An Overview of Convolutional Neural Network Architectures for Deep Learning

John Murphy[1]

Microway, Inc.

Fall 2016

[1] jmurphy@microway.com

Abstract

Since AlexNet was developed and applied to the ImageNet classification competition in 2012 [1], the quantity of research on convolutional networks for deep learning applications has increased remarkably. In 2015, the top 5 classification error was reduced to 3.57%, with Microsoft's Residual Network [2]. The previous top 5 classification error was 6.67%, achieved by GoogLeNet [3]. In recent years, new artificial neural network architectures have been developed which improve upon previous architectures. Specifically, these are the inception modules in GoogLeNet, and residual networks, in Microsoft's ResNet [2]. Here we will examine convolutional neural networks (convnets) for image recognition, and then provide an explanation for their architecture. The role of various convnet hyperparameters will be examined. The question of how to correctly size a neural network, in terms of the number of layers, and layer size, for example, will be considered. An example for determining GPU memory required for training a defined network architecture is presented. The training method of backpropagation will be discussed in the context of past and recent developments which have improved training effectiveness. Other techniques and considerations related to network training, such as choosing an activation function, and proper weight initialization, are discussed briefly. Finally, recent developments in convnet architecture are reviewed.

Contents

List of Figures

List of Tables

1 Introduction

Research in artificial neural networks began almost 80 years ago [4]. For many years, there was no widely accepted biological model for visual neural networks, until experimental work elucidated the structure and function of the mammalian visual cortex [5]. Thereafter, theoreticians constructed models bearing similarity to biological neural networks [6]. While theoretical progress continued with the development of methods for training convnets, there was no practical method for training large networks yet in place. With the development of LeNet, by Yann LeCunn,*et al.*, the first practical convnet was deployed for recognizing hand-written digits on bank checks [7]. This paper will discuss some of the major network parameters and learning techniques related to convnets.

2 Convnet Architecture

Since AlexNet was demonstrated, in 2012, to outperform all other methods for visual classification, convnets began to attract great attention from the research community[1]. By performing matrix operations in parallel on Graphical Processing Units (GPUs) in consumer desktop computers, it became possible to train larger networks in order to classify across a large number of classes, taken from ImageNet [8]. Since AlexNet, research activity in Deep Learning has increased remarkably.

Large neural networks have the ability to emulate the behavior of arbitrary, complex, non-linear functions. When trained effectively, the functions emulated by convnets have the ability to map complex, high-dimensional image data into a much lower dimensional space of finite distinct categories, consisting of hundreds, or thousands, of object classes. The distinction between categories can be nuanced, such as a Malamute, instead of a Siberian Husky, for example (see Figure 1).

Convnets are also used for object detection, scene recognition, human pose estimation, video caption generation, speech recognition, language translation, among other tasks. The idea for using *local filters* (instead of fully connected layers) with convnets first arose from the observation that, in images, there is no meaningful structure on larger distance scales. "Larger" is a relative term here, being dependent on the image composition. At large distances, pixels are randomly related. But, at shorter distances, they are correlated. This indicates that meaningful structure is found within local image patches. This is not surprising since objects are defined by their local structure, and not by what is distant from them. A vase, for example, can be defined by its contours. Pixels far away from the vase provide no further information about the vase, unless by

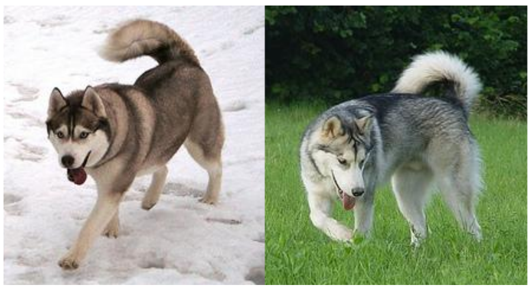
Figure 1: A Malamute can, in many instances, not be immediately discernable from a Siberian Husky (Malamute is on right).

coincidence, a vase, for example, is always positioned adjacent to a small cube. With such an image set, an image classifier trained to recognize vases may also use elements of the cube to determine whether a vase is in the image.

In fact, before training, it is not clear what sort of features a network will select in order to improve recognition. One network trained by [9], for example, could recognize race cars, in part, by developing detectors for the text content of advertisement decals stuck to the sides of them- not necessarily something one would immediately associate with a car. But since it was a distinguishing feature of race cars, the network learned to recognize the text in advertisements, and then use this feature, in a distributed representation, to activate the network output indicating race car category, and not some other type of car. Similar sorts of differences are used by networks to sort out recognition of similar breeds of dogs (such as Siberian Huskies and Malamutes).

