

Linear Algebra

Arthur J. Redfern

axr180074@utdallas.edu

Aug 22, 2018

Aug 27, 2018

Outline

- Motivation
- Vector spaces
- Linear feature extraction and prediction
- Linear pre processing
- References

Motivation

Pre Processing

- Pre processing methods simplify feature extraction and prediction
- Understanding linear transformations is a key to understanding many popular pre processing methods
- Example pre processing methods
 - Discrete Fourier transform
 - Principal component analysis

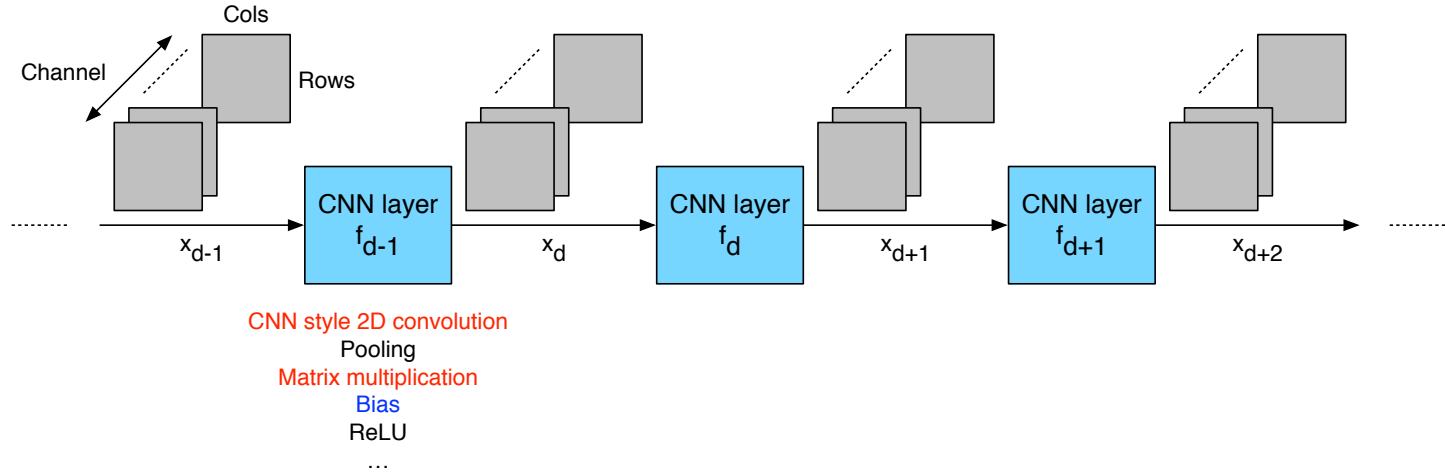
Feature Extraction And Prediction

- CNNs are compositions of nonlinear functions (layers)

$$\mathbf{y} = f_{D-1}(\dots(f_2(f_1(f_0(\mathbf{x}, h_0), h_1), h_2), \dots), h_{D-1}))$$

- Layers transform from data space to feature space to information space
- Examples layers
 - Fully connected layers with single and multiple inputs
 - CNN style 2D convolution layers

Feature Extraction And Prediction



- A key part of these 2 layers are linear transformations
- Understanding linear transformations is a key to the design and implementation of xNNs

Vector Spaces

Preliminaries

- Notation
 - Scalars are not bold
 - Vectors are bold lower case
 - Matrices are bold upper case
 - Indices start at 0 and go from 0, ..., size - 1

Set

- A collection of distinct objects

Field

- A set with well defined addition and multiplication operations
 - Associativity: $a + (b + c) = (a + b) + c$ and $a (b c) = (a b) c$
 - Commutativity: $a + b = b + a$ and $a b = b a$
 - Additive identity: $a + 0 = a$
 - Additive inverse: $a + (-a) = 0$
 - Multiplicative identity: $1 a = a$
 - Multiplicative inverse: $a a^{-1} = 1$
 - Distributivity: $a (b + c) = (a b) + (a c)$
- Elements of fields are generally referred to as scalars
- Examples: \mathbb{R} (real scalars), \mathbb{C} (complex scalars)

Vector

To do: add
an example
of a vector

- K tuple of scalars, always columns
- \mathbb{F}^K
- Examples: \mathbb{R}^K and \mathbb{C}^K

Matrix

To do: add
an example
of a matrix

- $M \times K$ tuple of scalars
- Collection of K vectors of size $M \times 1$ arranged in columns
 - Leads to column space and right null space
 - What can matrix vector multiplication reach and what can it not
 - Visualize using outer product of matrix vector multiplication
- Collection of M vectors of size $K \times 1$ transposed and arranged as rows
 - Leads to row space and left null space
 - What can vector matrix multiplication reach and what can it not
 - Visualize using outer product of vector matrix multiplication

Tensor

- $K_0 \times \dots \times K_{D-1}$ array of scalars
- Ordering
 - Last dimension is contiguous in memory
 - Working from right to left goes from closest to farthest spacing in memory
 - Feature maps: batch x channel x row x column
 - Filter coefficients: output channel x input channel x row x col

Function

- Mapping $f: X \rightarrow Y$ from domain to co domain
 - Injective: one to one; each y produced by at most one x
 - Surjective: onto; each y produced by at least one x
 - Bijective: one to one and onto
 - Bijective functions are invertible
- An infinite set is
 - Countably infinite if there's a bijection between the natural numbers and elements of the set
 - Uncountably infinite if there's not

Vector Space

- Set of vectors and linear combinations of those vectors

- Satisfy

- Associativity: $\mathbf{x} + (\mathbf{y} + \mathbf{z}) = (\mathbf{x} + \mathbf{y}) + \mathbf{z}$
- Commutativity: $\mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x}$
- Additive identity: $\mathbf{x} + \mathbf{0} = \mathbf{0}$
- Additive inverse: $\mathbf{x} + (-\mathbf{x}) = \mathbf{0}$
- Multiplicative compatibility: $a (b \mathbf{x}) = b (a \mathbf{x})$
- Multiplicative identity: $1 \mathbf{x} = \mathbf{x}$
- Distributivity: $(a + b)(\mathbf{x} + \mathbf{y}) = a \mathbf{x} + a \mathbf{y} + b \mathbf{x} + b \mathbf{y}$

- Examples: \mathbb{R}^K , \mathbb{C}^K , $\mathbb{R}^{K_0 \times \dots \times K_{D-1}}$

Normed Vector Space

- A vector space with a notion of distance
- A norm maps an element of the vector space to a scalar
- Satisfies
 - Non negativity: $||\mathbf{x}|| \geq 0$ and $||\mathbf{x}|| = 0$ iff $\mathbf{x} = 0$
 - Absolute scalability: $||a \mathbf{x}|| = |a| ||\mathbf{x}||$
 - Triangle inequality: $||\mathbf{x} + \mathbf{y}|| \leq ||\mathbf{x}|| + ||\mathbf{y}||$
- Example: l_p norm (common $p = 1, 2$ and ∞)

$$||\mathbf{x}||_p = (\sum_n (|x(n)|^p))^{1/p}, p \geq 1$$

Inner Product Space

- A vector space with a notion of distance and angle
- An inner product maps 2 elements of a vector space to a scalar
- Satisfies
 - Positive definiteness: $\langle \mathbf{x}, \mathbf{x} \rangle \geq 0$ and $\langle \mathbf{x}, \mathbf{x} \rangle = 0$ iff $\mathbf{x} = \mathbf{0}$
 - Conjugate symmetry: $\langle \mathbf{x}, \mathbf{y} \rangle = \text{conj}(\langle \mathbf{y}, \mathbf{x} \rangle)$
 - Linearity: $\langle a \mathbf{x}, \mathbf{y} \rangle = a \langle \mathbf{x}, \mathbf{y} \rangle$ and $\langle \mathbf{x} + \mathbf{y}, \mathbf{z} \rangle = \langle \mathbf{x}, \mathbf{z} \rangle + \langle \mathbf{y}, \mathbf{z} \rangle$
- Inner products induce norms on a vector space
 - But not all norms have associated inner products (e.g., l_∞)
- Example: dot product

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x} \bullet \mathbf{y} = \mathbf{x}^H \mathbf{y} = \sum_n (\text{conj}(x(n)) y(n)) = ||\mathbf{x}||_2 ||\mathbf{y}||_2 \cos(\theta)$$

Linear Feature Extraction And Prediction

Specifically

- Linear transformations used within fully connected and CNN style 2D convolution layers for feature extraction and prediction
 - Matrix vector multiplication
 - Matrix matrix multiplication
 - CNN style 2D convolution

Matrix Vector Multiplication

- Notation: M (output dimension), K (input dimension)
 - Setting up for BLAS notation

$$\begin{bmatrix} y(0) \\ \vdots \\ y(M-1) \end{bmatrix} = \begin{bmatrix} H(0,0) & \cdots & H(0,K-1) \\ \vdots & & \vdots \\ H(M-1,0) & \cdots & H(M-1,K-1) \end{bmatrix} \begin{bmatrix} x(0) \\ \vdots \\ x(K-1) \end{bmatrix}$$

- Mechanics
 - Inner product of matrix row and vector input to produce each output

Matrix Vector Multiplication

- A traditional neural network is composed of repeated fully connected layers
 - Output vector of features = pointwise nonlinearity (feature extraction matrix * input vector of data + bias vector)
 - Repeat with output of current layer as input to next layer

$$\mathbf{y} = \text{ReLU}(\mathbf{H} \mathbf{x} + \mathbf{b})$$

Matrix Vector Multiplication

- Matrix vector multiplication is a linear transformation
 - Every linear map can be represented as a matrix
 - Every matrix represents a linear map
 - So matrix vector multiplication in a neural network is doing linear transformations
- Multiple linear transformations can be composed into a single linear transformation
 - A reason why nonlinearities are included in xNNs
 - Otherwise there would be no depth

$$\mathbf{y} = \mathbf{H}_{D-1} \dots \mathbf{H}_1 \mathbf{H}_0 \mathbf{x} = \mathbf{H} \mathbf{x}$$

Matrix Vector Multiplication

- You design a network to accomplish a goal
 - Don't ever lose sight of this
 - Network design is not arbitrary
 - So it always makes sense to stop and think how the operations you're including help you accomplish that goal
 - Why do neural networks include these layers?
 - How do these layers map from data to features to predictions?
 - Pay attention to how the input features are combined to generate output features
- The purpose of the next few slides is to help build intuition on how matrix vector multiplication help layers map from data to features to classes

Matrix Vector Multiplication

- Intuition of feature extraction and prediction for a fully connected layer
 - Note: sometimes linear classification is viewed as template matching where each row is a different template and the predicted class is the maximum output
- Inner product depends on magnitude and angle
 - $y(m) = \mathbf{H}(m, :) \mathbf{x}$
 - $y(m)$ is the extracted feature or prediction
 - $\mathbf{H}(m, :)$ is the feature extractor or predictor
 - \mathbf{x} is the input

