Representational Capacity

More hidden neurons $\Rightarrow$ capacity to represent more complex functions

- Problem: overfitting vs. generalization
  - We will discuss the different strategies to help here (L2 regularization, dropout, input noise, using a validation set etc.)

Figure: https://cs231n.github.io/
Number of hidden layers

• Depends on the nature of the problem
  – Linear classification? ➜ No hidden layers needed
  – Non-linear classification?
What do the layers represent?

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]
Model Complexity

- Models range in their flexibility to fit arbitrary data

- **Simple model**
  - Low bias
  - High variance
  - Small capacity may prevent it from representing all structure in data

- **Complex model**
  - High bias
  - Low variance
  - Large capacity may allow it to memorize data and fail to capture regularities
Training Vs. Test Set Error

Previously on CENG501!

Slide Credit: Michael Mozer
Avoiding Overfitting

- Increase training set size
  - Make sure effective size is growing; redundancy doesn’t help
- Incorporate domain-appropriate bias into model
  - Customize model to your problem
- Set hyperparameters of model
  - number of layers, number of hidden units per layer, connectivity, etc.
- Regularization techniques
Regularization

• L2 regularization: $\frac{1}{2} \lambda w^2$
  – Very common
  – Penalizes peaky weight vector, prefers diffuse weight vectors

• L1 regularization: $\lambda |w|$
  – Enforces sparsity (some weights become zero)
  – Why? Weight decay is by a constant value if $|w|$ is non-zero.
  – Leads to input selection (makes it noise robust)
  – Use it if you require sparsity / feature selection

• Can be combined: $\lambda_1 |w| + \lambda_2 w^2$

• Regularization is not performed on the bias; it seems to make no significant difference
Regularization

• Enforce an upper bound on weights:
  – Max norm:
    • $||w||_2 < c$
    • Helps the gradient explosion problem
    • Improvements reported

• Dropout:
  – At each iteration, drop a number of neurons in the network
  – Use a neuron’s activation with probability $p$ (a hyperparameter)
  – Adds stochasticity!

Fig: Srivastava et al., 2014

Regularization: **Inverted Dropout**

Perform scaling while dropping at training time!

**Training-time:** Correct the expected expected output from $px$ to $x$.

```python
# forward pass for example 3-layer neural network
H1 = np.maximum(0, np.dot(W1, X) + b1)
U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
H1 *= U1 # drop!
H2 = np.maximum(0, np.dot(W2, H1) + b2)
U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
H2 *= U2 # drop!
out = np.dot(W3, H2) + b3
```

**Test-time:**

```python
def predict(X):
    # ensembled forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    out = np.dot(W3, H2) + b3
```
“Dropout performs gradient descent on-line with respect to both the training examples and the ensemble of all possible subnetworks.”


Fig: Srivastava et al., 2014
Data Augmentation

Previously on CENG501!

Spring 2021

Sinan Kalkan

When to stop training: Early stopping

Previously on CENG501!

Spring 2021

Sinan Kalkan

Slide Credit: Michael Mozer
Data Preprocessing: Mean subtraction

- Compute the mean of each dimension, \( \mu_i \), over the training set:

\[
\mu_i = \frac{1}{N} \sum_j x_{ji}
\]

- Subtract the mean for each dimension:

\[
x'_{ji} \leftarrow x_{ji} - \mu_i
\]

- Effect: Move the data center (mean) to coordinate center

Mean image of CIFAR10
(from PA1)

Previously on CENG501!

Spring 2021

Sinan Kalkan

Data Preprocessing: Normalization (or conditioning)

- Necessary if you believe that your dimensions have different scales
  - Might need to reduce this to give equal importance to each dimension
- Normalize each dimension by its std. dev. after mean subtraction:
  \[ \begin{align*}
  x_{ji}' &= x_{ji} - \mu_i \\
  x_{ji}'' &= x_{ji}' / \sigma_i
  \end{align*} \]

- Effect: Make the dimensions have the same scale

Previously on CENG501!

