensional deep learning features, the superior of PoT over BoW and HOF is more significant. This is because BoW and HOF clus-

Figure 10: Image sequence of cycling.

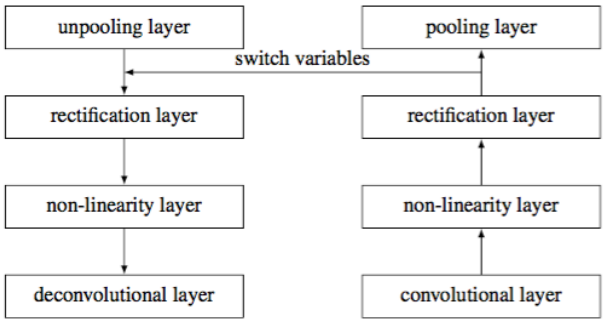


Figure 11: Structure of DeCNN taken from [65]. The right part is part of a CNN, and each layer is attached with a DeCNN layer at left.

ter similar deep feature vectors into single visual word and ignore small changes, but PoT captures these changes by temporal pooling. PoT which uses a pyramid of temporal filters and concatenates all pooled vectors outperform the one which considers only entire time interval and pools over it. In contrast, BoW and IVF do not get benefits by using pyramid of temporal filters for the same reason above.

Discussion PoT representation captures differences of feature vectors of adjacent video frames with high sample rate. Another interesting task is so-called "sparse motion" task coined by author of this survey. The task is to understand images of a certain action captured a wearable camera. Since the wearable camera typically takes images at a much lower rate, say 30s per frame, those images form a "sparse motion" which means only very few moments of the motion could be captured. Optical flow is hard to capture, or is highly unreliable because there is hardly any temporal connections between adjacent frames. For instance, the low level gradient changes of two adjacent frames can be useful for normal videos as we see in PoT, but can be meaningless for sparsely taken photo sequence. However, there are certainly motion-specific patterns can be captured in "sparse motion" image sequence. Figure 10 shows the cycling motion images. The order of each image in sequence is almost lost due to coarse sample rate. Order information is important to gradient pooling of PoT, but not for sum or max pooling. It is in the hope that by max or sum pooling of feature vectors from sparsely taken images, most distinctive patterns of this action are preserved while background or non-repeatable patterns of each image are suppressed.

8. CNN Deconvolution

Understanding feature maps in higher layers of a CNN is a difficult task. Feature maps at higher layers are pooled and rectified, and are of combinations of lower feature maps. The high abstraction of feature maps makes it hard to build a connection between feature maps and original images. Zeiler and Fergus propose a visualization method which discovers the input stimuli that excite individual feature maps [65]. The method is called multi-layered deconvolutional network, or deconvnet, but we will use the name DeCNN in accordance with our naming convention. The idea of DeCNN is to back-project feature activities to input image plane and show what input pattern actually gives rise to such activities. The name DeCNN comes from the fact that CNN maps input images to features while DeCNN does the opposite. In Zeiler's 2010 paper [66], a deconvnet was a unsupervised training model and its main task was to build mid-level image representations. But we only discuss the DeCNN used for visualization purpose which has no training step. A DeCNN is layer-wise attached to a CNN, sharing CNN's convolution kernels and pooling masks. Figure 11 is a good illustration of structure of DeCNN.

The DeCNN reconstructs feature maps in opposite direction. To examine a particular feature map, all other feature maps at same layer are set to zero and only pass this feature map as input to DeCNN layers attached. **Unpooling** layer follows what we discussed in Section 4.1: it inverts pooling layer by placing reconstructions from layer above on positions of local maxima during forward step. **ReLU** layer is simply appended after unpooling layer to pass through only positive reconstructions. **Deconvolution** layer inverts convolutional layer by transposing (flipping vertically and horizontally) convolutional kernels as deconvolutional kernels and applies on rectified feature maps. The logic of using transposed kernel is that in signal processing realm, a Hermitian or conjugate transpose of a filter is called a "matched filter". A matched filter is "optimal linear filter for maximizing the signal to noise ratio (SNR) in the presence of additive stochastic noise" [3]. In another word, the transposed filter is used to approximately reconstruct feature maps before convolution.