Matrix Vector Multiplication

- How strong or important is a feature extractor? $||\mathbf{H}(m, :)||_2$
 - Note that the input mag contributes the same to each extracted feature $||\mathbf{x}||_2$
 - So here input magnitude only matters relative to bias
 - But input magnitude will also matter for network structures with branches that come together
 - Input magnitude will also matter when the same feature extractor is applied to different inputs
- How aligned is the feature extractor with the input? θ
 - In same direction: positive feature
 - Orthogonal: 0 feature
 - In opposite direction: negative feature

Matrix Vector Multiplication

- Intuition of bias
 - Affine transformation
 - Allows the dividing line to shift
 - Implementation of rank 1 outer product
 - Will use bias in a constructive variant of the universal approximation proof
- Intuition of ReLU
 - Removes negatively aligned features or predictions
 - Allows depth
 - Subsequent layers combine positively aligned extracted features

Matrix Vector Multiplication

- Intuition of size of K and M
 - Small K to large M
 - Different combinations of a small number of features to predict a large number of classes
 - Large K to small M
 - 1 feature or a combination of features to predict a small number of classes is now possible
 - Example: ImageNet classification and final fully connected layer size

Matrix Vector Multiplication

- In general, for the final classification layer, it's better to have $K > M$
 - Classification goal is to create matrix and bias that takes K features and makes the correct 1 of the M elements at the output much larger (closer to $+\infty$) than all the others
 - To get a feel for this consider 2 extreme cases
 - 2 features K linearly combined + bias then ReLU to predict 200 classes M
 - 200 features K linearly combined + bias then ReLU to predict 2 classes M
 - Create example features, matrices and biases for both cases
 - Look at sensitivity of the prediction to errors in the features for each
 - Can relate to matrix condition number
 - Show the condition number of $K > M$ is always less than that of $K < M$
 - Now vary K and M and show diminishing returns after some point too
 - Generalize to considerations in a hierarchical head design
 - Start of a project idea

Matrix Vector Multiplication

- Arithmetic intensity

- Compute $= MK$ (MACs = multiply accumulates)
- Data movement $= K + MK + M$ (elements)
- Ratio $= (MK)/(K + MK + M)$ (consider M and K large)
 ≈ 1 (memory wall)

- If you want to make matrix vector multiplication run fast, you need to build a fast memory subsystem
 - Typically not an efficient thing to do from an operation per power perspective

Matrix Matrix Multiplication

- Notation: M (output dimension), K (input dimension), N (number of inputs and outputs)
 - BLAS M, N, K notation for $\mathbf{Y} = \mathbf{H} \mathbf{X}$
 - Will try and conform to this throughout
 - Matrix vector multiplication is a special case with $N = 1$

$$\begin{bmatrix} Y(0,0) & \cdots & Y(0,N-1) \\ \vdots & & \vdots \\ Y(M-1,0) & \cdots & Y(M-1,N-1) \end{bmatrix} = \begin{bmatrix} H(0,0) & \cdots & H(0,K-1) \\ \vdots & & \vdots \\ H(M-1,0) & \cdots & H(M-1,K-1) \end{bmatrix} \begin{bmatrix} X(0,0) & \cdots & X(0,N-1) \\ \vdots & & \vdots \\ X(K-1,0) & \cdots & X(K-1,N-1) \end{bmatrix}$$

Matrix Matrix Multiplication

- Application of same matrix transformation to multiple input vectors
 - Stack all the inputs next to each other
 - Get matrix matrix multiplication
- Lots of matrix operations and decompositions
 - Transpose is highlighted here because it will come up later
 - Transpose swaps matrix element indices
 - When applied to products of matrices remember socks then shoes, shoes then socks (or just remember the formula)

$$\mathbf{C}^T = (\mathbf{A} \mathbf{B})^T = \mathbf{B}^T \mathbf{A}^T$$

Matrix Matrix Multiplication

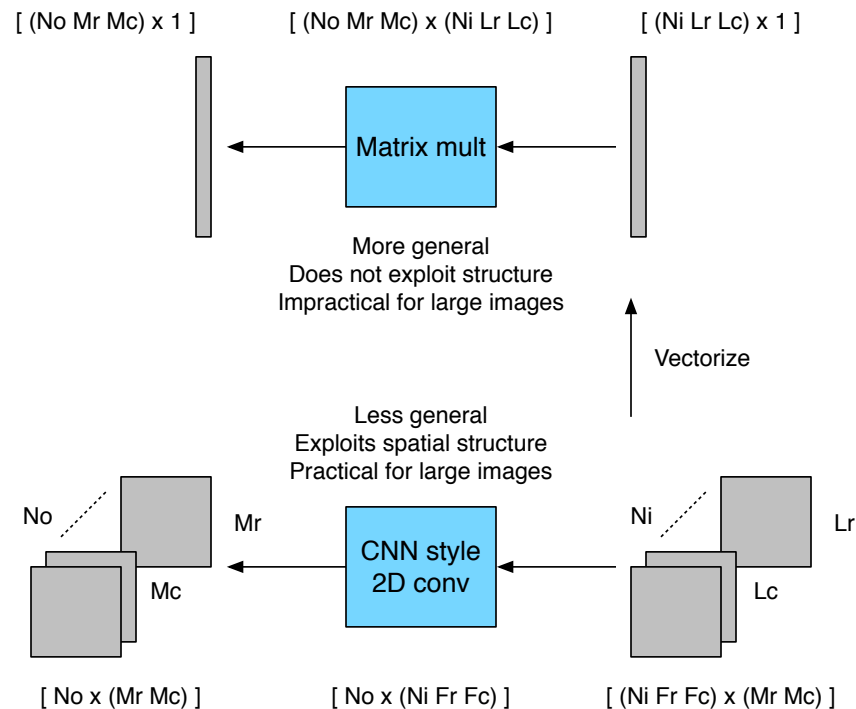
- Arithmetic intensity

• Compute	$= MNK$	(MACs)
• Data movement	$= KN + MK + MN$	(elements)
• Ratio	$= (MNK)/(KN + MK + MN)$	(cube in num, squares in den)
	$= N^3/(3 \cdot N^2)$	(special case $M = N = K$)
	$= N/3$	(ratio maxed with sq matrix)

- If you want to make matrix matrix mult run fast, if it's possible choose a large matrix size such that you get multiple ops per element of data moved
- Why are bubbles spherical? Min surface area per volume enclosed
 - Think of surface area as data movement and volume as MACs

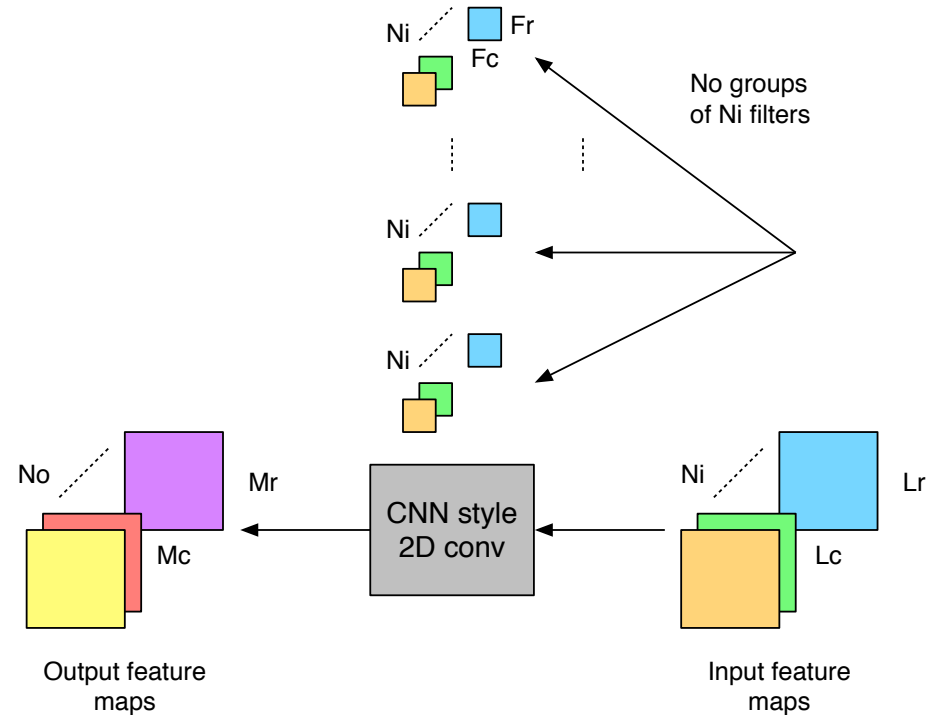
CNN Style 2D Convolution

- Consider applying a standard neural network to an image
 - Dimensions / memory / arithmetic intensity make it unreasonable to apply normal neural network linear layer to large images
- Use CNN style 2D convolution layer instead
 - It's a less general transformations but if the input / problem has translational invariance then perhaps the loss of generality is ok (and for many applications it is)
 - Very high (but not unreasonable) memory and compute for modern hardware



CNN Style 2D Convolution

- Input feature maps
 - 3D tensor
 - N_i inputs \times L_r rows \times L_c cols
- Filter coefficients
 - 4D tensor
 - N_o outputs \times N_i inputs \times F_r rows \times F_c cols
- Output feature maps
 - 3D tensor
 - N_o outputs \times M_r rows \times M_c cols



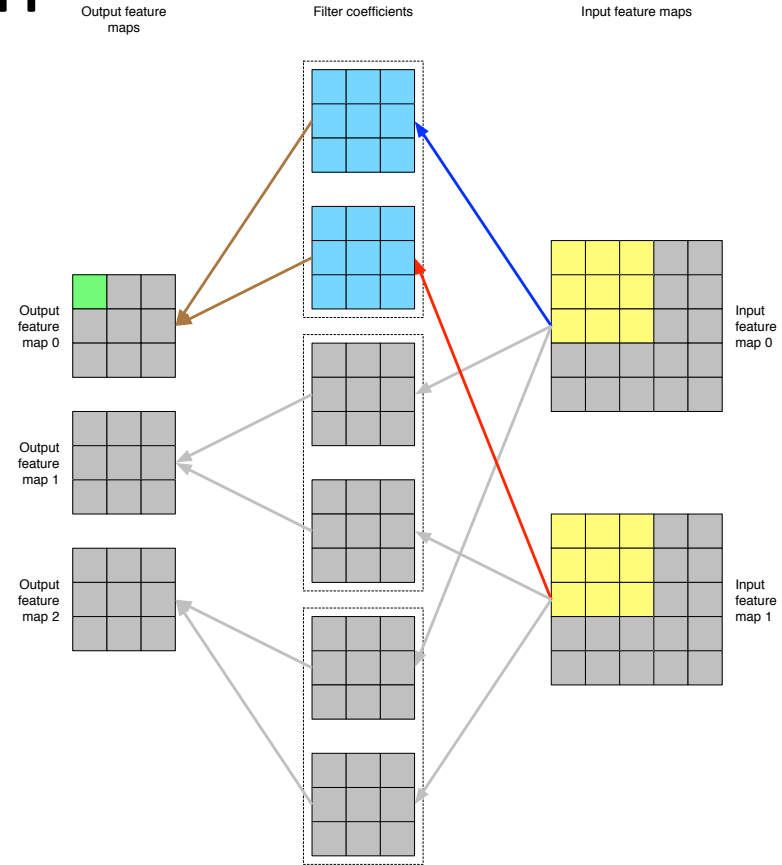
CNN Style 2D Convolution

- A brief comment on a complete CNN style 2D convolution layer before returning to the CNN style 2D convolution operation at it's core
- A traditional CNN is composed of multiple CNN style 2D convolution layers (plus some other layers that will be discussed layer)
 - \otimes is used here to notate CNN style 2D convolution
 - The notation for the bias is meant to indicate that the same bias value $V(no)$ is added to every $M_r \times M_c$ element of output feature map no
 - ReLU is applied pointwise

$$\mathbf{Y}^{No \times M_r \times M_c} = \text{ReLU}(\mathbf{H}^{No \times N_i \times F_r \times F_c} \otimes \mathbf{X}^{N_i \times L_r \times L_c} + \mathbf{V}^{No})$$