The structural information contained in local regions of images motivated the use of patch-like connections between layers, instead of full connections. This is the same as using connections where every weight is zero, except, possibly, for weights within some patch (filter, or kernel) region. The zeros represent the fact that information outside of the patch is not determining anything about structure present in the filter's local area. Only the neurons within a (square) patch/filter/kernel would be fully connected to single neurons in the next layer (see Figure 2). By reducing the number of connections, the number of weight parameters is reduced. This reduces computation, both in training and during deployment.

Further, filters keep the same weights as they are convolved through various positions in a feature map. This means that a filter is using the same weights for detecting the same sort of feature at multiple locations in a feature map. This helps to further reduce the number of parameters and also helps to prevent overfitting of the network to the testing data. Using local filters (patch connections) instead of full connections also reduces overfitting.

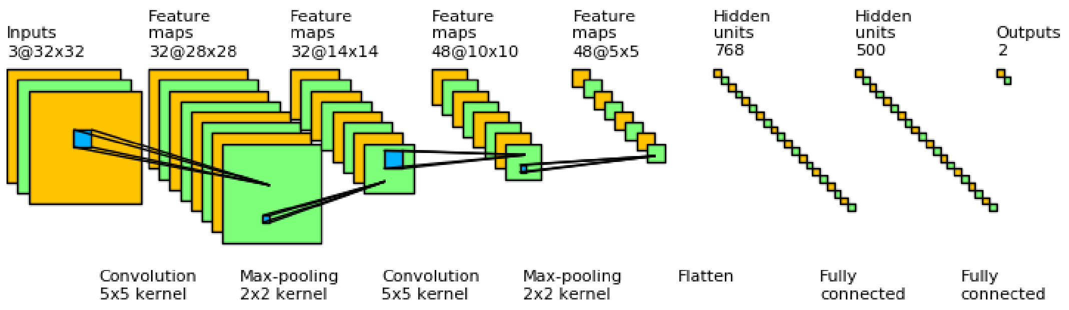

Figure 2: A convolutional neural network with max pool layers. Here, max pooling chooses the highest pixel value in a 2×2 patch translated in increments of 2 pixels. This reduced the number of pixels by a factor of 4. See the section below on Pooling for more details on max pooling).

3 Filter Convolution

How a filter is convolved across an image depends on several parameters, which are described here. In determining an appropriate filter size, stride, and zero padding parameter values (described below), some basic arithmetic must be done to determine if the convolution will work with the feature map dimensions. Figure 3 shows an example of how a convolution is computed with a filter across the top three filter positions in an image.

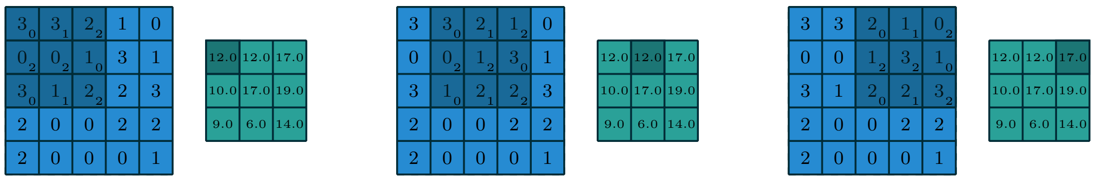

Figure 3: Computing the output values of a discrete convolution. For the subplots, dark blue squares indicate neurons in the filter region. Dark green squares indicate the output neuron for which the total input activation is computed.

3.1 Filter Size

Almost all filters found in the literature are square. The size of the filter with respect to the size of the image (or activation layer), determines what features can be detected by the filters. In AlexNet, for example, the input layer has 11×11 filters, applied to images which are 224×224 pixels. Each filter length, on side, comprises only 4.3% of the (square) image side length. These filters in the first layers cannot extract features which span more than 0.24% of the input image area.

3.2 Padding

Padding is a margin of zero values which are placed around the image. Padding depth can be set so that the output from the current convolutional layer does not become smaller in size, after convolution. Figure 4 below shows this effect.

With many successive convolutional layers, the reduction in output dimension can become a problem, since some area is lost at every convolution. In a convnet having many layers, this effect of shrinking the output can be countered by zero padding the input. Zero padding, as shown in Figure 4, can have the effect of canceling dimensional reduction, and maintaining the input dimension at the output. In networks having many layers, this approach may be necessary in order to prevent outputs from becoming too reduced.