Initial Weight Normalization

• Problem: Variance of the output changes with the number of inputs

• If $s = \sum_i w_i x_i$ (note that $\text{Var}(X) = E[(X - \mu)^2]$):

$$\text{Var}(s) = \text{Var}(\sum_i w_i x_i)$$

$$= \sum_i \text{Var}(w_i x_i)$$

$$= \sum_i [E(w_i)]^2 \text{Var}(x_i) + E[(x_i)]^2 \text{Var}(w_i) + \text{Var}(x_i) \text{Var}(w_i)$$

$$= \sum_i \text{Var}(x_i) \text{Var}(w_i)$$

$$= (n \text{Var}(w)) \text{Var}(x)$$

Eqn: https://cs231n.github.io/neural-networks-2/#init
Initial Weight Normalization

• Solution:
  – Get rid of $n$ in $\text{Var}(s) = (n \ \text{Var}(w))\text{Var}(x)$

• How?
  – Scale the initial weights by $\sqrt{n}$
  – Why? Because: $\text{Var}(aX) = a^2\text{Var}(X)$

• Standard Initialization (top plots in Figure 6 & 7):
  $w_i \sim U \left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$
  which yields $n \ \text{Var}(w) = \frac{1}{3}$
  because variance of $U[-r, r]$ is $\frac{r^2}{3}$.

Xavier initialization for symmetric activation functions (Glorot & Bengio):

$$w_i \sim N \left(0, \frac{\sqrt{2}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}\right)$$

With Uniform distribution:

$$w_i \sim U \left[-\frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}, \frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}\right]$$

Initial Weight Normalization

He et al. shows that Xavier initialization does not work well for ReLUs.

\[
\begin{align*}
\hat{w}_i &= \text{rand}(0,1) \times \frac{\sqrt{2}}{\sqrt{n}} \\
\end{align*}
\]

They suggest the following:

Figure 2. The convergence of a **22-layer** large model (B in Table 3). The x-axis is the number of training epochs. The y-axis is the top-1 error of 3,000 random val samples, evaluated on the center crop. We use ReLU as the activation for both cases. Both our initialization (red) and “Xavier” (blue) [7] lead to convergence, but ours starts reducing error earlier.

Figure 3. The convergence of a **30-layer** small model (see the main text). We use ReLU as the activation for both cases. Our initialization (red) is able to make it converge. But “Xavier” (blue) [7] completely stalls - we also verify that its gradients are all diminishing. It does not converge even given more epochs.
More on Weight Initialization

Tutorial and Demo:
https://www.deeplearning.ai/ai-notes/initialization/index.html

Tutorial:
Alternative: Batch Normalization

Normalization is differentiable

– So, make it part of the model (not only at the beginning)
– I.e., perform normalization during every step of processing

• More robust to initialization

• Shown to also regularize the network in some cases (dropping the need for dropout)

• Issue: How to normalize at test time?
  1. Store means and variances during training, or
  2. Calculate mean & variance over your test data

• PyTorch: use model.eval() in test time.

---

**Input:** Values of $x$ over a mini-batch: $B = \{x_1, \ldots, x_m\}$; Parameters to be learned: $\gamma, \beta$

**Output:** $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

\[
\begin{align*}
\mu_B &\leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i & \text{ // mini-batch mean} \\
\sigma_B^2 &\leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2 & \text{ // mini-batch variance} \\
\hat{x}_i &\leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} & \text{ // normalize} \\
y_i &\leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) & \text{ // scale and shift}
\end{align*}
\]

**Algorithm 1:** Batch Normalizing Transform, applied to activation $x$ over a mini-batch.

---

Alternative Normalizations

Batch Norm
Layer Norm
Instance Norm
Group Norm

https://medium.com/syncedreview/facebook-ai-proposes-group-normalization-alternative-to-batch-normalization-fb0699bfae7
Previously on CENG501!

Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift

Xiang Li1, Shuo Chen1, Xiaolin Hu2, Jian Yang1

2018

Since we get a clear knowledge about the disharmony between Dropout and BN, we can easily develop several approaches to combine them together, to see whether an extra improvement could be obtained. In this section, we introduce two possible solutions in modifying Dropout. One is to avoid the scaling on feature-map before every BN layer, by only applying Dropout after the last BN block. Another is to slightly modify the formula of Dropout and make it less sensitive to variance, which can alleviate the shift problem and stabilize the numerical behaviors.
What if things are not working?

Learning rate might be too low;
Batch size might be too small
What if things are not working?

Previously on CENG501!
Today

• Convolutional Neural Networks
  – Disadvantages of MLPs
  – Advantages of CNNs
  – CNN Operations: Convolution, pooling, nonlinearity, normalization, FC
  – CNN Architectures:

• Next week:
  – Backpropagation with a CNN
  – Transfer learning
  – Visualizing and understanding CNNs
  – Widely-used CNN architectures
  – CNN Applications

Administrative Issues

• Programming assignment 1

• Take-Home Exam 1:
  – Announced: 3\textsuperscript{rd} of May
  – Deadline: 9\textsuperscript{th} of May?