Zeiler and Fergus used DeCNN to visualize the feature activations upon ImageNet validation set. After training step is done, they randomly chose a subset of feature maps

at each layer from 2 to 5, and selected top 9 activations of each feature map when validation set of images are forwarded into CNN. These 9 activations of each feature map are projected back to pixel space to produce the reconstructions. In Figure 12, 9 reconstructions of each selected feature map at each layer from 2 to 5 are shown. Figure 12 shows several interesting facts. Each feature map is strongly responding to one pattern, as its reconstructions of 9 top activations share this similar pattern. Lower layers could capture corners, edges and simple shapes like circles, as shown in Figure 12(a). Middle layers could capture greater transform invariance, e.g., Row 1 Column 1 of layer 3 captures grid patterns, as shown in Figure 12(b). Higher layers could capture very discriminative features of images, e.g., dog faces and bird legs at layer 4, and dog eyes, ears, wheels at layer 5, as shown in Figure 12(c).

One success application of visualization technique is to help improve CNN architectures. In Figure 13, the left image shows visualization result of ConvNet 2nd layer features. The aliasing artifacts indicate the too large convolution kernel size 11 and the too large stride 4 [65]. Zeiler suggested decreasing kernel size to 7 and stride to 2, and produced more natural feature maps as shown in right image. The top-5 classification performance is also reduced by 1.7%.

9. Region Proposal

Krizhevsky *et al*. [32] demonstrated the advance of CNN on image classification tasks, and later, the power of CNN has been generalized to object detection tasks. We will discuss the state of the art objection detection method which is called Regions with CNN features (R-CNN) proposed by Girshick *et al*. [21] in 2014. R-CNN decomposes object detection to two main steps. In first step, low-level image information such as color and texture are used to generate object location proposals without incorporating any category concept. In second step, CNN classifiers are used to identify object categories at locations proposed in first step. "Such a two stage approach leverages the accuracy of bounding box segmentation with low-level cues, as well as the highly powerful classification power of CNNs" [55]. The pipeline of R-CNN is shown in Fig 14. In first step, the method used to generate category-agnostic region proposals is selective search [57, 62], which generates about 2000 region proposals for each image. Each proposed region is then cropped out of original image and warped into 227×227 image patch which is fit for CNN input, and fed forward through a standard CNN. The output of last fully-connected layers are extracted out as region features. At training stage, training image regions together with their known category labels are used to supervisely train SVM classifiers for each category. Having these trained category-specific SVM classifiers, at testing stage, regions features extracted from testing images are scored and classified to most probable categories. Finally, a greedy non-maximum method suppresses non-maximum regions for each class independently. R-CNN is very scalable with growth of category number. The only category-related computation during testing time is the matrix product of region features with SVM weight matrices, which grows linearly with category number N. R-CNN is a successful application of CNN deep features which is used on image location task.

The CNN used to generate region features is pre-trained on ILSVRC 2012 dataset, followed by fine-tuning on PASCAL data set. This is critical because PASCAL dataset has not enough labeled objects and thus not capable of training a deep CNN from scratch. Though ILSVRC 2012 dataset only includes image category labels but not specific to object level, pre-training CNN with ILSVRC dataset is viewed as image-level annotation with entire image as an object (bounding box is whole image). This pre-trained CNN has an ILSVRC ImageNet-specific 1000-way classification layer. At fine-tuning stage, this classification layer is replaced with task-specific classification layer, e.g., a 20-way classification layer corresponding to 20 classes for PASCAL dataset. The idea of pre-training a general CNN and fine-tuning on task dataset now becomes a standard way of training a domain-specific CNN if task dataset is insufficient. Caffe deep learning package [28] makes this even more simpler by providing parameters and structures of pre-trained general-purpose CNNs such as ConvNet and GoogLeNet, and thus saves the effort of pre-training step.

