MO434 - Deep Learning
Fundamentals for Image Analysis by DL

Alexandre Xavier Falcão

Institute of Computing - UNICAMP

afalcao@ic.unicamp.br

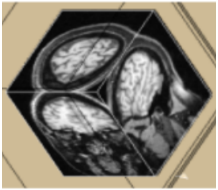
Agenda

- Multichannel images and tensors.
- Adjacency relation.
- Patches and kernels.
- Convolution, activation, pooling and normalization.
- Applications to image analysis.

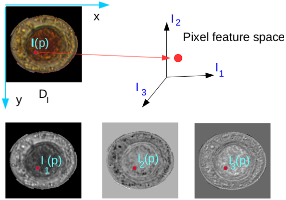
Multichannel images and tensors

A multichannel image $\hat{I} = (D_I, \mathbf{I})$ of dimension n consists of an array $D_I \in Z^n$ of spels (space elements – e.g., pixels in 2D, voxels in 3D) such that each spel $p \in D_I$ is represented by m channel values in a feature vector $\mathbf{I}(p) = (I_1(p), I_2(p), \ldots, I_m(p)) \in \Re^m$.

(a) n=2 and m=1.



(b) n=3 and m=1.



(c) n=2 and m=3.

The mappings I_j, $j \in [1, m]$, are called channels (nD arrays). Our focus will be on images with $n = 2$ and $m \geq 1$.

Multichannel images and tensors

- The domain $D_I \in \mathbb{Z}^2$ of image $\hat{I}$ forms a matrix with $ncols \times nrows$ cells (i.e., spatial dimensions $xsize \times ysize$).

Multichannel images and tensors

- The domain $D_I \in \mathbb{Z}^2$ of image $\hat{I}$ forms a matrix with *ncols* $\times$ *nrows* cells (i.e., spatial dimensions *xsize* $\times$ *ysize*).

- Since each element of that matrix has m values (channels), we need a 3D array with *xsize* $\times$ *ysize* $\times$ m cells.

Multichannel images and tensors

- The domain $D_I \in \mathbb{Z}^2$ of image $\hat{I}$ forms a matrix with $ncols \times nrows$ cells (i.e., spatial dimensions $xsize \times ysize$).
- Since each element of that matrix has m values (channels), we need a 3D array with $xsize \times ysize \times m$ cells.
- We can also store N images of the same dimensions in a 4D array with $N \times xsize \times ysize \times m$ cells.

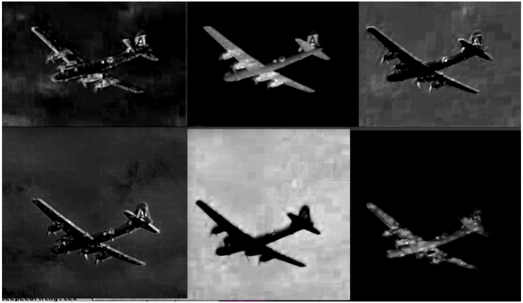
Multichannel images and tensors

- The domain $D_I \in \mathbb{Z}^2$ of image $\hat{I}$ forms a matrix with *ncols* $\times$ *nrows* cells (i.e., spatial dimensions *xsize* $\times$ *ysize*).
- Since each element of that matrix has m values (channels), we need a 3D array with *xsize* $\times$ *ysize* $\times$ m cells.
- We can also store N images of the same dimensions in a 4D array with $N \times$ *xsize* $\times$ *ysize* $\times$ m cells.
- Such multidimensional arrays are called tensors and they are also used to store the weights and biases of the neural network.

Multichannel images

As we will see, the convolution between an image and k kernels (filters), both with dimension $n = 2$ and $m = 3$ channels, results into another image with dimension $n = 2$ and $m = k$ channels.

(a) $n = 2$ and $m = 3$.



(b) $n = 2$ and $m = 6$.

The output tensor has $xsize \times ysize \times k$ cells.