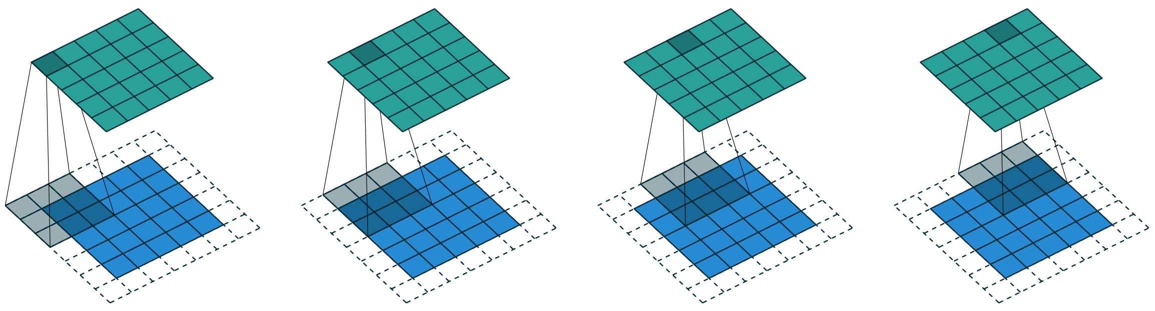
Figure 4: With zero padding, the effect of output size reduction is counteracted to maintain the input size at the output.

3.3 Stride

Stride, usually denoted S, is a parameter which describes how many pixels a filter will be translated horizontally (and then vertically, while being convolved across the next row). In some network architectures, stride is used instead of max pooling in order to reduce layer size. This approach was taken in GoogLeNet [3].

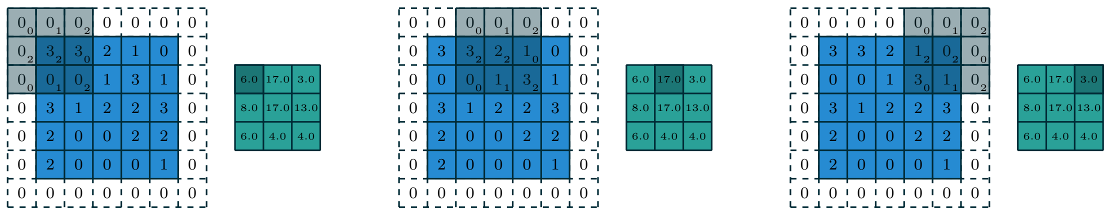
Figure 5: Convolution with zero padding and stride > 1. Only the fist three subplots are shown here. Padding is of depth 1, stride is 2, and filter size is 3×3

3.4 Convolution Arithmetic

When considering a network design, it is necessary to know how the parameters will affect output dimension. We will use the variable N as the length on side of the current activation layer. P is the padding depth. F is the length on side of filter. S is the length of stride. The equations below relate the size of the output to the size of the input, filter size, padding, and stride.

$$L_{output} = (N - F)/S + 1 \tag{1}$$

$$L_{output} = (N + 2P - F)/S + 1 \tag{2}$$

Equation 1 computes the side length of an activation layer after convolution with no zero padding. Equation 2 accounts for zero padding. Zero padding can counteract the normal reduction in output area after convolution. This is discussed in more detail further below, under 'Determining Compute Resources Required for a Convnet'.

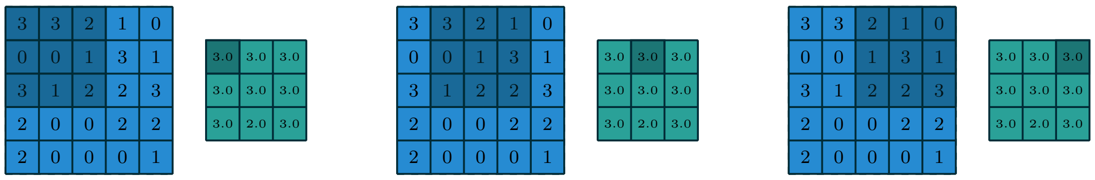

Figure 6: This is an example of computing the maxpool output at three 3×3 patch locations on an activation layer. Max pooling is applied on 3×3 patches. Max pooling increases the intensity of pixels upon pooling, since only the maximum intensity pixel was chosen from each pooled area.

For computational reasons, it is best to set the number of filters to be a power of 2. This helps to vectorize computations. Filter sizes are usually odd numbers. This is to have a filter center pixel. This way, the filter is applied to some position that exists on the input tensor.

Note that after convolving a 5×5 pixel image with a 3×3 filter in Figure 3, the output (activation layer) has dimensions of 3×3 pixels, which is consistent with Equation 1.