Experiment results on ILSVRC 2013 detection dataset showed that R-CNN has significant better mean average precision (mAP) than second best result produced by OverFeat [50]. The authors also compared the mAP achieved by R-CNN with fine-tuned CNN and without fine-tuned CNN. Fine-tuning increases mAP by 8% on PASCAL VOC 2007 dataset. In addition, it is reported that the choice of CNN architecture results in a large difference in performance. The authors compared the results of two R-CNNs with ConvNet and VGGNet as their basic CNN respectively. VGGNet has a much deeper structure than ConvNet, as shown in Table 1. R-CNN with VGGNet structure outperforms R-CNN with ConvNet structure by 7.5%, at the cost of about 7 times longer time in forward propagating.

10. Neural Networks in Natural Language Processing

In this section, we will discuss recent progress in applying deep neural networks to natural language processing (NLP) realm. This section is related to this survey of CNNs for two reasons: both DNNs in NLP and CNNs in Computer Vision share similar thoughts and design in structure, and many efforts have been made to combine deep neural networks to model joint distribution of image and text and

perform classification or prediction tasks. In Section 10.1 we will introduce two types of neural networks as word encoders which produce word features as CNNs produce image features. In Section 10.2 we will introduce RNNs and LSTMs as sentence encoders which produce sentence features. In Section 10.3 we will discuss recent work in image-sentence prediction by combining deep features of images and sentences.

10.1. Word Representation

In Section 6, we have shown that CNNs can produce rich representation of an image with a fixed-length feature vector, and this deep feature vector has been applied successfully to many computer vision tasks. CNNs are thus viewed as an effective image encoder. In Natural Language Processing realm, several specially structured neural networks have been developed recently as encoders for words, such as CBOW and Skig-gram [39, 38]. They have also shown deep network's power in representing words' syntactic and semantic relationships. Both CBOW and Skig-gram use neural network as basic model of training a word encoder.

CBOW uses a word's context of D future words and D history words as input, and trains a neural network to predict current word which is in middle. The hidden layer of a neural network is replaced with a projection layer, which projects input words into vectors and averages them to construct the middle word vector. A log-linear classifier is built upon middle word vector, and error signals are back-propagated to tune the weights of projection layer. CBOW is similar to bag-of-words model in that the order of words has no influence of projection. However, CBOW uses continuous distributed representation of the context, and shows to contain more rich representations of a word. **Skip-gram** is different from CBOW that it uses current word as input to projection layer and predicts C words before and C words after current word. A log-linear classifier is built upon projection layer to train projection weights. In Skip-gram, the order of words has influence on word representation. The motivation of skip-gram is based on the assumption that words with similar contexts are likely to be both semantically and syntactically similar. The similarity of two words is measured by inner product of word vectors. Later, [67] extends this idea to form sentence vectors and measure similarity of two sentences.

Mikolov *et al*. [38] reported that CBOW achieves the best performance on syntactic questions, slightly better than Skip-gram; Skip-gram achieves the best performance on semantic questions, and significant better than all the other models. Some of the semantic and syntactic questions are shown in Figure 15.

The common ideas shared between deep image representation and deep word representation are that both incorporate contextual information to represent a unit in a context, as pixel in an image or word in a sentence, and both produce continuous distributed representations which have been shown their richness.

10.2. Sentence Representation

Let's consider a recent interesting task in NLP and Computer Vision, which asks to generate full sentence descriptions for images automatically. There are two problems need to be solved first. The first question is how do we use image information and map image and sentence into a common space. The second question is what is the language model of generating a sentence. We will consider a class of data-driven methods which are proposed recently. In 2014, the topic of generating captions for images is the main theme of aligning texts with images [60, 30, 31]. In 2015, more works have been focusing on generating descriptions for short videos clips [44, 59, 58], and aligning book paragraphs with movie shots [67].