• Project paper selection
  – https://docs.google.com/spreadsheets/d/1tzPHq_Vgu6gCwNyXJHGvqeA6p
gU67H0nKYjqkisWfKc/edit?usp=sharing
  – Deadline: 19\textsuperscript{th} of April
CONVOLUTIONAL NEURAL NETWORKS: MOTIVATION
Disadvantages of MLPs: **Curse of Dimensionality**

- Number of required samples for obtaining small error increases exponentially with input dimensions
- Too many many parameters

Illustration of the curse of dimensionality; in order to approximate a Lipschitz-continuous function composed of Gaussian kernels placed in the quadrants of a $d$-dimensional unit hypercube (blue) with error $\epsilon$, one requires $\Theta(1/\epsilon^d)$ samples (red points).

Disadvantages of MLPs: **Equivariance**

- Vectorizing an image breaks patterns in consecutive pixels.
  - Shifting one pixel means a whole new vector
  - Makes learning more difficult
  - Requires more data to generalize
Equivariance vs. Invariance

• Equivariant problem: image segmentation.
  – $f(g(x)) = g(f(x))$

• Invariant problem: object recognition.
  – $f(g(x)) = f(x)$

• Pooling provides invariance, convolution provides equivariance.

https://www.mathworks.com/discovery/image-segmentation.html
An Alternative to MLPs

Solution (inspiration):

- Hubel & Wiesel: Brain neurons are not fully connected. They have local receptive fields.

![Graph showing neural response vs. stimulus orientation](http://fourier.eng.hmc.edu/e180/lectures/retina/node1.html)
An Alternative to MLPs

Solution: Neocognitron (Fukushima, 1979):

A neural network model unaffected by shift in position, applied to Japanese handwritten character recognition.

- S (simple) cells: local feature extraction.
- C (complex) cells: provide tolerance to deformation, e.g. shift.
- Self-organized learning method.

Figure: Fukushima (2019), Recent advances in the deep CNN neocognitron.
An Alternative to MLPs

Solution:

Neocognitron’s self-organized learning method (Fukushima, 2019):

“For training intermediate layers of the neocognitron, the learning rule called AiS (Add-if-Silent) is used. Under the AiS rule, a new cell is generated and added to the network if all postsynaptic cells are silent in spite of non-silent presynaptic cells. The generated cell learns the activity of the presynaptic cells in one-shot. Once a cell is generated, its input connections do not change any more. Thus the training process is very simple and does not require time-consuming repetitive calculation.”
An Alternative to MLPs

Solution: Convolutional Neural Networks (Lecun, 1998)

– Gradient descent
– Weights shared
– Document recognition
CNNs: Underlying Principle

CNNs vs. MLPs: Curse of Dimensionality

• A fully-connected network has too many parameters
  – On CIFAR-10:
    • Images have size 32x32x3 ➞ one neuron in hidden layer has 3072 weights!
  – With images of size 512x512x3 ➞ one neuron in hidden layer has 786,432 weights!
  – This explodes quickly if you increase the number of neurons & layers.

• Alternative: enforce local connectivity!

When things go deep, an output may depend on all or most of the input:

How Many Samples are Needed to Learn a Convolutional Neural Network?

Simon S. Du\textsuperscript{1}, Yining Wang\textsuperscript{1}, Xiyu Zhai\textsuperscript{2}, Sivaraman Balakrishnan\textsuperscript{3}, Ruslan Salakhutdinov\textsuperscript{1}, and Aarti Singh\textsuperscript{1}

\textsuperscript{1}Machine Learning Department, Carnegie Mellon University
\textsuperscript{2}University of Cambridge
\textsuperscript{3}Department of Statistics, Carnegie Mellon University