3.5 Pooling

Pooling layers were developed to reduce the number of parameters needed to describe layers deeper in the network. Pooling also reduces the number of computations required for training the network, or for simply running it forward during a classification task. Pooling layers provide some limited amount of translational and rotational invariance. However, it will not account for large translational or rotational perturbations, such as a face flipped by $180°$. Perturbations like these could only be detected if images of faces, rotated by $180°$, were present in the original training set. Accounting for all possible orientations of an object would require more filters, and therefore more weight parameters would be required.

A larger question, however, is whether a simple convnet might be the most appropriate network architecture for accounting for various representations of the same object class. In 2015, Google developed the Inception module which was an architectural module designed for detecting features with some scale invariance[3]. This is discussed further below. The inception module is an extension to previous convolutional network architectures. Other functionally distinct network modules have been developed for other purposes, such as retaining information in a recurring neural network. Examples of this include the Long Short Term Memory module (LSTM), and the gated recurring unit (GRU)[10, 11]. It is speculated here that more effective modules for capturing translational, and rotational invariance may be developed.

Figure 6 shows max pooling applied on 3×3 patches. Because max pooling increases the intensity of pixels, one might expect that a better way to pool would be to keep the average intensity per pixel the same. However, this was discovered to not generally be the case. Some convnet architectures, such as GoogLeNet, are replacing max pooling layers by using larger strides.

4 Network Design Questions

1. How is an adequate number of filters determined?
2. Given a number of image categories (classes), and the number of images per class, what is the learning capacity of a neural network?
3. How can image classification tasks be expressed in terms of learning capacity requirements?

More classes imply that more features must be learned. More features require more filters, and more filters require more parameters. It would seem that the depth of the network should be determined by some measure of complexity common to almost all images in the data set. Residual networks operate better at arbitrarily large numbers of layers. The relation speculated here between a sufficient number of layers and a complexity measure of images in the dataset applies to regular convnets, and not to residual networks. If the number of hidden layers required could be determined at the outset, then only the number and size of filters for each layer would be left to determine. The number of classes would primarily determine the number of filters. However, because of the effectiveness of *distributed representation*, the number of filters required will not increase linearly with the number of classes. It may instead increase in proportional to some fractional power of it.

During training, patches low in the network usually become resolved into edge detectors [1]. This helps the network to then build more complex representations in the filters for the next layer. Also, the small size of filters used in the first layer leaves them little flexibility but to pick out only small, nearly featureless structures, such as edges, or blobs of color, for instance. Filters for the next layer then become combinations of filter features from the current layer.

5 Convnet Training

During network training, the filter weights are adjusted, so as to improve the classification performance of the network. This can be done using a method called backpropagation, where the gradient of an error function is computed with respect to all network weights, going all the way to the input connections of the network. Network weights are updated by the following equation relating the step size to the gradient and the learning rate, denoted η.

$$W_{\text{new}} = W - \eta \frac{dE}{dW} \tag{3}$$

An error (or "scoring") function can be expressed as a sum of squared differences between the network's output and the correct output, over all discrete points in the output. This sort of scoring function works for cases where the network output is a vector, matrix, or tensor of continuous real values.

$$E(W,b) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\|h_{W,b}(x^{(i)}) - y^{(i)}\|^2 \tag{4}$$

The gradient of this scoring function would be taken in Equation 3.

For classification networks, the scoring function is computed differently. Because the output is a one-of-N vector, where the highest value component represents the network's best estimation of the image class (i.e., car, boat, airplane, etc.), an error function for categorical output is needed. This is called the categorical cross entropy [12], which will not be explained here, other than to state that it is used to estimate the error in categorical outputs, such as with image classification convnets.

5.1 Optimization Methods

Names for optimization methods, both old and new, are listed below.

Stochastic Gradient Descent (SGD)
SGD is used with batch processing, where the gradient of a batch is averaged over and used for making network weight updates.

SGD with Momentum
When momentum is added, jitter in steep gradient directions is reduced, while momentum is built in shallow directions.

Nesterov's Accelerated Gradient (NAG)
NAG evaluates at the next position indicated by the momentum. This is called a "lookahead" gradient. In practice, NAG almost always converges faster than SGD with momentum.

AdaGrad
AdaGrad scales the gradient inversely proportional to the square root of the sum of the square of previous gradients. In practice, this means that you can have a larger learning rate than in steep directions. Since the sum of the square of previous gradient will only grow larger, the learning rate with AdaGrad eventually falls to zero and network learning stops.

RMSProp
RMSProp is similar to AdaGrad, except that a decay parameter is applied to the cached sum of previous squared gradient values. This has the effect of preventing the learning rate from goin to zero, by constantly re-energizing the search.