Let's address the first question as "can we map image and sentence into a same space by using image representations and sentence representations". As we have discussed in Section 6 and Section 10.1, images and words can be represented as feature vectors of fixed length. A sentence representation can be generated by stacking word vectors together. A simple linear transformation upon image and word feature vectors can easily align transformed features to a common space. We could use word vectors of CBOW or Skip-gram pre-trained on a large corpus, but several reasons lead to finding a better solution of modeling sentence representation: words are not independent in sentences, so simply stacking word vectors is not enough; we want to use full context of a word in the sentence as well as the order (time) information; we want to predict a complete sentence, so the ability to predict an end token is essential. Let's address the second question as "by using a date-driven method, is there a language model that could produce conditional probability distribution over whole dictionary for next word, based on all previously observed words and the image". Recently, a rediscovery of *Recurrent Neural Networs* (RNN) [16, 22], and its modified version *Long Short-Term Memory* (LSTM) [25, 22] have shown that they are good candidates for both tasks.

Let's denote an input sentence as a sequence of words $x = (x_1, x_2, \dots, x_t, \dots)$, and the hidden state of a RNN at time t as h_t. The hidden state is updated each time a new word is observed: $h_t = \text{LSTM}(h_{t-1}, x_t)$. In RNNs or LSTMs, hidden units are employed to store hidden states and develop internal representations for the patterns of input sequences [16]. Hidden units not only preserve previous states, but also receive external input observed at current time. This design purpose determines that hidden units must be recurrently connected with themselves, as shown in Figure 16 taken from [22].

Recent works prefer LSTM rather than RNN, for the reason RNN may suffer from vanishing or exploding gradients [25], which leads to weights oscillating or stopping learning. This is more likely to happen during back-propagation of a long neural network, for the intuition that accumulated error signal passed through a long sequence path is exponentially scaled at each layer, and tends to blow up or diminish to zero. LSTM minimizes the negative impact of degrading gradient effects by a specially designed hidden unit, which is called *memory block*. Memory block contains three *gates* and a *memory cell*, as shown in Figure 17. Memory cell c is used to *remember* important knowledge observed up to current step. Memory block can choose to read or not read current input, remember or forget current cell value, and whether or not output the new cell value. These behaviors of memory block are controlled by three types of gates: input gate i, forget gate f, and output gate o. During back-propagation, output gate controls error signals flow into memory cell and thus has to learn which errors should influence memory cell. Accordingly, input gate has to learn when to release errors and back-propagate. Only error signal arriving at memory cell gets back-propagated by proper scaling; error signals of gates' weights are not back-propagated. This achieves a so-called *constant error flow* and avoids vanishing or exploding gradients [25].

10.3. Image-Sentence Alignment and Prediction

Having introduced RNNs and LSTMs, we now describe the whole process of sentence prediction. Deep image feature vector for input image is firstly extracted out with a CNN model. In training stage, this image feature vector is the initial input to LSTM followed by words in training sentence. We can think LSTM in its unrolled form, as shown in Figure 18 taken from [60]. In Figure 18, a memory block is created for image feature vector and each word in training sentence in sequence. All such memory blocks share the same set of parameters that are updated at each time step. Output of LSTM memory block at each time step is interpreted by softmax layer as probability distribution of words in a pre-built dictionary. During back-propagation, error signal from softmax loss of word prediction at this step, as well as error signal back-propagated from later steps are combined together to update parameters of LSTM memory block. In testing stage, image feature vector is the only external input to a pre-trained LSTM. A forward pass of LSTM will generate predicted words in sequence to complete a sentence prediction. The termination condition is that LSTM predicts a period or a special stopping word, which is also appended to training data to signal the end of a sentence.

11. Conclusion

CNN, as well as general neural networks, experience ups and downs in different stages of history. CNN is obviously at its thrive today, and it redefines many of state of the arts methods in computer vision. We are lucky or unlucky in this rapidly changing time where numerous innovative ideas are created as fast as some of them are forgotten. This is an exciting era.

References

[1] Convolutional neural networks (lenet). `http://deeplearning.net/tutorial/lenet.html`. Accessed: 2015-08-11.

[2] Cs231n convolutional neural networks for visual recognition. `http://http://cs231n.github.io/convolutional-networks/#overview`. Accessed: 2015-08-12.