Adjacency relation

- An adjacency relation $\mathcal{A} \subset D_I \times D_I$ is a relation between pixels that satisfy distance-based properties. For instance,

$$\mathcal{A} = \{(p, q) \in D_I \mid \sqrt{(x_q - x_p)^2 + (y_q - y_p)^2} \leq r\},$$

for $r \geq 1$, $p = (x_p, y_p)$ and $q = (x_q, y_q)$.

Adjacency relation

- An adjacency relation $\mathcal{A} \subset D_I \times D_I$ is a relation between pixels that satisfy distance-based properties. For instance,

$$\mathcal{A} = \{(p, q) \in D_I \mid \sqrt{(x_q - x_p)^2 + (y_q - y_p)^2} \leq r\},$$

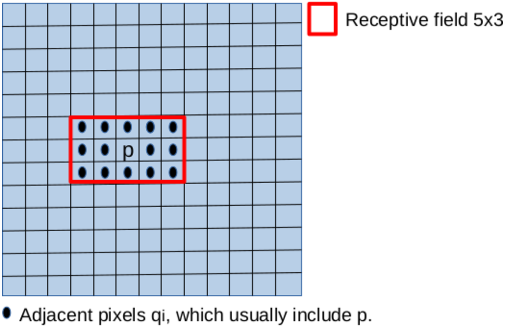
  for $r \geq 1$, $p = (x_p, y_p)$ and $q = (x_q, y_q)$.

- We may say that $q_i \in \mathcal{A}(p)$ (adjacency set of p) when $q_i - p \in \{(dx_{q_i}, dy_{q_i})\}_{i=1}^{|\mathcal{A}|}$ (a set of displacements) – i.e.,

$$(x_{q_i}, y_{q_i}) = (x_p, y_p) + (dx_{q_i}, dy_{q_i}).$$

Adjacency relation

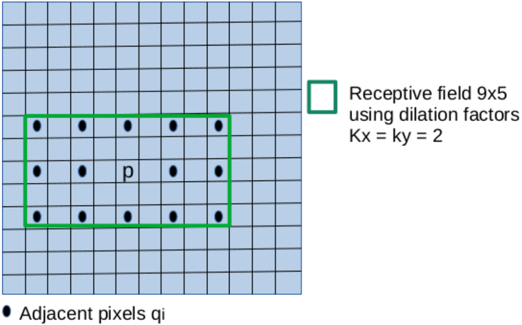
By imagining $p \in D_I$ as a neuron, such displacements usually define a receptive field in $\hat{I}$ of sizes $w \times h$ around p, for $\frac{w}{2} = \max_{q_i \in \mathcal{A}}\{|dx_{q_i}|\}$ and $\frac{h}{2} = \max_{q_i \in \mathcal{A}}\{|dy_{q_i}|\}$.



It is also common to define $w = h$ and add $(\frac{w}{2}, \frac{h}{2})$ zeros around the image (padding) to guarantee adjacency sets of the same size for all pixels p.

Adjacency relation

One may also increase the receptive field without increasing the number of adjacents (synaptic connections): $q_i \in \mathcal{A}(p)$, when $q_i - p \in \{(kxdx_{q_i}, kydy_{q_i})\}_{i=1}^{|\mathcal{A}|}$, with (kx, ky) being dilation factors.



Patches and kernels

- The pair $(\mathcal{A}, \mathbf{I})_p$ defines a subimage (patch) around any pixel $p \in D_I$ with values $\mathbf{I}(q)$, $q \in \mathcal{A}(p)$.

Patches and kernels

- The pair $(\mathcal{A}, \mathbf{I})_p$ defines a subimage (patch) around any pixel $p \in D_I$ with values $\mathbf{I}(q)$, $q \in \mathcal{A}(p)$.