ADAM
ADAM combines the concept of momentum with AdaGrad [13].

Stochastic gradient descent derives its name from the random fluctuations of the gradient vector of the scoring function with respect to weights across networks in a batch. The computed gradients are averaged across and the network scoring functions are minimized using the average gradient.

5.2 Regularization

Regularization is another mechanism for preventing networks from overfitting to the data. A regularization technique called L2 regularization places a penalty on the sum of weights squared, with a weight decay parameter, denoted λ.

$$E(W,b) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\|h_{W,b}(x^{(i)}) - y^{(i)}\|^2 + \frac{\lambda}{2}\sum_{l=1}^{n_{l-1}}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ji}^{(l)})^2 \tag{5}$$

This sort of regularization results in smaller, more distributed weights, which reduces overfitting. Another technique, called L1 regularization, places a penalty on the sum of the absolute values of the weights.

$$E(W,b) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\|h_{W,b}(x^{(i)}) - y^{(i)}\|^2 + \frac{\lambda}{2}\sum_{l=1}^{n_{l-1}}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left|W_{ji}^{(l)}\right| \tag{6}$$

L1 regularization promotes sparsity, and many weights end up being zero.

5.3 Network Weight Initialization

Careful consideration must be given to how a network's weights are initialized. This consideration must be made, while keeping in mind the type of non-linear activating function being used. If weights are initialized too large in magnitude, then a non-linear saturating function, such as tanh, or sigmoid could become saturated in either the positive or negative plateau regions. This is bad because during training by backpropagation it causes the gradient of the non-linear functions to go to a near-zero value, for those neurons which are in the saturation region. As more near-zero derivatives are chained on, the gradient becomes essentially zero with respect to any weight, and network training by backpropagation will not be possible.

One method for initializing network weights distributes values randomly, from a Gaussian distribution, but with the variance inversely proportional to the sum of network inputs and outputs [14]. A network with more input and output neurons would be initialized with random weights drawn from a narrower Gaussian distribution, centered on zero, compared to a network with fewer input and output neurons. This approach was proposed to prevent the backpropagation gradient from vanishing or exploding. However, in practice, the Caffe deep learning framework has implemented a slightly different expression for the variance, which is inversely proportional to only the number of network inputs[15]. Further research has discovered that, when using ReLU activation units, multiplying the inverse of the number of inputs by 2 yields a variance which gives the best results [16].

5.4 Dropout

Dropout was a technique developed by [17], for preventing overfitting of a network to the particular variations of the training set. In fully connected layers, which are prone to overfitting, dropout can be used, where nodes in a layer will be removed with some defined probability, p. Nodes which are removed do not participate in training. After training, they are replaced in the network with their original weights. This prevents the fully connected layers from overfitting to the training data set, and improves performance on the validation set and during deployment.

5.5 Image Data Augmentation

Data augmentation is an effective way to increase the size of a data set, while not degrading the quality of the data set. Training on the augmentation data set yields networks which perform better than those trained on an unaugmented data set. To augment an image data set, the original images can be flipped horizontally and vertically, and subsamples of the original images can be selected at random positions in the original. These subsampled images can then also be flipped horizontally and vertically [1].

5.6 Activation Functions

The relation between image class and image data is non-linear. In order for a neural network to construct the non-linear relation between the data and image class, the activation function must have a nonlinearity. Without non-linearities, a neural network would be capable of only linear classification.

Hyperbolic tangent and sigmoidal activations were used before it was discovered that ReLUs could mitigate the exploding/vanishing gradient problem, while also reducing training time. ReLUs require only two floating point operations to produce an output.

tanh

$$f(x) = \tanh(x) \tag{7}$$

sigmoid

$$f(x) = \frac{1}{1 + e^{-x}} \tag{8}$$

radial basis functions

$$f(x) = e^{-\frac{(x-x_i)^2}{2\alpha}} \tag{9}$$

ReLU (Rectified Linear Unit)

$$f(x) = max(0, x) \tag{10}$$

ReLUs continue to be widely used for their convenient properties. Using ReLU as an activation function can lead to "dead" neurons during training. These dead neurons can be reset, however.

leaky ReLU

$$f(x) = max(-0.1x, x) \tag{11}$$

Leaky ReLUs, instead of remaining at zero for all input values less than zero, outputs a negative value, scaling with the input, but at a slope substantially less than 1. The resulting piecewise defined function looks like the ReLU, but with the zero portion rotated only slightly counterclockwise into Quadrant III. ReLUs can sometimes go "dead" when they get stuck in the zero output plateau region. Leaky ReLUs, on the other hand, do not have this tendency to go "dead" during training.