[3] Matched filter — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Matched_filter`. Accessed: 2015-07-29.

[4] C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[5] J. Bouvrie. Notes on convolutional neural networks. 2006.

[6] R. H. Byrd, S. Hansen, J. Nocedal, and Y. Singer. A stochastic quasi-newton method for large-scale optimization. *arXiv preprint arXiv:1401.7020*, 2014.

[7] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014.

[8] R. Chaudhry, A. Ravichandran, G. Hager, and R. Vidal. Histograms of oriented optical flow and binet-cauchy kernels on nonlinear dynamical systems for the recognition of human actions. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1932–1939. IEEE, 2009.

[9] Q. Chen, Z. Song, Y. Hua, Z. Huang, and S. Yan. Hierarchical matching with side information for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3426–3433. IEEE, 2012.

[10] J. Choi, W. J. Jeon, and S.-C. Lee. Spatio-temporal pyramid matching for sports videos. In *Proceedings of the 1st ACM international conference on Multimedia information retrieval*, pages 291–297. ACM, 2008.

[11] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[13] M. Denil, B. Shakibi, L. Dinh, N. de Freitas, et al. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, pages 2148–2156, 2013.

[14] C. Doersch, A. Gupta, and A. A. Efros. Mid-level visual element discovery as discriminative mode seeking. In *Advances in Neural Information Processing Systems*, pages 494–502, 2013.

[15] J. Dong, W. Xia, Q. Chen, J. Feng, Z. Huang, and S. Yan. Subcategory-aware object classification. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 827–834. IEEE, 2013.

[16] J. L. Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[17] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. Scalable object detection using deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 2155–2162. IEEE, 2014.

[18] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[19] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding*, 106(1):59–70, 2007.

[20] R. Girshick. Fast r-cnn. In *International Conference on Computer Vision (ICCV)*, 2015.

[21] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 580–587. IEEE, 2014.

[22] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[23] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. 2007.

[24] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer Series in Statistics. Springer, 2009.

[25] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[26] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[27] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE, 2009.

[28] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[29] M. Juneja, A. Vedaldi, C. Jawahar, and A. Zisserman. Blocks that shout: Distinctive parts for scene classification. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 923–930. IEEE, 2013.

[30] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. *arXiv preprint arXiv:1412.2306*, 2014.

[31] A. Karpathy, A. Joulin, and F. F. Li. Deep fragment embeddings for bidirectional image sentence mapping. In *Advances in neural information processing systems*, pages 1889–1897, 2014.

[32] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[33] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[34] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[35] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky. Sparse convolutional neural networks. June 2015.

[36] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *arXiv preprint arXiv:1411.4038*, 2014.

[37] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[38] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[39] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[40] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[41] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International journal of computer vision*, 42(3):145–175, 2001.

[42] F. Perronnin, J. Sánchez, and T. Mensink. Improving the fisher kernel for large-scale image classification. In *Computer Vision–ECCV 2010*, pages 143–156. Springer, 2010.

[43] K. B. Petersen et al. The matrix cookbook.

[44] H. Pirsiavash, C. Vondrick, and A. Torralba. Inferring the why in images. *arXiv preprint arXiv:1406.5472*, 2014.

[45] A. Quattoni and A. Torralba. Recognizing indoor scenes. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 413–420. IEEE, 2009.

[46] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pages 512–519. IEEE, 2014.

[47] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015.

[48] M. S. Ryoo, B. Rothrock, and L. Matthies. Pooled motion features for first-person videos. *arXiv preprint arXiv:1412.6505*, 2014.

[49] P. Sermanet, S. Chintala, and Y. LeCun. Convolutional neural networks applied to house numbers digit classification. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 3288–3291. IEEE, 2012.

[50] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[51] P. Sermanet and Y. LeCun. Traffic sign recognition with multi-scale convolutional networks. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 2809–2813. IEEE, 2011.

[52] H. Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

[53] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[54] Z. Song, Q. Chen, Z. Huang, Y. Hua, and S. Yan. Contextualizing object detection and classification. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1585–1592. IEEE, 2011.