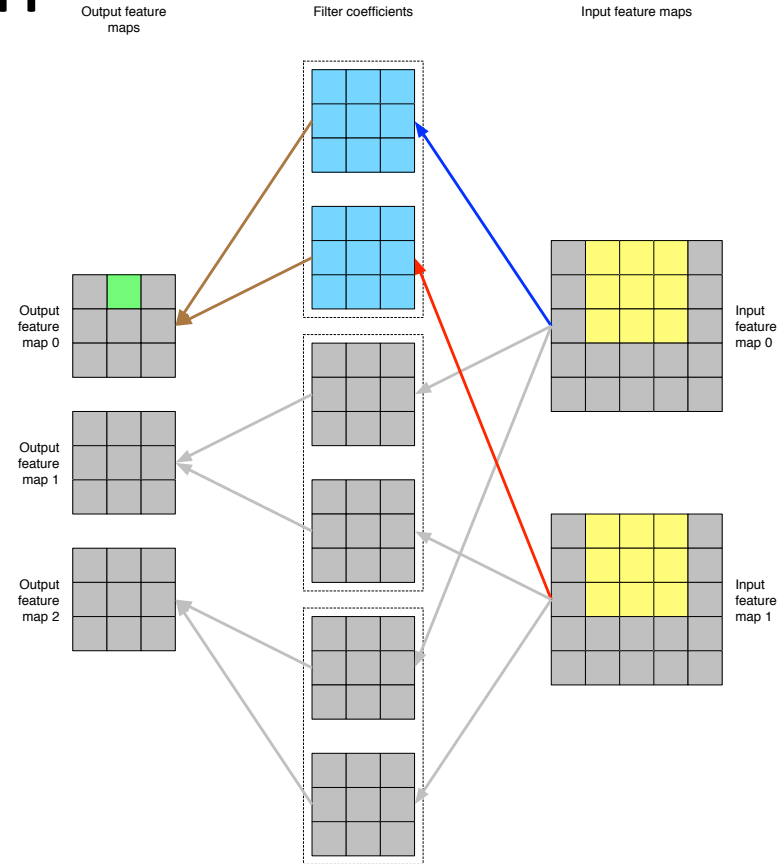
CNN Style 2D Convolution

- An example showing CNN style 2D convolution is matrix matrix multiplication using $2 \times 5 \times 5$ input feature maps, $3 \times 2 \times 3 \times 3$ filters and $3 \times 3 \times 3$ output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



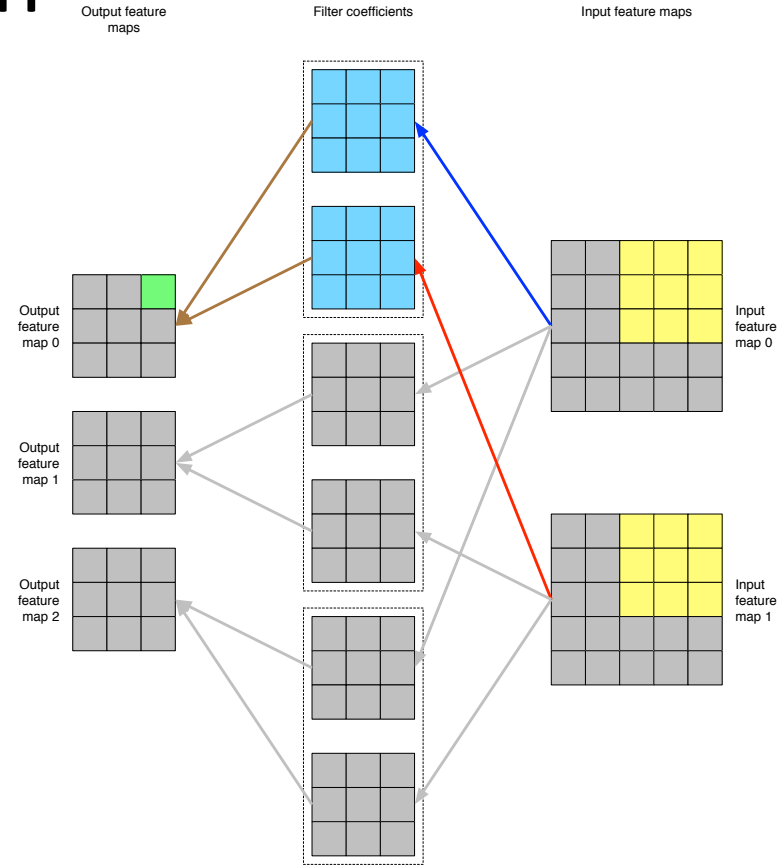
CNN Style 2D Convolution

- An example showing CNN style 2D convolution is matrix matrix multiplication using $2 \times 5 \times 5$ input feature maps, $3 \times 2 \times 3 \times 3$ filters and $3 \times 3 \times 3$ output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



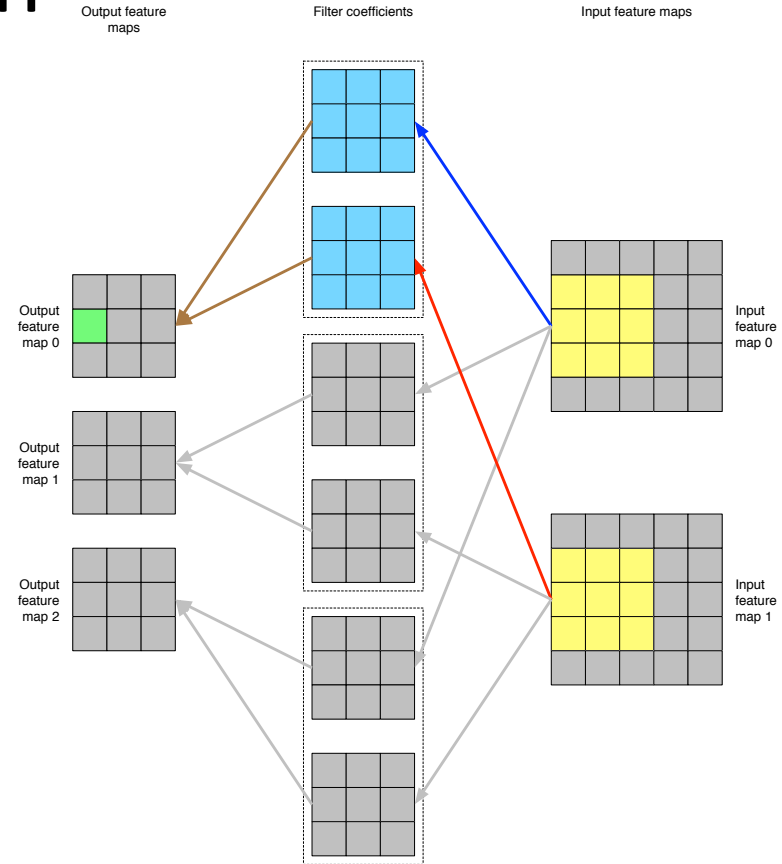
CNN Style 2D Convolution

- An example showing CNN style 2D convolution is matrix matrix multiplication using $2 \times 5 \times 5$ input feature maps, $3 \times 2 \times 3 \times 3$ filters and $3 \times 3 \times 3$ output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



CNN Style 2D Convolution

- An example showing CNN style 2D convolution is matrix matrix multiplication using $2 \times 5 \times 5$ input feature maps, $3 \times 2 \times 3 \times 3$ filters and $3 \times 3 \times 3$ output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



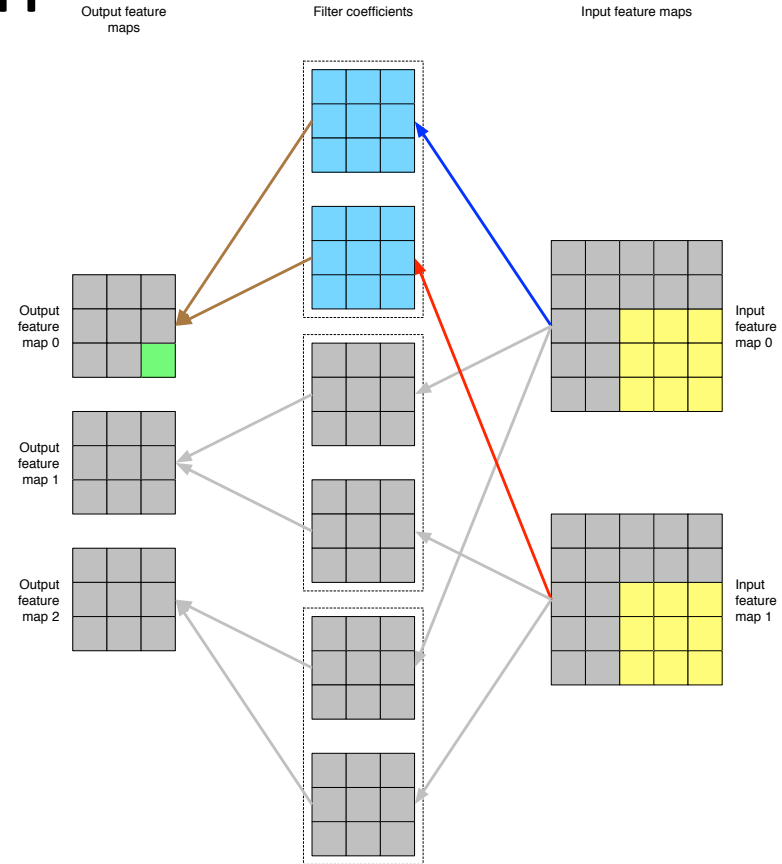
CNN Style 2D Convolution

- An example showing CNN style 2D convolution is matrix matrix multiplication using $2 \times 5 \times 5$ input feature maps, $3 \times 2 \times 3 \times 3$ filters and $3 \times 3 \times 3$ output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