6 Determining Compute Resources Required for a Convnet

An example is presented here on how to determine the number of parameters a convnet requires, and how to determine the number of FLOPs required for a forward or backward pass through the network. Consider the convnet depicted in Figure 7. The network depicted

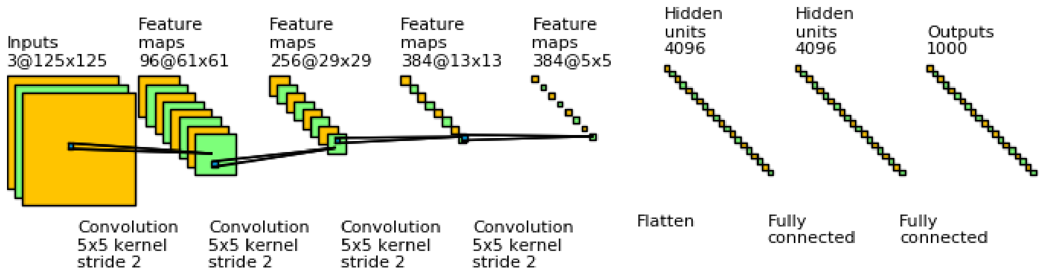

Figure 7: A convolutional neural network with max pool layers replaced by convolution layers having stride 2.

consists of an input layer, an output layer, and five hidden layers. Two hidden layers are patch connected to the previous layer. The last three hidden layers are fully connected to the previous layer. These are called fully connected, or affine layers.

If given a 125×125 input image, and sets of 96, 256, 384, and 384 filters of dimension 5×5 are to be convolved at the first, second, third, and fourth convolutional layers, with no zero padding, and with stride 2, then the activation maps will be reduced in dimension, as shown in the table below.

6.1 Parameter Counting

To begin counting the total number of parameters required to describe a network, consider the network in Figure 7, the first convolution is done with 96 filters. We will use the variable K to represent the number of filters.

$$Q_{\text{params}} = K(5 \times 5 \times 3 + 1) \tag{12}$$

The extra 3 arises from the three color channels for the input image.

Activation Layer Sizes after Convolution	
Convnet Layer	Activation Layer Size
input	125×125×3
Conv Layer1	61×61×96
Conv Layer2	29×29×256
Conv Layer3	13×13×384
Conv Layer4	5×5×384
FC1	1x1x4096
FC2	1x1x4096
Output	1x1x1000

Table 1: Because of the stride of 2, the activation layers become reduced approximately by a factor of 2 after every convolution. Max pooling is becoming replaced by convnets having stride > 1.

The neurons in each successive layer have a bias term adding to the sum over all 25 weighted inputs from the filter. Note that there is only one bias across all three color channels. This leads to 126 parameters per filter: 125 for the weights, and one for the bias term. In total, the quantity of parameters, Q_{params}, is then 12,096 (12,000 weights and 96 bias terms). The weight parameters then fully describe all 5×5×3+1 parameters for each of the 96 filters.

6.2 Memory Requirement

The memory requirement is determined by the memory for the feature map activations. The memory footprint for a network is twice the total given in Table 2, because the gradients must also be stored along the forward pass. As the network is computed forward, the derivatives can be calculated along the way, and both the neurons' activations and gradient of the output with respect to neuron weights can be stored in cache. The stored values will then be used when backpropagation is computed during training. Some deep learning frameworks will cache these values during the forward pass.

Doubling the memory value in Table 2 yields 9.22MB as the total memory footprint for one network. On a GPU with 8GB onboard, 867 networks could be fit onto the GPU at once. However, there is a limit to the batch size, beyond which there is no improvement in the averaged gradient. A batch size of 256 is common, if the network memory footprint will fit into the GPU memory 256 times.

Before we can begin estimating the number of parameters required, we must consider several parameters related to the filter itself. These are the *stride*, *filter size*, and *zero padding depth*.

Output Tensor Sizes of the Activation Map			
Convnet Layer	expression	number of parameters	memory required
Conv Layer1	61×61×96	357,216	1.43MB
Conv Layer2	29×29×256	215,296	0.86MB
Conv Layer3	13×13×384	64,896	0.26MB
Conv Layer4	5×5×384	9,600	0.04MB
FC1	1×1×4,096	4096	16.4KB
FC2	1×1×4,096	4096	16.4KB
Output	1×1×1000	1000	4KB
			4.61 MB in total

Table 2: Number of activations present in each layer, and the memory required for single precision activation values. Notice that network activations occupy much more memory toward the input. Multiply the memory requirement by two to also account for the memory needed to cache gradients during the forward pass, for later backpropagation.