[55] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

[56] C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. In *Advances in Neural Information Processing Systems*, pages 2553–2561, 2013.

[57] J. R. Uijlings, K. E. van de Sande, T. Gevers, and A. W. Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[58] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko. Sequence to sequence–video to text. *arXiv preprint arXiv:1505.00487*, 2015.

[59] S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. Mooney, and K. Saenko. Translating videos to natural language using deep recurrent neural networks. *arXiv preprint arXiv:1412.4729*, 2014.

[60] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. *arXiv preprint arXiv:1411.4555*, 2014.

[61] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu. Dense trajectories and motion boundary descriptors for action recognition. *International journal of computer vision*, 103(1):60–79, 2013.

[62] X. Wang, M. Yang, S. Zhu, and Y. Lin. Regionlets for generic object detection. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 17–24. IEEE, 2013.

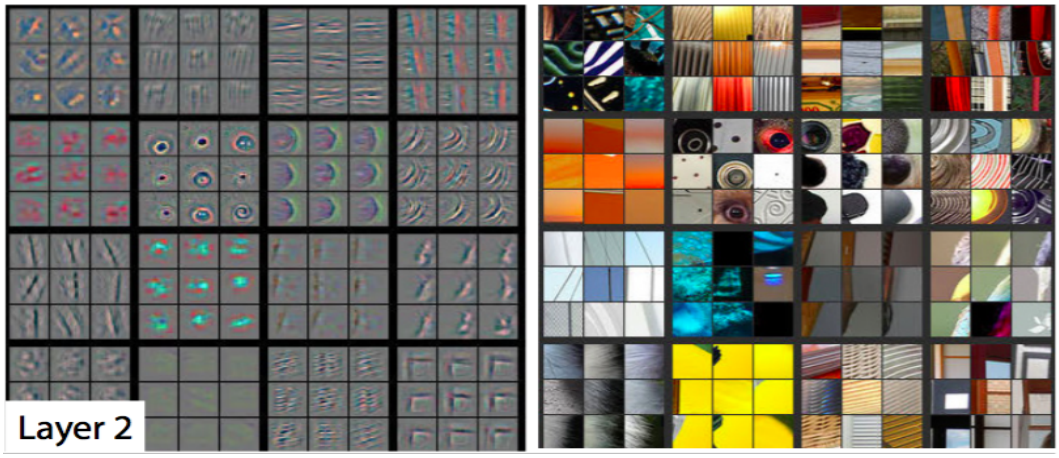
[63] S. J. Wright. *Numerical optimization*, volume 2.

[64] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

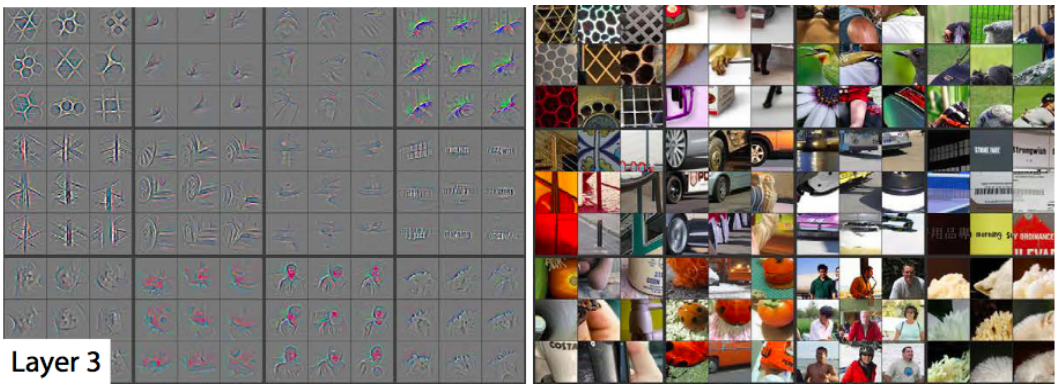
[65] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014*, pages 818–833. Springer, 2014.