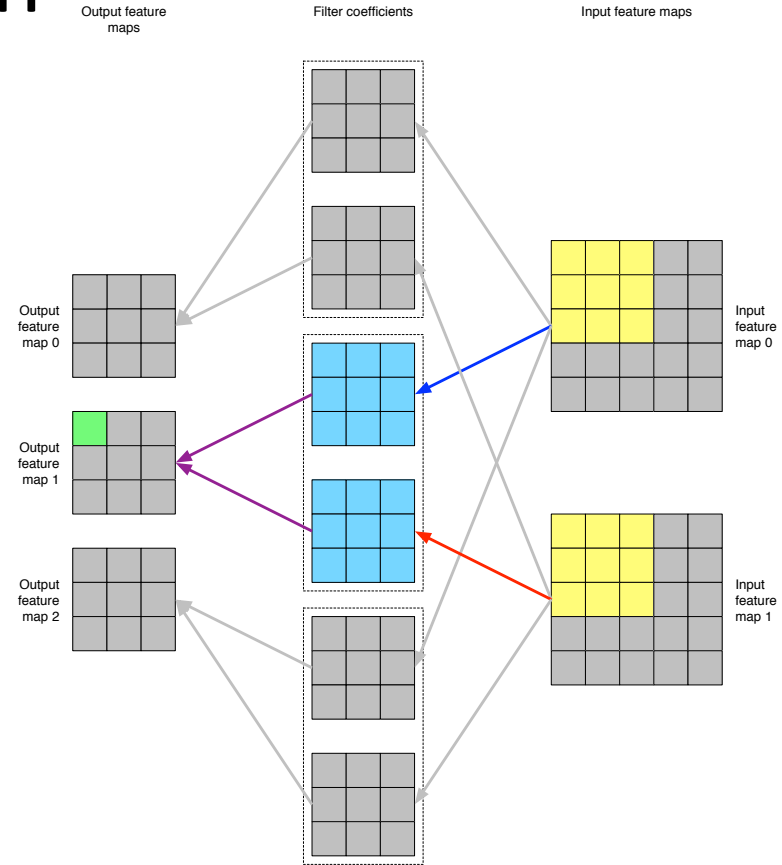
CNN Style 2D Convolution

- An example showing CNN style 2D convolution is matrix matrix multiplication using $2 \times 5 \times 5$ input feature maps, $3 \times 2 \times 3 \times 3$ filters and $3 \times 3 \times 3$ output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



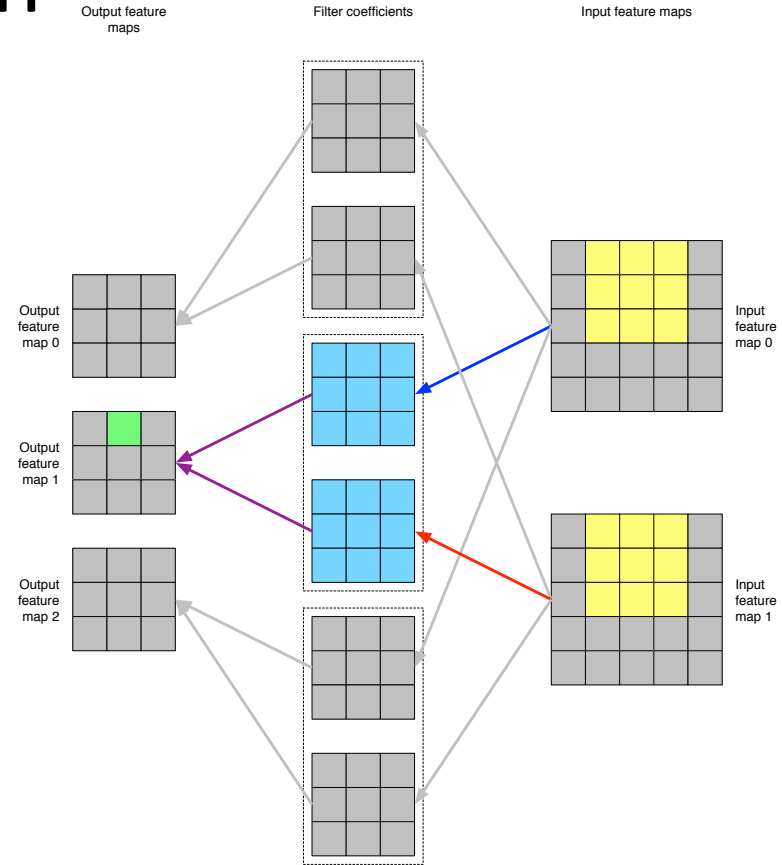
CNN Style 2D Convolution

- An example showing CNN style 2D convolution is matrix matrix multiplication using $2 \times 5 \times 5$ input feature maps, $3 \times 2 \times 3 \times 3$ filters and $3 \times 3 \times 3$ output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



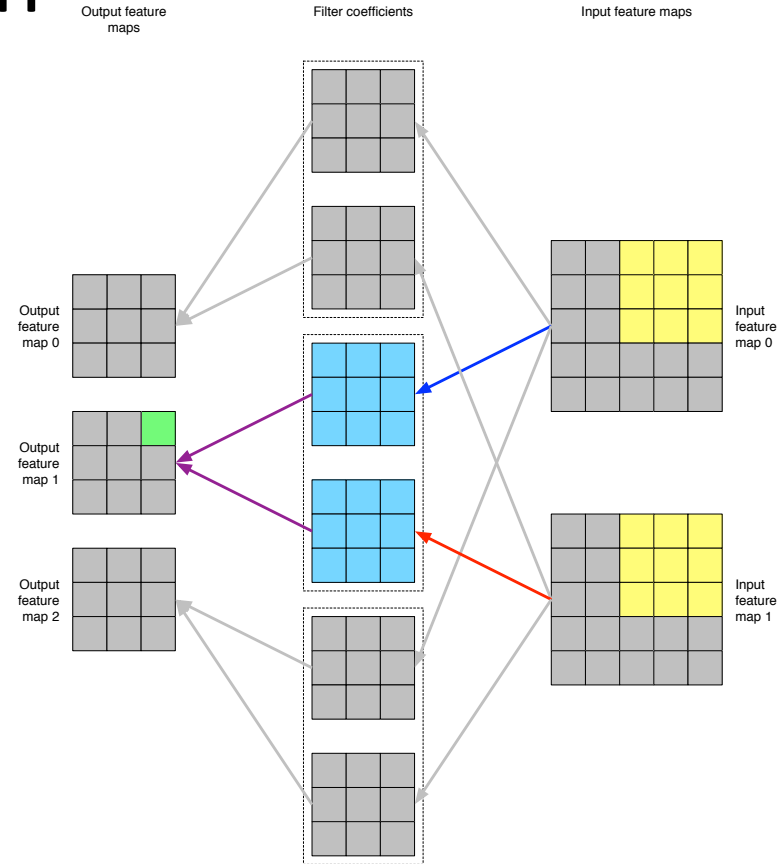
CNN Style 2D Convolution

- An example showing CNN style 2D convolution is matrix matrix multiplication using $2 \times 5 \times 5$ input feature maps, $3 \times 2 \times 3 \times 3$ filters and $3 \times 3 \times 3$ output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



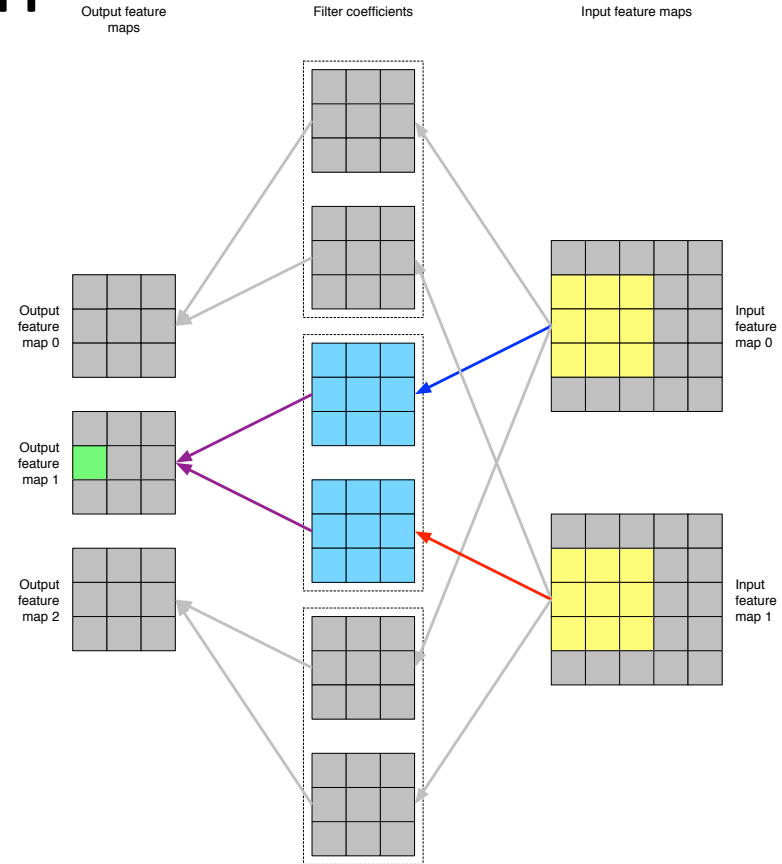
CNN Style 2D Convolution

- An example showing CNN style 2D convolution is matrix matrix multiplication using $2 \times 5 \times 5$ input feature maps, $3 \times 2 \times 3 \times 3$ filters and $3 \times 3 \times 3$ output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



CNN Style 2D Convolution

- An example showing CNN style 2D convolution is matrix matrix multiplication using $2 \times 5 \times 5$ input feature maps, $3 \times 2 \times 3 \times 3$ filters and $3 \times 3 \times 3$ output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



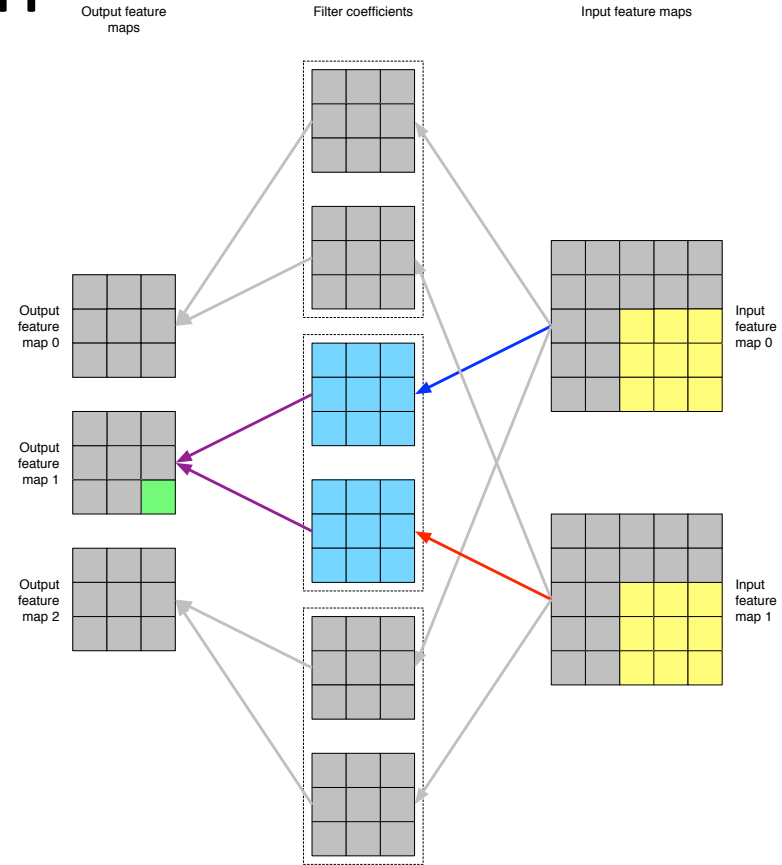
CNN Style 2D Convolution

- An example showing CNN style 2D convolution is matrix matrix multiplication using $2 \times 5 \times 5$ input feature maps, $3 \times 2 \times 3 \times 3$ filters and $3 \times 3 \times 3$ output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