6.3 Similar Network, with More Layers, and Stride 1

Suppose for a moment that there are 20 convolutional layers instead. The activation map size will be reduced substantially after 20 convolutions. However, having stride 1 means the activation map outputs will not decrease in size nearly as quickly as they did with in the case with stride 2. At each layer, the dimension is reduced by $F-1$, or 4. After 20 layers, the 125×125 input image will be reduced to 45×45 activation maps.[1]

If this network were made deeper, then after another ten convolutions, the output activation maps would be reduced to 5×5. One more convolution reduces it to 1×1. If the last convolution was instead done with a thousand different 5×5 filters, then the final output, after 31 convolutions, would be a 1×1×1000 tensor (see related footnote).

This would seem to offer an alternative to running several full connection layers near the output of the network. However, this may also diminish the effectiveness of leveraging *full distributed representation* in the fully connected layers. The activations in the final affine layers represent the presence of high-level features in the image. Combinations of the first abstract, high level, representation vector will be recombined by the second fully connected layer. This allows for multiple recombination of high level features which then fully connect to the classification layer. The classification layer has a number of neurons equal to the number of image classes. The output is then passed through a softmax function, so each element of the classification vector is a probability between zero and one.

As the dimension in activation layers is decreased, progressing toward the network output,

[1] the input image is actually a 125×125×3 *tensor*. The three extra dimensions arise from the RGB channels of the original input image. A 125×125×3 input tensor would be transformed to a 109×109×384 tensor volume after the 4th convolution. The extra ×384 here arises from the fact that 384 different filters are applied to the input tensor at the 4th layer.

Network Parameter Count			
Convnet Layer	expression	number of parameters	memory required
Conv Layer1	(5×5×3+1)×96	12,096	0.05MB
Conv Layer2	(5×5×96+1)×256	614,656	2.46MB
Conv Layer3	(5×5×256+1)×256	1,638,656	6.55MB
Conv Layer4	(5×5×384+1)×4096	39,325,696	157MB
FC1	4096×(4096+1)	16,781,312	67.1MB
FC2	4096×(4096+1)	16,781,312	67.1MB
Output	4096×(1000+1)	4,100,096	16.4MB
			321MB in total

Table 3: Number of parameters required for each network layer, and the memory required for single precision parameters. The memory requirements assume that each parameter is 4 bytes, or a single precision floating point number.

information loss could result, if the number of filters per layer is not increased at a minimum pace. With a deep network, it is possible to maintain layer size by padding layers before convolving them. An example of this is illustrated in Figure 4. When layer size is maintained, a side effect occurs, however, which is that pixels closer to the edge are lower in value, or they contain redundant information. This effect increases in deeper network layers. Even with padding, some information loss still occurs.

By increasing the number of filters as the effective area of activation layers decreases, the high dimensional, low-level image information is transformed to low dimensional, high-level information. The central challenge in neural networks is obtaining networks which perform this transition so that an effective, high-level, compressed information representation is produced at the output, *e.g.*, image class.

Newer convnet architectures are beginning to replace the fully connected layers in favor of more convolutional layers. With current trends, better performance is seen when average pooling over an output tensor of activation maps when each cross section is 9×9, or slightly larger. Average pooling here replaces full connections. This approach has been used with GoogLeNet, which performed better than previous architectures having fully connected layers near the output. This suggests that high level feature recombination in successive fully connected layers is excessive, and that better performance can be obtained with less recombination.

6.4 Computational Requirement

Computational requirements for fully connected and convnets are determined differently. The full connection in multilayer networks allows for the computation to be expressed as a single matrix multiplication for each layer.

Because of the use of filters, convnets do have have a method as straightforward as a single matrix computation for determining the number of floating point operations involved in a single forward pass. New developments have shown that the compute requirement for convnets can be reduced by using the Strassen algorithm, by a factor down to $\frac{1}{8}$ [18].

While the number of floating point operations can be determined, the speed at which the total number of operations can be executed depends on the programming language, algorithmic implementation in the application, and the hardware compute capacity, expressed in floating point operations per second (FLOPS). Host to device communication latencies also play a determining factor in program execution speed. Modern GPUs, such as the Tesla P100 have compute capacities of 10.3 TFLOPS for single precision floating point operations.

The use of sigmoidal or tanh activation functions are more expensive than ReLU. ReLU requires only two FLOPs per output in an activation layer. Sigmoidal and tanh functions require more, since they are represented by a truncated Taylor series expansion.