[66] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus. Deconvolutional networks. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2528–2535. IEEE, 2010.

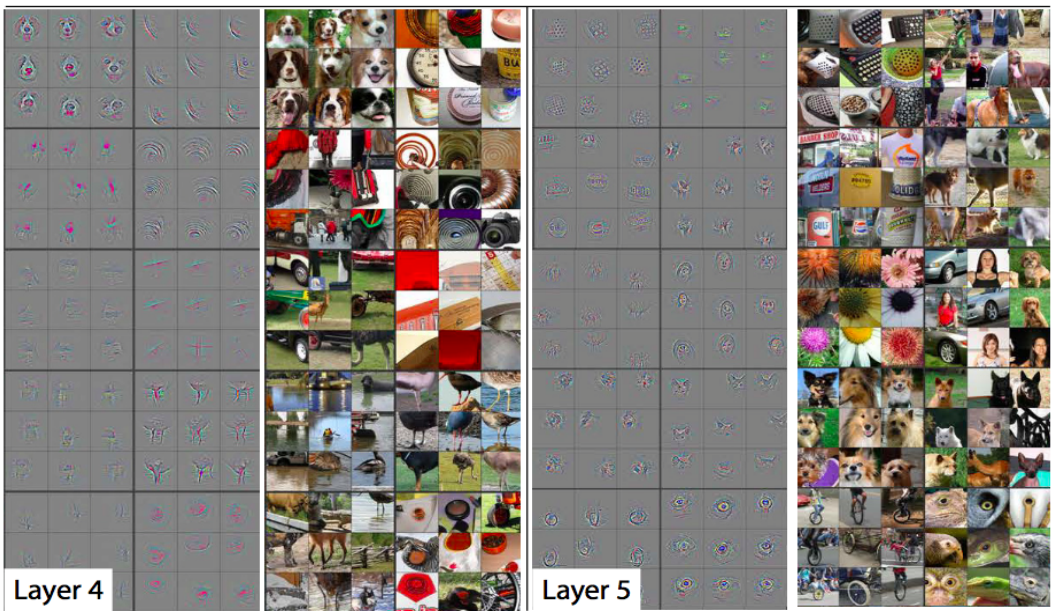
[67] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *arXiv preprint arXiv:1506.06724*, 2015.



(a) Layer 2 top 9 activations of feature maps and corresponding image patches



(b) Layer 3 top 9 activations of feature maps and corresponding image patches



(c) Layer 4,5 top 9 activations of feature maps and corresponding image patches

Figure 12: Visualization of feature maps projected back in pixel space as shown in [65]. For a subset of feature maps at each layer from 2 to 5, top 9 activations of each selected feature maps on validation set are projected back to pixel space using DeCNN. Reconstructed feature maps and their corresponding image patches are shown.

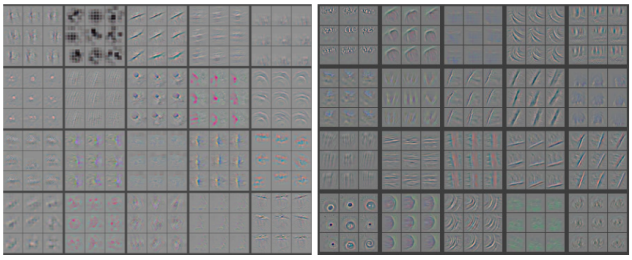


Figure 13: Visualization of 2nd layer features from Krizhevsky's ConvNet [32] and from Zeiler's suggested CNN architecture [65].

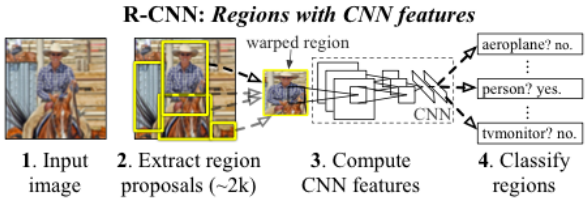


Figure 14: Object detection procedures of R-CNN taken from [21].