- A kernel (filter) is also a pair $(\mathcal{A}, \mathbf{W})$ which assigns fixed synaptic weights $\mathbf{W}(q)$, $q \in \mathcal{A}(p)$, indepent of $p \in D_I$.

Patches and kernels

- The pair $(\mathcal{A}, \mathbf{I})_p$ defines a subimage (patch) around any pixel $p \in D_I$ with values $\mathbf{I}(q)$, $q \in \mathcal{A}(p)$.
- A kernel (filter) is also a pair $(\mathcal{A}, \mathbf{W})$ which assigns fixed synaptic weights $\mathbf{W}(q)$, $q \in \mathcal{A}(p)$, indepent of $p \in D_I$.
- Note that such fixed synaptic weights considerably reduce the number of parameters to be estimated, explaining the success of convolution in DL.

Patches and kernels

- The pair $(\mathcal{A}, \mathbf{I})_p$ defines a subimage (patch) around any pixel $p \in D_I$ with values $\mathbf{I}(q)$, $q \in \mathcal{A}(p)$.

- A kernel (filter) is also a pair $(\mathcal{A}, \mathbf{W})$ which assigns fixed synaptic weights $\mathbf{W}(q)$, $q \in \mathcal{A}(p)$, indepent of $p \in D_I$.

- Note that such fixed synaptic weights considerably reduce the number of parameters to be estimated, explaining the success of convolution in DL.

- Patches and kernels can also be stored in tensors of sizes $w \times h \times m$.

Convolution

The convolution between an image $\hat{I} = (D_I, \mathbf{I})$ and a kernel $(\mathcal{A}, \mathbf{W})$ results a single-channel image $\hat{J} = (D_J, J)$, with

$$J(p) = \sum_{i=1}^{|\mathcal{A}|} \langle \mathbf{I}(q_i), \mathbf{W}(q_i) \rangle,$$

for $p \in D_I$.

Convolution

The convolution between an image $\hat{I} = (D_I, \mathbf{I})$ and a kernel $(\mathcal{A}, \mathbf{W})$ results a single-channel image $\hat{J} = (D_J, J)$, with

$$J(p) = \sum_{i=1}^{|\mathcal{A}|} \langle \mathbf{I}(q_i), \mathbf{W}(q_i) \rangle,$$

for $p \in D_I$.

The convolution with a kernel bank $\{(\mathcal{A}, \mathbf{W}_k)\}_{k=1}^{b}$ of b kernels results into a multichannel image $\hat{J} = (D_J, \mathbf{J})$ with b channels $\mathbf{J}(p) = (J_1(p), J_2(p), \ldots, J_b(p))$,

$$J_k(p) = \sum_{i=1}^{|\mathcal{A}|} \langle \mathbf{I}(q_i), \mathbf{W}_k(q_i) \rangle,$$

$k \in [1, b]$, $p \in D_I$.

Activation

Any activation function can then be applied to an output $J_k(p)$, $k \in [1, b]$. For instance, the Rectified Linear Unit (ReLU).

Activation

Any activation function can then be applied to an output $J_k(p)$, $k \in [1, b]$. For instance, the Rectified Linear Unit (ReLU).

From the output $\hat{J} = (D_I, \mathbf{J})$, ReLU creates an image $\hat{R} = (D_I, \mathbf{R})$, $\mathbf{R}(p) = (R_1(p), R_2(p), \ldots, R_b(p))$,

$$R_k(p) \quad = \quad \max\{0, J_k(p)\},$$

for $p \in D_I$ and $k \in [1, b]$.

Activation

Any activation function can then be applied to an output $J_k(p)$, $k \in [1, b]$. For instance, the Rectified Linear Unit (ReLU).

From the output $\hat{J} = (D_I, \mathbf{J})$, ReLU creates an image $\hat{R} = (D_I, \mathbf{R})$, $\mathbf{R}(p) = (R_1(p), R_2(p), \ldots, R_b(p))$,

$$R_k(p) \quad = \quad \max\{0, J_k(p)\},$$

for $p \in D_I$ and $k \in [1, b]$.