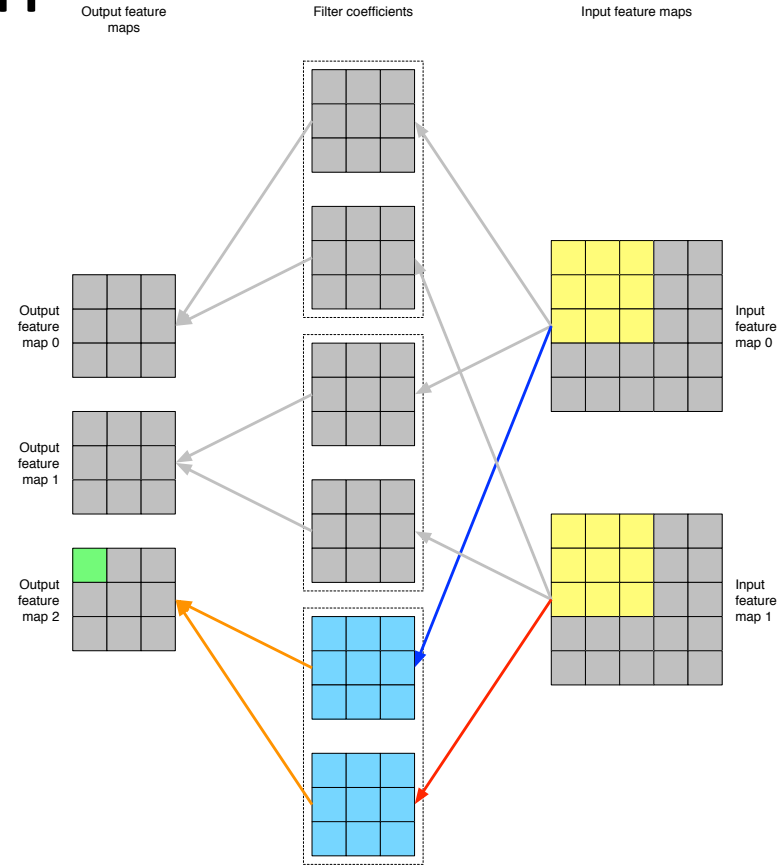
CNN Style 2D Convolution

- An example showing CNN style 2D convolution is matrix matrix multiplication using $2 \times 5 \times 5$ input feature maps, $3 \times 2 \times 3 \times 3$ filters and $3 \times 3 \times 3$ output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



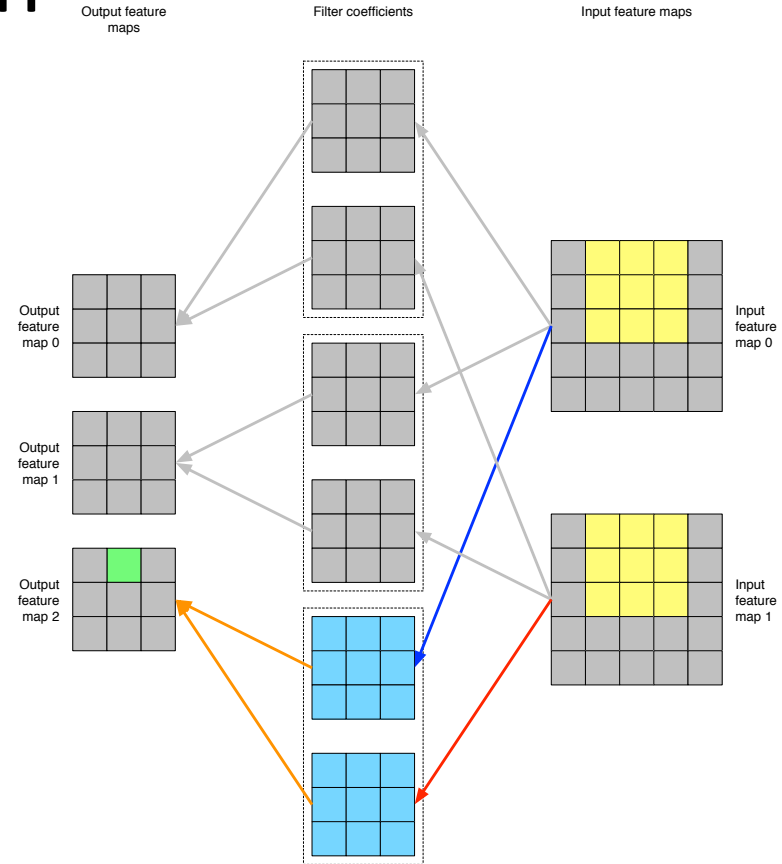
CNN Style 2D Convolution

- An example showing CNN style 2D convolution is matrix matrix multiplication using $2 \times 5 \times 5$ input feature maps, $3 \times 2 \times 3 \times 3$ filters and $3 \times 3 \times 3$ output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



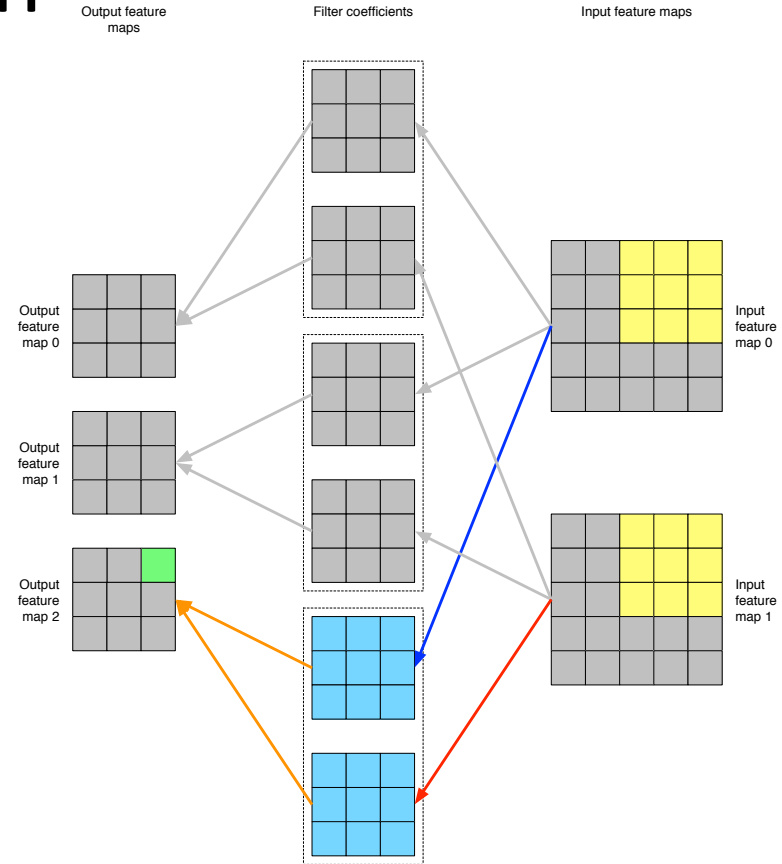
CNN Style 2D Convolution

- An example showing CNN style 2D convolution is matrix matrix multiplication using $2 \times 5 \times 5$ input feature maps, $3 \times 2 \times 3 \times 3$ filters and $3 \times 3 \times 3$ output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



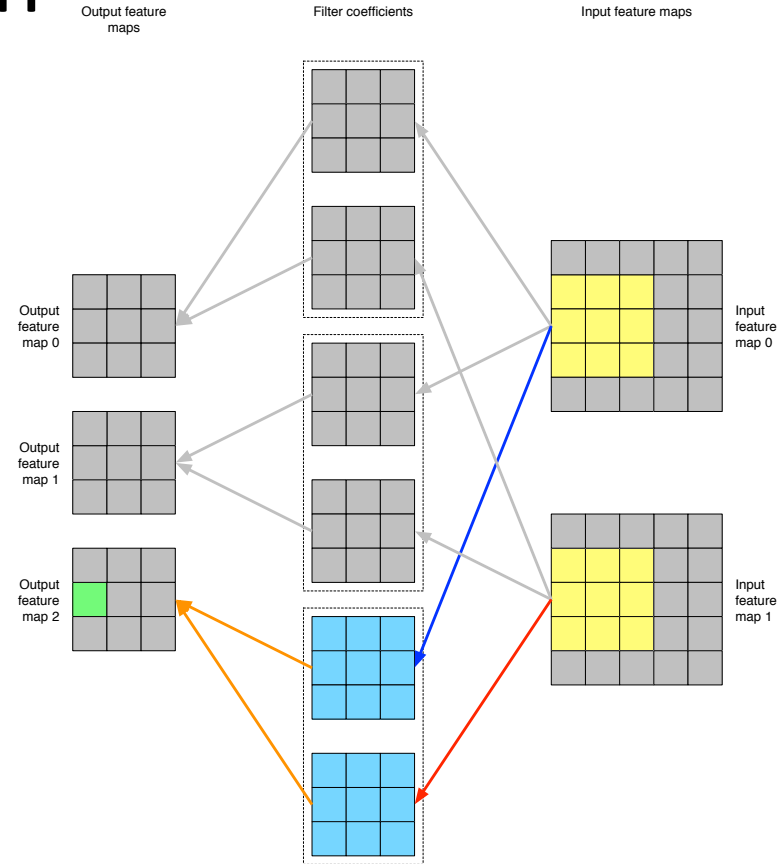
CNN Style 2D Convolution

- An example showing CNN style 2D convolution is matrix matrix multiplication using $2 \times 5 \times 5$ input feature maps, $3 \times 2 \times 3 \times 3$ filters and $3 \times 3 \times 3$ output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



CNN Style 2D Convolution

- An example showing CNN style 2D convolution is matrix matrix multiplication using $2 \times 5 \times 5$ input feature maps, $3 \times 2 \times 3 \times 3$ filters and $3 \times 3 \times 3$ output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