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

Figure 15: Some semantic and syntactic questions used by Mikolov *et al*. [38] to test CBOW and Skip-gram word features. A semantic question is like "what is the city has the relationship with Norway as Athens has the same relationship with Greece". Their method of answering this question with word vectors is to compute vector $X = vector(\text{“}Norway\text{”}) - vector(\text{“}Greece\text{”}) + vector(\text{“}Athens\text{”})$. Then they find the word w with closest word vector X' with X in vector space measured by cosine distance as the answer to the question.

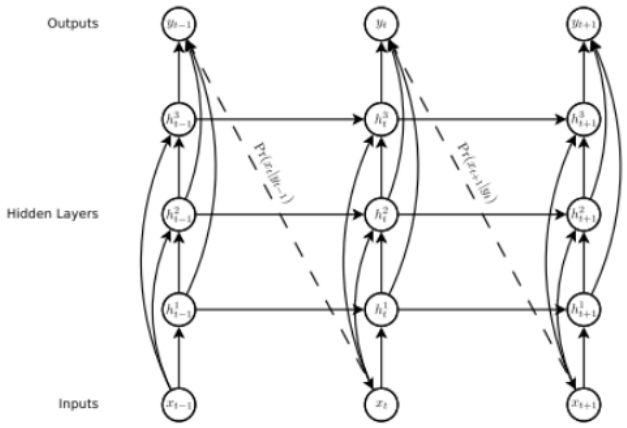


Figure 16: Demonstration of RNN structure taken from [22]. Hidden units receive internal inputs from previous hidden states and external input observed at current time. In horizontal direction, hidden units are unrolled with time; in vertical direction, three layers of hidden units are stacked in a hierarchical way.

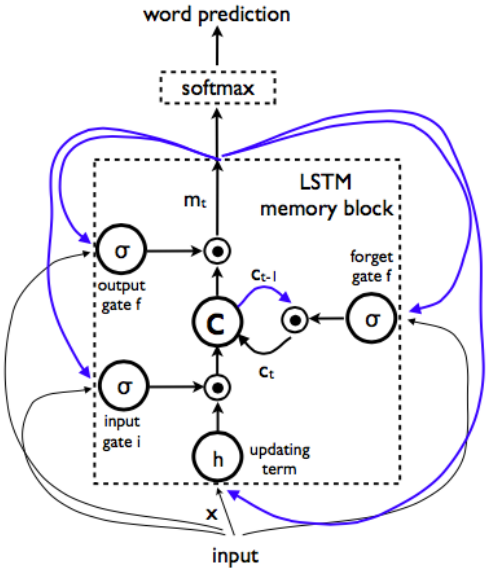


Figure 17: Demonstration of LSTM memory block taken from [60]. Memory cell c is at the heart of the block. Three gates control the data flow through memory cell. In word prediction scenario, at training time, input at time t is from output at t-1 and t-th ground truth word in training sentence; at testing time, output at t-1 is the only input into memory cell, and output at t is fed into softmax layer for word prediction.

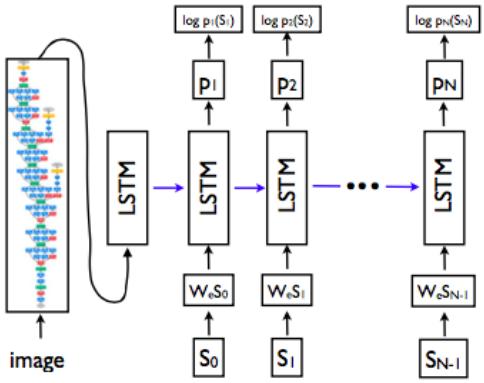


Figure 18: Demonstration of unrolled form of LSTM model taken from [60]. The leftmost part is a CNN which generates deep feature vector for input image, which is fed into LSTM block. Each LSTM block is created for image feature input and each word input. All LSTM blocks share the same set of parameters, and they are shown to be recurrently connected. Outputs are fed into softmax layer to produce next word predictions. In back-propagation, error signals are propagated from right to left, and are combined with the new error signal generated at softmax layer at each step.