Image function J

6	1	7	-9
9	8	-2	-17
12	13	-10	-19
11	11	-9	-13

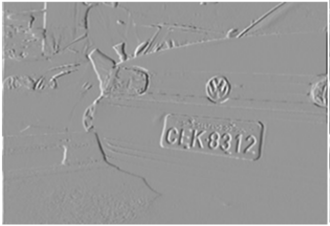
Image function R

6	1	7	0
9	8	0	0
12	13	0	0
11	11	0	0

Convolution followed by activation

Kernel 3 x 3 x 1

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

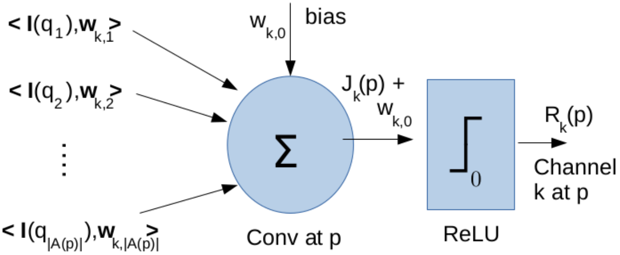


After convolution

After ReLU

Transitions from dark to bright are enhanced.

By convolving $\hat{I} = (D_I, \mathbf{I})$ and the k-th filter $\{(\mathcal{A}, \mathbf{W}_k)\}$, $k \in [1, b]$, adding a bias $w_{k,0} \in \Re$ to each output $J_k(p)$, and applying a ReLU operation, we have one perceptron per pixel $p \in D_I$ (neuron).



$$J_k(p) = \sum_{i=1}^{|A(p)|} \langle I(q_i), w_{k,i} \rangle$$

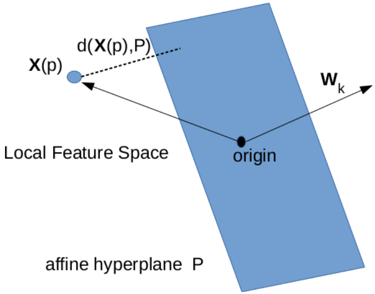
$$R_k(p) = \max\{0, J_k(p) + w_{k,0}\}$$

Convolution, bias, and activation

Let $\mathbf{X}(p) \in \Re^{|\mathcal{A}(p)| \times m}$ be a patch $(\mathcal{A}, \mathbf{I})_p$ (local feature vector),

$$\mathbf{X}(p) = (\mathbf{I}(q_1), \mathbf{I}(q_2), \ldots, \mathbf{I}(q_{|\mathcal{A}(p)|}))$$

and P be an affine hyperplane $\langle \mathbf{X}(p), \mathbf{W}_k \rangle + w_{k,0} = 0$ in $\Re^{|A(p)| \times m}$.



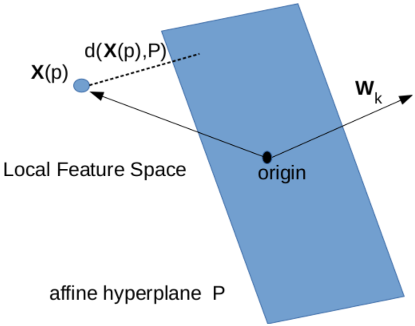
The distance $d(\mathbf{X}(p), P)$ from $\mathbf{X}(p)$ to the hyperplane is given by

$$d(\mathbf{X}(p), P) = \frac{\langle \mathbf{X}(p), \mathbf{W}_k \rangle + w_{k,0}}{\|\mathbf{W}_k\|} = \frac{J_k(p) + w_{k,0}}{\|\mathbf{W}_k\|}$$

Convolution, bias, and activation

The perceptron at p selects $R_k(p)$ as a local feature only when the activation

$$\langle \mathbf{X}(p), \mathbf{W}_k \rangle + w_{k,0} \quad = \quad J_k(p) + w_{k,0} > 0,$$



meaning that, the bias moves P such that $\mathbf{X}(p)$ falls in its positive side.