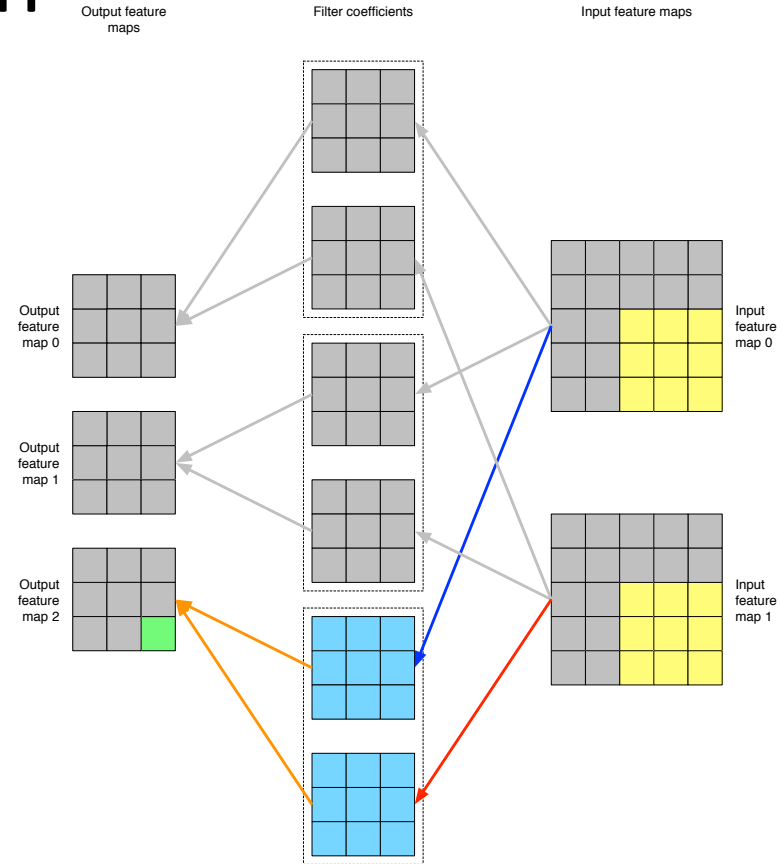
CNN Style 2D Convolution

- An example showing CNN style 2D convolution is matrix matrix multiplication using $2 \times 5 \times 5$ input feature maps, $3 \times 2 \times 3 \times 3$ filters and $3 \times 3 \times 3$ output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



CNN Style 2D Convolution

- An example showing CNN style 2D convolution is matrix matrix multiplication using $2 \times 5 \times 5$ input feature maps, $3 \times 2 \times 3 \times 3$ filters and $3 \times 3 \times 3$ output feature maps
- This sequence of figures illustrates the specific set of inputs and filter coefficients used to generate each output



CNN Style 2D Convolution

- Mathematically it's 6 loops (listed from common outer to inner)

- Output feature map channel $n_o = 0, \dots, N_o - 1$
- Output feature map row $m_r = 0, \dots, L_r - F_r = M_r - 1$
- Output feature map col $m_c = 0, \dots, L_c - F_c = M_c - 1$
- Input feature map channel $n_i = 0, \dots, N_i - 1$
- Filter row $f_r = 0, \dots, F_r - 1$
- Filter col $f_c = 0, \dots, F_c - 1$

- For each n_o , m_r and m_c

$$Y(n_o, m_r, m_c) = \sum_{n_i} \sum_{f_r} \sum_{f_c} H(n_o, n_i, f_r, f_c) X(n_i, m_r + f_r, m_c + f_c)$$

CNN Style 2D Convolution

- For each n_o , m_r and m_c

$$Y(n_o, m_r, m_c) = \sum_{n_i} \sum_{f_r} \sum_{f_c} H(n_o, n_i, f_r, f_c) X(n_i, m_r + f_r, m_c + f_c)$$

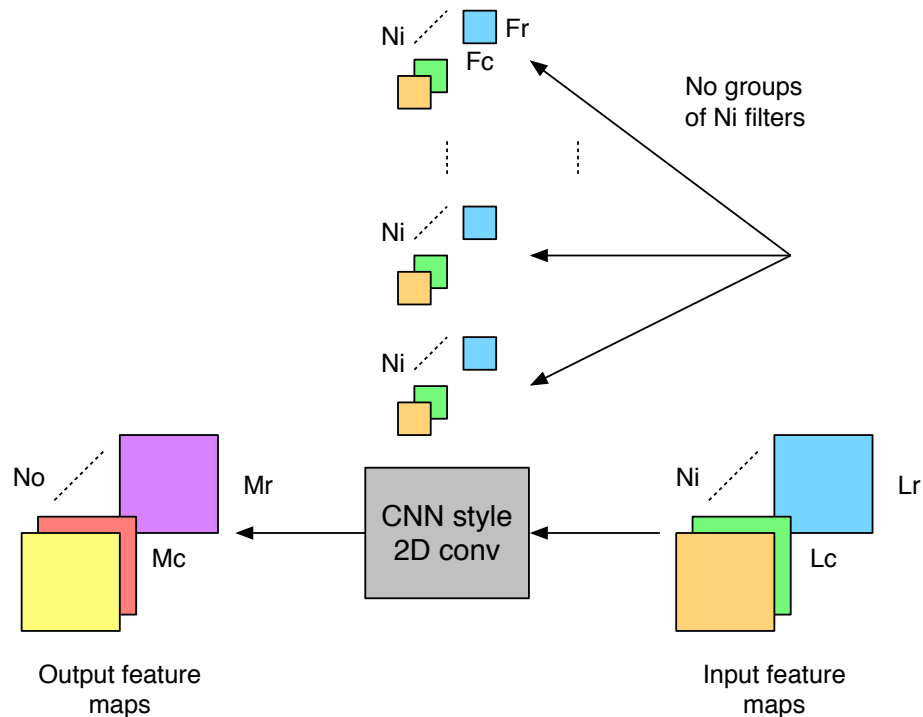
- Can be viewed as an inner product (by expanding the summations)
 - Of a vector formed from $N_i F_r F_c$ filter coefficients
 - With a vector formed from $F_r F_c$ elements of each N_i input feature maps
 - To produce a single output at the corresponding row col of an output feature map
- Repeated
 - For all row col values of the output feature map using the same filter coefficients
 - For all output feature map channels using different filter cxs for each output feature map channel
- Using 2D correlation instead of 2D convolution
 - Which is equivalent with a flip of the filter and indexing change

CNN Style 2D Convolution

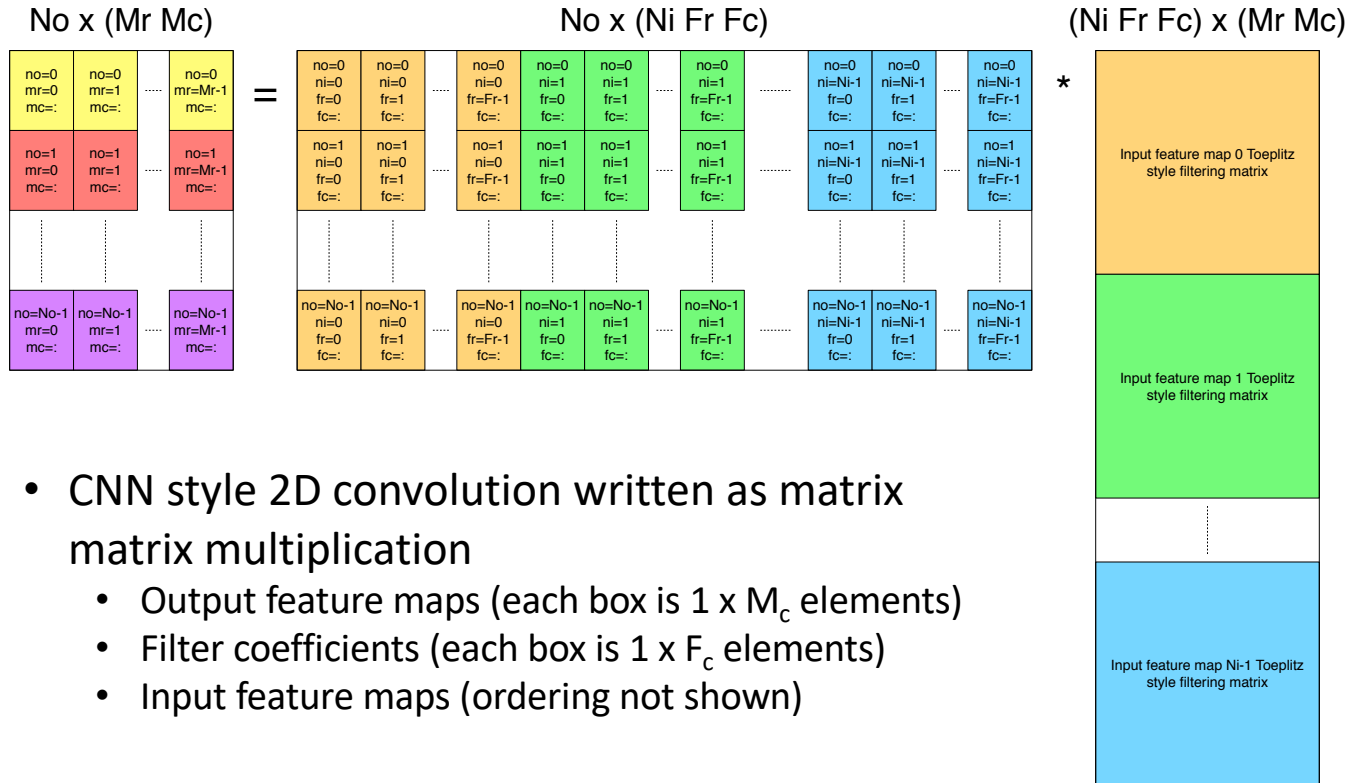
- High performance implementations of CNN style 2D convolution do not explicitly use 6 loops (but compute the same thing)
- Key realization is that CNN style 2D convolution is matrix matrix multiplication: $\mathbf{Y}^{2D} = \mathbf{H}^{2D} \mathbf{X}^{2D}$
 - \mathbf{H}^{2D} = reshape 4D filter coefficient tensor to 2D matrix
 - Trivial, nothing actually needs to be reshaped in practice
 - \mathbf{X}^{2D} = form 3D input feature map tensor into 2D Toeplitz style filtering matrix
 - This is the key
 - Will generate blocks of this on the fly as each input is repeated $\sim F_r F_c$ times
 - \mathbf{Y}^{2D} = compute 2D matrix of output feature maps
 - Matrix matrix multiplication is efficient on hardware
 - Trivial to reshape to 3D output feature map tensor, nothing actually needs to be done in practice

CNN Style 2D Convolution

- Starting point / reminder
- Input feature maps
 - 3D tensor
 - N_i inputs \times L_r rows \times L_c cols
- Filter coefficients
 - 4D tensor
 - N_o outputs \times N_i inputs \times F_r rows \times F_c cols
- Output feature maps
 - 3D tensor
 - N_o outputs \times M_r rows \times M_c cols