May 22, 2018

Abstract

A widespread folklore for explaining the success of convolutional neural network (CNN) is that CNN is a more compact representation than the fully connected neural network (FNN) and thus requires fewer samples for learning. We initiate the study of rigorously characterizing the sample complexity of learning convolutional neural networks. We show that for learning an \( m \)-dimensional convolutional filter with linear activation acting on a \( d \)-dimensional input, the sample complexity of achieving population prediction error of \( \epsilon \) is \( \tilde{O}(m/\epsilon^2) \), whereas its FNN counterpart needs at least \( \Omega(d/\epsilon^2) \) samples. Since \( m \ll d \), this result demonstrates the advantage of using CNN. We further consider the sample complexity of learning a one-hidden-layer CNN with linear activation where both the \( m \)-dimensional convolutional filter and the \( r \)-dimensional output weights are unknown. For this model, we show the sample complexity is \( \tilde{O}\left((m + r)/\epsilon^2\right) \) when the ratio between the stride size and the filter size is a constant. For both models, we also present lower bounds showing our sample complexities are tight up to logarithmic factors. Our main tools for deriving these results are localized empirical process and a new lemma characterizing the convolutional structure. We believe these tools may inspire further developments in understanding CNN.
**CNNs vs. MLPs: Curse of Dimensionality**

- Parameter sharing
  - In regular ANN, each weight is independent
- In CNN, a layer might re-apply the same convolution and therefore, share the parameters of a convolution
  - Reduces storage and learning time

For a neuron in the next layer:
- With ANN: 320x280x320x280 multiplications
- With CNN: 320x280x3x3 multiplications

CNNs vs. MLPs: Equivariance

• Equivariant to translation
  – The output will be the same, just translated, since the weights are shared.

![Diagram showing equivariance](https://towardsdatascience.com/translational-invariance-vs-translational-equivariance-f9fbc8fca63a)

• Not equivariant to scale or rotation.
A CRASH COURSE ON CONVOLUTION
Formulating Signals in Terms of Impulse Signal

\[ x[0] \delta[n] + x[1] \delta[n-1] + x[-1] \delta[n+1] + x[2] \delta[n-2] + \ldots \]

Alan V. Oppenheim and Alan S. Willsky
Formulating Signals in Terms of Impulse Signal

\[ x[n] = \cdots + x[-2] \delta[n + 2] + x[-1] \delta[n + 1] + x[0] \delta[n] + x[1] \delta[n - 1] + \cdots \]

\[ x[n] = \sum_{k=-\infty}^{+\infty} x[k] \delta[n - k] \]

Important to note the “-” sign

Coefficients

Basic Signals

Alan V. Oppenheim and Alan S. Willsky
Unit Sample Response

- Now suppose the system is LTI, and define the unit sample response $h[n]$:

$$
\delta[n] \rightarrow h[n] \\
\downarrow \\
\text{From Time-Invariance:}
$$

$$
\delta[n-k] \rightarrow h[n-k]
$$

From Linearity:

$$
x[n] = \sum_{k=-\infty}^{\infty} x[k] \delta[n-k] \rightarrow y[n] = \sum_{k=-\infty}^{\infty} x[k] h[n-k] = x[n] \ast h[n] \\
\text{convolution sum}
$$
Conclusion

The output of any DT LTI System is a convolution of the input signal with the unit-sample response, i.e.

\[ y[n] = x[n] * h[n] \]

\[ = \sum_{k=-\infty}^{+\infty} x[k] h[n-k] \]

As a result, any DT LTI Systems are completely characterized by its unit sample response.
Power of convolution

• Describe a “system” (or operation) with a very simple function (impulse response).
• Determine the output by convolving the input with the impulse response
Convolution

• Definition of continuous-time convolution

\[ x(t) \ast h(t) = \int x(\tau)h(t - \tau) \, d\tau \]
Convolution

- Definition of discrete-time convolution

\[ x[n] \ast h[n] = \sum x[k]h[n - k] \]

Choose the value of \( n \) and consider it fixed

\[ y[n] = \sum_{k=-\infty}^{+\infty} x[k]h[n - k] \]

View as functions of \( k \) with \( n \) fixed
Discrete-time 2D Convolution

• For images, we need two-dimensional convolution:

\[ s[i, j] = (I \ast K)[i, j] = \sum_m \sum_n I[m, n] K[i - m, j - n] \]

• These multi-dimensional arrays are called tensors

• We have commutative property:

\[ s[i, j] = (I \ast K)[i, j] = \sum_m \sum_n I[i - m, j - n] K[m, n] \]

• Instead of subtraction, we can also write (easy to drive by a change of variables). This is called cross-correlation:

\[ s[i, j] = (I \ast K)[i, j] = \sum_m \sum_n I[i + m, j + n] K[m, n] \]
Example multi-dimensional convolution


https://github.com/vdumoulin/conv_arithmetic
What can filters do?