Adding a regularization term to the scoring function, or incorporating dropout into any layers would complicate determining the computational requirement. This topic will be revisited in another paper.

7 Newer Developments

7.1 Inception Modules

Inception is an architectural element developed to allow for some scale invariance in object recognition. If an object in an image is too small, a convnet may not be able to detect it because the network filters were trained to recognize larger versions of the same object. In other words, a network could be trained to recognize faces very well. But if the face is too far away in the image, the filters, trained on larger faces, will not pick out the scaled down, distant face. Inception is able to convolve through the image with various filter sizes, all at the same computational step in the network. The results are computed and combined together into a single vector.

Combining the scale invariance of Inception modules with Residual Networks might result in an architecture worth investigating [3, 2]. The best features of both GoogLeNet and ResNet could possibly be combined into a single network for better performance.

As mentioned previously, GoogLeNet does not use fully connected layers at the end of the network. Instead, it computes average pooling across a $7 \times 7 \times 512$ output volume, to yield a $1 \times 1 \times 512$ vector. This avoids a substantial portion of the parameters required for fully connected layers, and suggests that successive recombination of high level features in fully connected layers is excessive for constructing a low dimension, high level representation of the image.

7.2 Residual Networks

Microsoft ResNet was the winner of the ILSVRC competition in 2015. Their entry introduced a new architectural design, called residual networks. One of the advantages of residual networks is that backpropagation does not encounter the vanishing/exploding gradient problem. A network output is determined by the sum of a series of small residual changes made to the original input. Because the gradient can be backpropagated without attenuation with ResNets, network architectures can be deeper than anything previously possible. Microsoft introduced a network having 152 layers (8 times deeper than VGG Net) [2],[19].

Previously, increasing network depth past a certain point was shown to actually decrease network performance. With Residual networks, network performance continues to increase beyond depths at which regular convnets yielded decreased performance.

8 Conclusion

Several trends have been noted throughout this paper. They are summarized here for overview.

1. filters of 3×3 are being used more often, instead of larger filters, but with more network layers [19]
2. Fully connected layers are being replaced with average pooling over some output volume
3. With batch normalization, networks can do without dropout and have higher learning rates

Two major architectural improvements have been developed recently. The inception module, along with residual networks, have improved convnet performance, and introduced new capabilities. The Inception module provides some scale invariance, while residual networks allow for training deeper networks. Residual networks can train deep networks without the degradation of performance seen when regular convnets are extended to the number of layers used in Microsoft's ResNet.

Several methods for preventing overfitting of a network to training data have been developed, including:

- convolutional networks with local filters
- dropout
- data set augmentation
- L2 Regularization

Methods for more effective training discussed are:

ReLU
Using ReLU as an activation function speeds up training, and works around the vanishing/exploding gradient problem. ReLU can result in "dead" neurons during training. The Leaky ReLU was developed to address this problem.

Xavier Initialization
tunes the variance of initial weight distribution to be inversely proportional to the number of network inputs

ADAM, RMSProp
more recently developed minimization algorithms such as ADAM, and RMSProp result is faster training times.

Calculating the memory requirement for a convnet will help determine an appropriate batch size. The exercise illustrated demonstrates how to compute memory requirements for a convolutional network with four convolutional layers, and three fully connected layers (including the output layer).

9 Acknowledgements

The code used to generate the figures illustration discrete convolution is available on GitHub.[2]. The code used to generate the convolutional neural networks in Figure 2 and Figure 7 is also available on GitHub.[3].

[2]`https://github.com/vdumoulin/conv_arithmetic`
[3]`https://github.com/gwding/draw_convnet`

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[3] Christian Szegedy, Wei Liue, Yangqing Jia, Pierre Sermanet Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

[4] Warren McCulloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[5] David Hubel and Torsten Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160:106–154, 1962.

[6] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[7] Yann Le Cun, Léon Bottou, and Yoshua Bengio. Reading checks with multilayer graph transformer networks. In *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, volume 1, pages 151–154. IEEE, 1997.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition*. IEEE, 2009.

[9] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer vision–ECCV 2014*, pages 818–833. Springer, 2014.

[10] Hochreiter, Sepp, and Schmidhuber. Long short-term memory. *Neural Computation*, 9.8:1735–1780, 1997.

[11] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. *CoRR*, 2015.

[12] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, 1631, 2013.

[13] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] Glorot Xavier and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *Aistats*, 9, 2010.

[15] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv preprint arXiv:1502.01852*, 2015.

[17] Nitish Srivastava, Geoffrey Hinton, and Alex and Krizhevsky. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[18] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[19] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.