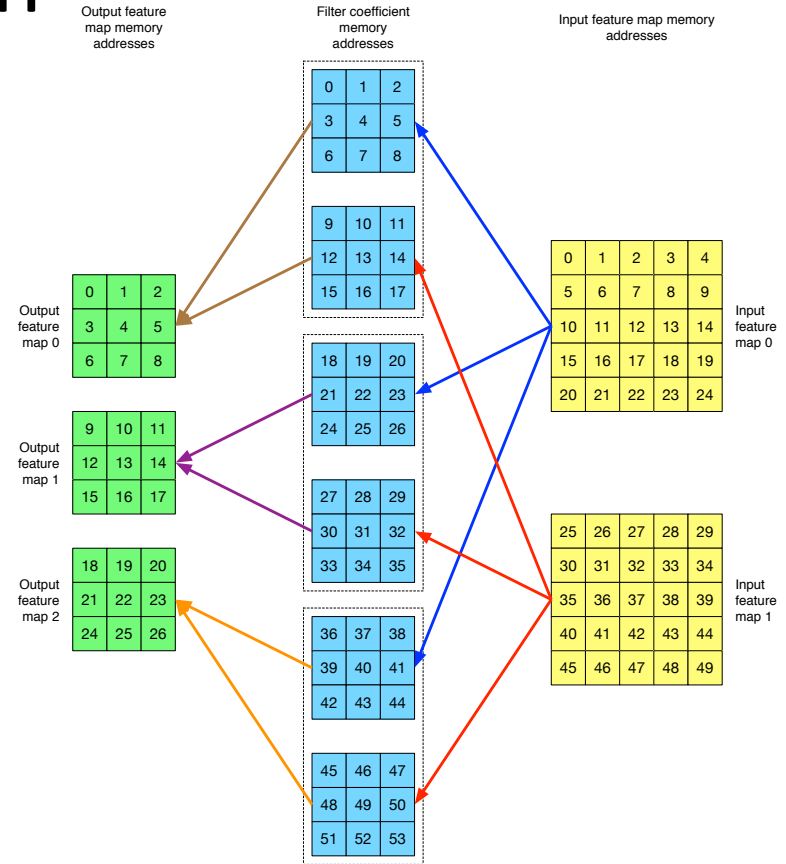
CNN Style 2D Convolution



- CNN style 2D convolution written as matrix multiplication
 - Output feature maps (each box is $1 \times M_c$ elements)
 - Filter coefficients (each box is $1 \times F_c$ elements)
 - Input feature maps (ordering not shown)

CNN Style 2D Convolution

- An example showing CNN style 2D convolution is matrix multiplication using $2 \times 5 \times 5$ input feature maps, $3 \times 2 \times 3 \times 3$ filters and $3 \times 3 \times 3$ output feature maps
- This figure illustrates memory addresses (specifically offsets to the initial pointer for each array)
- The next page shows where the memory addresses go in matrix multiplication



CNN Style 2D Convolution

0	1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26

Output feature map memory addresses
(note vectorization)

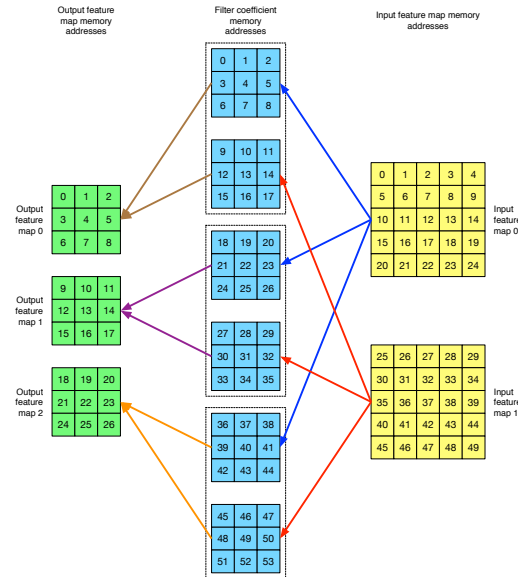
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53

Filter coefficient memory addresses
(note vectorization)

0	1	2	5	6	7	10	11	12
1	2	3	6	7	8	11	12	13
2	3	4	7	8	9	12	13	14
5	6	7	10	11	12	15	16	17
6	7	8	11	12	13	16	17	18
7	8	9	12	13	14	17	18	19
10	11	12	15	16	17	20	21	22
11	12	13	16	17	18	21	22	23
12	13	14	17	18	19	22	23	24
25	26	27	30	31	32	35	36	37
26	27	28	31	32	33	36	37	38
27	28	29	32	33	34	37	38	39
30	31	32	35	36	37	40	41	42
31	32	33	36	37	38	41	42	43
32	33	34	37	38	39	42	43	44
35	36	37	40	41	42	45	46	47
36	37	38	41	42	43	46	47	48
37	38	39	42	43	44	47	48	49

Input feature map memory addresses
(note Toeplitz filtering matrix structure)

- Main figure is matrix form
- Small figure is convolution form from previous page for reference



CNN Style 2D Convolution

- Limiting cases illustrated via depth wise separable convolution
 - Depth wise separable convolution splits a CNN style 2D conv layer into 2 layers
 - Traditional 2D convolution
 - CNN style 2D convolution with 1×1 filters
 - Less generality of either vs original, but 1 extra level of depth
 - Traditional 2D convolution to mix across space ($N_i = N_o = 1$)
 - Can also get small values of N_i and N_o via grouping
 - Equivalent to vector matrix multiplication
 - Note that K dimension reduces from $(N_i F_r F_c)$ to $(F_r F_c)$
 - CNN style 2D convolution with 1×1 filters to mix across channel
 - Equivalent to standard matrix matrix multiplication
 - Note that K dimension reduces from $(N_i F_r F_c)$ to N_i

CNN Style 2D Convolution

0	1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26

Output feature map memory addresses
(note vectorization)

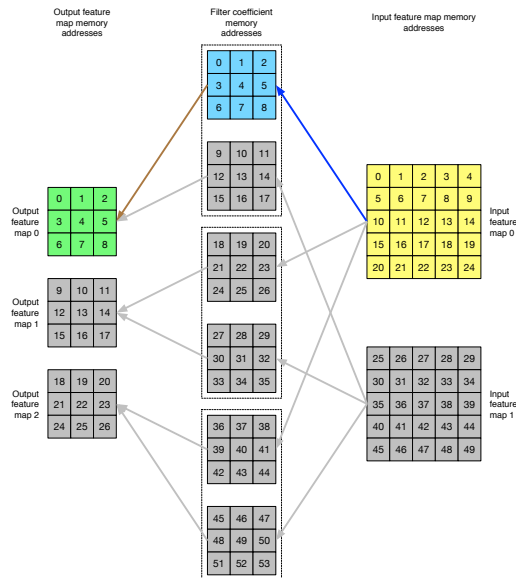
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53

Filter coefficient memory addresses
(note vectorization)

0	1	2	5	6	7	10	11	12
1	2	3	6	7	8	11	12	13
2	3	4	7	8	9	12	13	14
5	6	7	10	11	12	15	16	17
6	7	8	11	12	13	16	17	18
7	8	9	12	13	14	17	18	19
10	11	12	15	16	17	20	21	22
11	12	13	16	17	18	21	22	23
12	13	14	17	18	19	22	23	24
25	26	27	30	31	32	35	36	37
26	27	28	31	32	33	36	37	38
27	28	29	32	33	34	37	38	39
30	31	32	35	36	37	40	41	42
31	32	33	36	37	38	41	42	43
32	33	34	37	38	39	42	43	44
35	36	37	40	41	42	45	46	47
36	37	38	41	42	43	46	47	48
37	38	39	42	43	44	47	48	49

Input feature map memory addresses
(note Toeplitz filtering matrix structure)

- Traditional convolution
- Equivalent to vector matrix multiplication

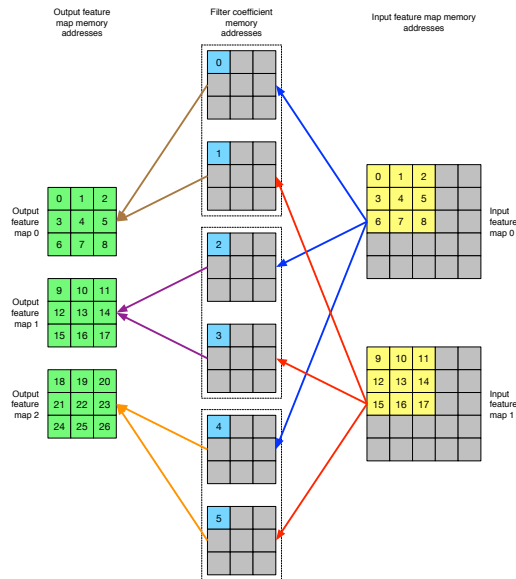


0								1							
2								3							
4								5							

Filter coefficient memory addresses
(note vectorization)

[illegible]

- CNN style convolution with 1x1 filters
- Equivalent to pure matrix multiplication



Input
feature
map
memory
addresses
(note
Toeplitz
filtering
matrix
structure)

CNN Style 2D Convolution

- This is important
- Intuition of feature extraction
 - CNN style 2D convolution is a linear transformation
 - Output feature maps matrix = filter coefficient matrix * input feature maps filtering matrix
 - Matrix vector multiplication as used in a fully connected layer of a neural network had the intuition of matching features to inputs over channel
 - For CNN style 2D convolution have the intuition of matching features to inputs over channel and space
 - How far can it see in space? For 1 layer? For repeated layers?
 - How many features does it work over?