Rectangular filter

\[ \text{g}[m,n] \ast \text{h}[m,n] = \text{f}[m,n] \]

Slide: A. Torralba
What can filters do?

Rectangular filter

\[ g[m,n] \ast h[m,n] = f[m,n] \]

Slide: A. Torralba
What can filters do?

Rectangular filter

\[
g[m,n] \ast h[m,n] = f[m,n]
\]
What can filters do?

Sharpening filter

- Filter coefficient: -0.35
- Sharpened result:
  - Differences are accentuated;
  - Constant areas are left untouched.

Slide: A. Torralba
What can filters do?
Sharpening filter

before
after

Slide: A. Torralba
What can filters do? Gaussian filter

\[ G(x, y; \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2 + y^2}{2\sigma^2}} \]
Global to Local Analysis

Dali

Slide: A. Torralba
What can filters do?

\([-1 \ 1]\)

\[\begin{array}{c}
g[m,n] \\
\otimes \\
[-1, 1] \\
h[m,n] \\
= \\
f[m,n]
\end{array}\]

Slide: A. Torralba
What can filters do?

\[-1 \ 1\]^T

\[
g[m,n] \ast [-1 \ 1]^T = h[m,n] = f[m,n]
\]
OVERVIEW OF CNN
CNN layers

- Stages of CNN:
  - Convolution (in parallel) to produce pre-synaptic activations
  - Detector: Non-linear function
  - Pooling: A summary of a neighborhood

- Pooling of a rectangular region:
  - Max
  - Average
  - L2 norm
  - Weighted average acc. to the distance to the center
  - …

An example architecture
Regular ANN

http://cs231n.github.io/convolutional-networks/

CNN
OPERATIONS IN A CNN: CONVOLUTION
Convolution in CNN

- The weights correspond to the kernel
- The weights are shared in a channel (depth slice)
- We are effectively learning filters that respond to some part/entities/visual-cues etc.

Local connectivity in CNN
= Receptive fields

• Each neuron is connected to only a local neighborhood, i.e., receptive field
• The size of the receptive field ➔ another hyper-parameter.
Connectivity in CNN

- Local: The behavior of a neuron does not change other than being restricted to a subspace of the input.
- Each neuron is connected to slice of the previous layer
- A layer is actually a volume having a certain width x height and depth (or channel)
- A neuron is connected to a subspace of width x height but to all channels (depth)
- Example: CIFAR-10
  - Input: 32 x 32 x 3 (3 for RGB channels)
  - A neuron in the next layer with receptive field size 5x5 has input from a volume of 5x5x3.
Important parameters

- **Depth (number of channels)**
  - We will have more neurons getting input from the same receptive field
  - This is similar to the hidden neurons with connections to the same input
  - These neurons learn to become selective to the presence of different signals in the same receptive field

- **Stride**
  - The amount of space between neighboring receptive fields
  - If it is small, RFs overlap more
  - If it is big, RFs overlap less

- **How to handle the boundaries?**
  i. Option 1: Don’t process the boundaries. Only process pixels on which convolution window can be placed fully.
  ii. Option 2: Zero-pad the input so that convolution can be performed at the boundary pixels.

[Diagram showing convolution weights]

Weights:

-2 2 1 2 1
-2 1 1
0 1 2 -1 1 -3 0
0 1 2 -1 1 -3 0

http://cs231n.github.io/convolutional-networks/
Padding illustration

- Only convolution layers are shown.
- Top: no padding $\Rightarrow$ layers shrink in size.
- Bottom: zero padding $\Rightarrow$ layers keep their size fixed.

Size of the next layer

- Along a dimension:
  - \( W \): Size of the input
  - \( F \): Size of the receptive field
  - \( S \): Stride
  - \( P \): Amount of zero-padding

- Then: the number of neurons as the output of a convolution layer:

\[
\frac{W - F + 2P}{S} + 1
\]

- If this number is not an integer, your strides are incorrect and your neurons cannot tile nicely to cover the input volume

Weights:

```
1 0 -1
```

Zero padding
Size of the next layer

• Arranging these hyperparameters can be problematic
• Example:
• If W=10, P=0, and F=3, then

\[
\frac{W - F + 2P}{S} + 1 = \frac{10 - 3 + 0}{S} + 1 = \frac{7}{S} + 1
\]

i.e., \( S \) cannot be an integer other than 1 or 7.

• Zero-padding is your friend here.
Real example – AlexNet (Krizhevsky et al., 2012)

• Image size: 227×227×3

• $W=227$, $F=11$, $S=4$, $P=0$ \[ \frac{227-11}{4} + 1 = 55 \]
  
  (55 => the width of the convolution layer)

• Convolution layer: 55×55×96 neurons
  
  (96: the depth, the number of channels)

• Therefore, the first layer has 55×55×96 = 290,400 neurons
  
  – Each has 11×11×3 receptive field \[ \Rightarrow \] 363 weights and 1 bias
  
  – Then, 290,400×364 = 105,705,600 parameters just for the first convolution layer (if there were no weight sharing)
  
  – With weight sharing: 96 x 364 = 34,944
However, we can share the parameters
  – For each channel (slice of depth), have the same set of weights
  – If 96 channels, this means 96 different set of weights
  – Then, 96×364 = 34,944 parameters
  – 364 weights shared by 55×55 neurons in each channel

Real example –
AlexNet (Krizhevsky et al., 2012)

Example filters learned by Krizhevsky et al. Each of the 96 filters shown here is of size [11x11x3], and each one is shared by the 55x55 neurons in one depth slice. Notice that the parameter sharing assumption is relatively reasonable: If detecting a horizontal edge is important at some location in the image, it should intuitively be useful at some other location as well due to the translationally-invariant structure of images. There is therefore no need to relearn to detect a horizontal edge at every one of the 55x55 distinct locations in the Conv layer output volume.

http://cs231n.github.io/convolutional-networks/
More on connectivity

Small RF & Stacking
• E.g., 3 CONV layers of 3x3 RFs
• Pros:
  – Same extent for these example figures
  – With non-linearity added with 2\textsuperscript{nd} and 3\textsuperscript{rd} layers \( \Rightarrow \) More expressive! More representational capacity!
  – Less parameters: \( 3 \times [(3 \times 3 \times C) \times C] = 27CxC \)
• Cons?

Large RF & Single Layer
• 7x7 RFs of single CONV layer
• Cons:
  – Linear operation in one layer
  – More parameters: \( (7 \times 7 \times C) \times C = 49CxC \)
• Pros?

So, we prefer a stack of small filter sizes against big ones
Implementation Details: NumPy example

• Suppose input is volume is $X$ of shape $(11, 11, 4)$
• Depth slice at depth $d$ (i.e., channel $d$): $X[:, :, d]$
• Depth column at position $(x, y)$: $X[x, y, :]$
• $F: 5$, $P: 0$ (no padding), $S=2$
  – Output volume ($V$) width, height = $(11-5+0)/2+1 = 4$
• Example computation for some neurons in first channel:

  \[
  V[0, 0, 0] = \text{np}.\text{sum}(X[:, 5, 5, :] \ast W_0) + b_0 \\
  V[1, 0, 0] = \text{np}.\text{sum}(X[2:7, 5, :] \ast W_0) + b_0 \\
  V[2, 0, 0] = \text{np}.\text{sum}(X[4:9, 5, :] \ast W_0) + b_0 \\
  V[3, 0, 0] = \text{np}.\text{sum}(X[6:11, 5, :] \ast W_0) + b_0
  \]

• Note that this is just along one dimension

http://cs231n.github.io/convolutional-networks/
Implementation Details: NumPy example

- A second activation map (channel):

\[
\begin{align*}
V[0,0,1] &= \text{np.sum}(X[:,5,5,:]) * W1 + b1 \\
V[1,0,1] &= \text{np.sum}(X[2:7,:5,:]) * W1 + b1 \\
V[2,0,1] &= \text{np.sum}(X[4:9,:5,:]) * W1 + b1 \\
V[3,0,1] &= \text{np.sum}(X[6:11,:5,:]) * W1 + b1 \\
V[0,1,1] &= \text{np.sum}(X[5,2:7,:]) * W1 + b1 \quad \text{(example of going along y)} \\
V[2,3,1] &= \text{np.sum}(X[4:9,6:11,:]) * W1 + b1 \quad \text{(or along both)}
\end{align*}
\]
Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters $K$
  - their spatial extent $F$
  - the stride $S$
  - the amount of zero padding $P$
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and $K$ biases.
- In the output volume, the $d$-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the $d$-th filter over the input volume with a stride of $S$, and then offset by $d$-th bias.