CNN Style 2D Convolution

- Intuition of bias
 - Add a constant to all elements in an output feature map
 - Can be a different constant for each output feature map
 - Affine transformation
 - Allows the dividing line to shift
 - Implementation using a rank 1 outer product
- Intuition of ReLU
 - Removes negatively aligned features or predictions
 - Allows depth
 - Subsequent layers combine positively aligned extracted features

CNN Style 2D Convolution

- Memory
 - Formulas
 - Input feature maps: $N_i L_r L_c$
 - Output feature maps: $N_o M_r M_c$
 - Filter coefficients: $N_i N_o F_r F_c$
 - Early in the network feature map memory tends to dominate
 - Deeper in the network filter coefficient memory tends to dominate

CNN Style 2D Convolution

- Compute
 - Formulas
 - $(N_o) (M_r M_c) (N_i F_r F_c)$ (MACs)
= $(N_o M_r M_c) (N_i F_r F_c)$
= (number of outputs) (number of input MACs per output)
 - Tends to be highest in the beginning of the network
 - If $(M_r M_c)$ is more aggressively reduced than $(N_i N_o)$ is increased
 - Scaling the input size by 1/2 in rows and cols ~ reduces compute by 1/4

CNN Style 2D Convolution

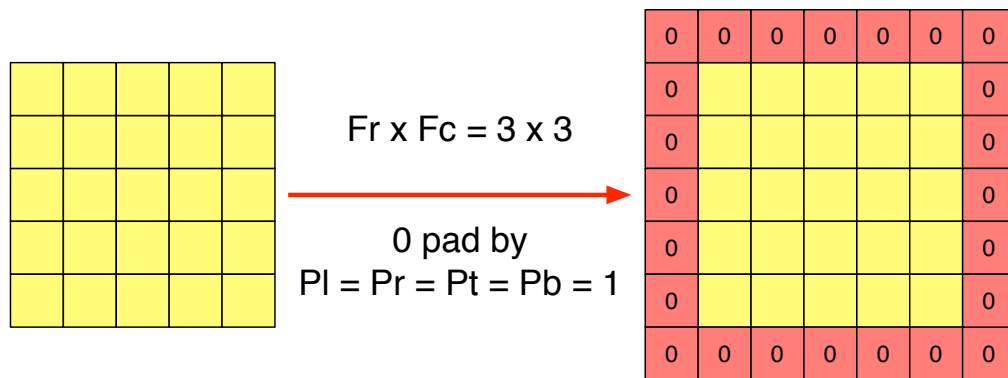
- Arithmetic intensity

- Compute $= N_i N_o F_r F_c M_r M_c$ (MACs)
- Data movement $= N_i L_r L_c + N_o M_r M_c + N_i N_o F_r F_c$ (elements)
- Ratio $= \text{compute} / \text{data movement}$

CNN Style 2D Convolution

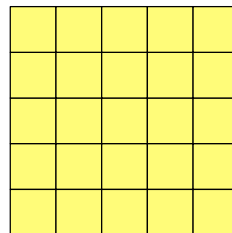
- Variant: input feature map 0 padding

- P_l left, P_r right, P_t top, P_b bottom
 - Typically $P_l + P_r = F_c - 1$ and $P_t + P_b = F_r - 1$
 - Used for same size input / output feature maps
 - Implementation key is efficient 0 insert

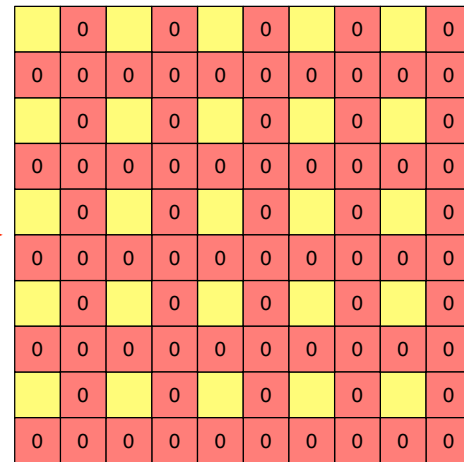


CNN Style 2D Convolution

- Variants: input feature map up sampling
 - U_r rows, U_c cols
 - Typically called deconvolution
 - Used in decoder style head designs
 - Implementation key is input memory reuse
 - Alternatives are bilinear and nearest neighbor interpolation

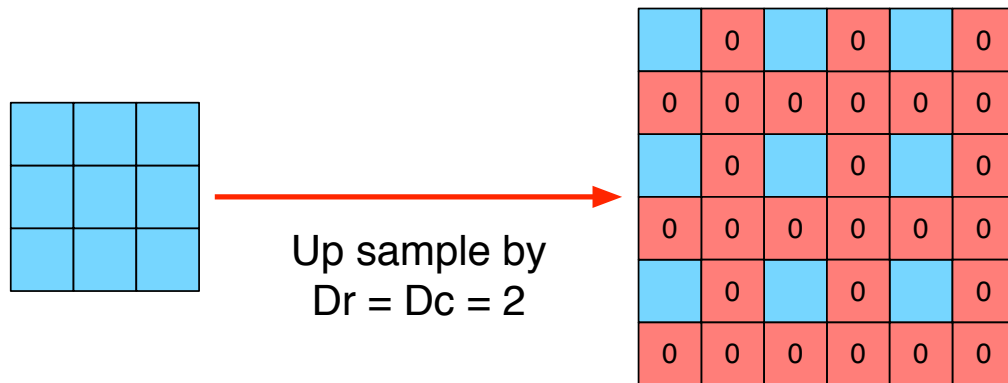


Up sample by
 $U_r = U_c = 2$



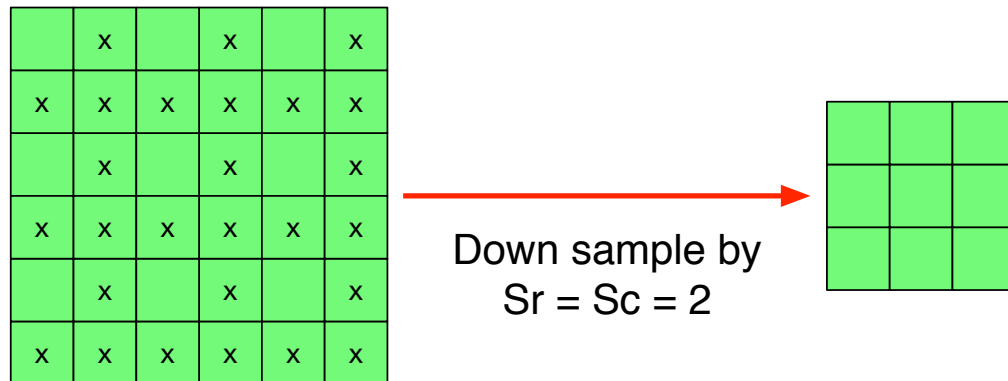
CNN Style 2D Convolution

- Variants: filter coefficient up sampling
 - D_r rows, D_c cols
 - Typically called dilated or Atrous convolution
 - Used to maintain spatial resolution with large receptive field
 - Implementation key is input feature map filtering matrix row removal



CNN Style 2D Convolution

- Variants: output feature map down sampling
 - S_r rows, S_c cols
 - Typically called strided convolution
 - Used to reduce spatial resolution
 - Implementation key is input feature map filtering matrix column removal
 - Alternative is pooling



CNN Style 2D Convolution

To do: add a pic showing this in graph format to connect to back propagation

- Alternate view of CNN style 2D convolution
 - Add together $F_r \times F_c$ CNN style 2D convolutions with 1×1 filter size
 - Reminder: CNN style convolution with 1×1 filters is pure matrix matrix multiplication
 - Input size is reduced to $M_r \times M_c$ for each with an appropriate shift / offset of the original input
 - Tradeoffs
 - Advantage of simpler input feature map matrix structure and associated data movement logic
 - Drawback of additional input feature map memory movement
 - Side benefit: useful for understanding back propagation through a CNN style 2D convolution layer

Linear Pre Processing

Discrete Fourier Transform

- A linear transformation from domain to 1/domain via a projection onto a complex exponential basis
 - Example: time to 1/time = frequency
 - Index ranges
 - $k = 0, \dots, K - 1$
 - $n = 0, \dots, K - 1$

$$y(k) = (1/\text{sqrt}(K)) \sum_n x(n) e^{-i(2\pi/K)nk}$$

Discrete Fourier Transform

- Can be written as a $K \times K$ DFT matrix \mathbf{F}_K that transforms input vectors \mathbf{x} to output vectors \mathbf{y}
 - $\mathbf{y} = \mathbf{F}_K \mathbf{x}$ where $F_K(a, b) = (1/\sqrt{K}) e^{-i(2\pi/K)ab}$
 - \mathbf{F}_K is a unitary matrix so it's invertible (conj transpose = inverse, called the IDFT)
 - Output is typically circular complex Gaussian (will discuss implications later)
 - Fast implementations possible
 - $O(K \log K)$ for fast Fourier transform (FFT)
 - Vs $O(K^2)$ for DFT

Discrete Fourier Transform

- Data transformation
 - Sometimes it's easier to do feature extraction in the frequency domain vs time domain
 - Common example of this is speech to text
 - Unitary so invertible (no information lost (until you read the next slide))
 - Effectively lets the network decide what data to keep and what data to throw away

Discrete Fourier Transform

- Dimensionality reduction
 - The DFT frequently concentrates the majority of information in naturally occurring signals to $L < K$ basis components
 - A common dimensionality reduction strategy is to keep the L main components and get rid of the rest
 - General comment: often want to keep as much information as possible while getting rid of as much data as possible
 - Difficulty: knowing that it's really ok to get rid of data

Principal Component Analysis

- Note
 - A little bit of this is dependent on probability for parts of the understanding
- Setup
 - $M \times K$ data matrix \mathbf{X}
 - Each row is a different trial (ex: point in time)
 - Each column is a different measurement from that trial (ex: different stock)
 - Columns are normalized to 0 mean
 - Columns are potentially linearly correlated

Principal Component Analysis

- Goal
 - Linearly transform to a new $M \times K$ matrix \mathbf{Y} via a $K \times K$ matrix \mathbf{Q}

$$\mathbf{Y} = \mathbf{X} \mathbf{Q}$$

- Where \mathbf{Q} is chosen such that columns of \mathbf{Y} are orthogonal and ordered from largest to smallest variance
- For dimensionality reduction keep first $L < K$ columns

Principal Component Analysis

- Mechanics for finding \mathbf{Q}
 - Note that $\mathbf{X} = \mathbf{U} \mathbf{S} \mathbf{V}^T$ via SVD
 - \mathbf{U} is a $M \times M$ orthogonal matrix of eigenvectors of $\mathbf{X} \mathbf{X}^T$
 - An eigenvector \mathbf{v} of matrix \mathbf{A} is a nonzero vector \mathbf{v} that satisfies $\mathbf{A} \mathbf{v} = \lambda \mathbf{v}$
 - Intuition of matrix scaling eigenvectors inputs by the eigenvalue λ
 - Side note: multicarrier modulation exploits this
 - \mathbf{S} is a $M \times K$ diagonal matrix of singular values
 - \mathbf{V}^T is a $K \times K$ orthogonal matrix of eigenvectors of $\mathbf{X}^T \mathbf{X}$
 - Select $\mathbf{Q} = \mathbf{V}$

$$\mathbf{Y} = \mathbf{X} \mathbf{Q} = \mathbf{U} \mathbf{S} \mathbf{V}^T \mathbf{V} = \mathbf{U} \mathbf{S}$$

Principal Component Analysis

- Example
 - Statistical arbitrage (e.g., SPY, MDY and IJR)
 - Stock 0 time series in col 0, stock 1 time series in col 1, ..., stock $K - 1$ time series in col $K - 1$
 - 0 th principal component for trend trading (you would keep this for feature extract)
 - $K - 1$ th principal component for stat arb (throw away for feature extract)
- Project idea (while on the topic of finance)
 - Information extraction project
 - Prediction of the pmf of various stocks / ETFs at different times into the future
 - Show that it's possible testing out of sample and taking transaction friction into account to profit

References

List

- Convolutional neural networks: theory, implementation and application
 - Chapter 3 linear algebra
 - <https://github.com/arthurredfern/UT-Dallas-CS-6301-CNNs/blob/master/References/ConvolutionalNeuralNetworks.pdf>
- Linear algebra
 - <https://www.math.ucdavis.edu/~linear/linear-guest.pdf>
- A guide to convolution arithmetic for deep learning
 - <https://arxiv.org/abs/1603.07285>