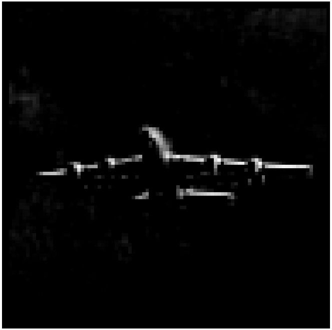
Convolution, bias, and activation

Therefore, convolution, bias, and activation — a neuronal layer (layer of perceptrons $p \in D_I$) — should extract and select pixel features in parts that best represent the object characteristics for image analysis.

Output of activation for four random kernels: some kernels may be better than others and some may be redundants.

Convolution, bias, and activation

Therefore, convolution, bias, and activation — a neuronal layer (layer of perceptrons $p \in D_I$) — should extract and select pixel features in parts that best represent the object characteristics for image analysis.



Output of activation for four random kernels: some kernels may be better than others and some may be redundants.

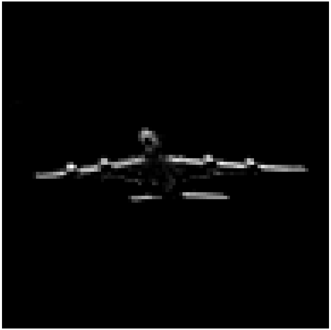
Convolution, bias, and activation

Therefore, convolution, bias, and activation — a neuronal layer (layer of perceptrons $p \in D_I$) — should extract and select pixel features in parts that best represent the object characteristics for image analysis.

Output of activation for four random kernels: some kernels may be better than others and some may be redundants.

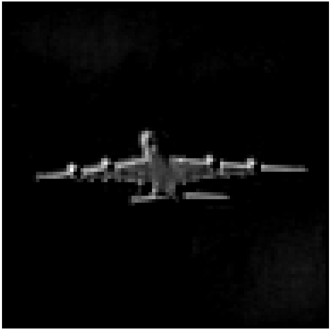
Convolution, bias, and activation

Therefore, convolution, bias, and activation — a neuronal layer (layer of perceptrons $p \in D_I$) — should extract and select pixel features in parts that best represent the object characteristics for image analysis.



Output of activation for four random kernels: some kernels may be better than others and some may be redundants.

Therefore, convolution, bias, and activation — a neuronal layer (layer of perceptrons $p \in D_I$) — should extract and select pixel features in parts that best represent the object characteristics for image analysis.



Output of activation for four random kernels: some kernels may be better than others and some may be redundants.

Pooling

The activations $R_k(p)$ related to an object of interest might also appear at nearby positions within and across images.

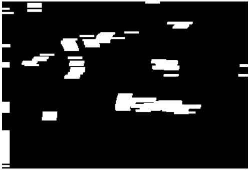
Pooling

The activations $R_k(p)$ related to an object of interest might also appear at nearby positions within and across images.

Max-pooling can aggregate them by transforming $\hat{R} = (D_I, \mathbf{R})$ into $\hat{P} = (D_P, \mathbf{P})$, $\mathbf{P}(p) = (P_1(p), P_2(p), \ldots, P_b(p))$,

$$P_k(p) \quad = \quad \max_{q \in \mathcal{B}(p)} \{R_k(q)\},$$

where $\mathcal{B}$ is an adjacency relation.



The widest component among the proposed regions is the plate.

Pooling

The activations $R_k(p)$ related to an object of interest might also appear at nearby positions within and across images.

Max-pooling can aggregate them by transforming $\hat{R} = (D_I, \mathbf{R})$ into $\hat{P} = (D_P, \mathbf{P})$, $\mathbf{P}(p) = (P_1(p), P_2(p), \ldots, P_b(p))$,

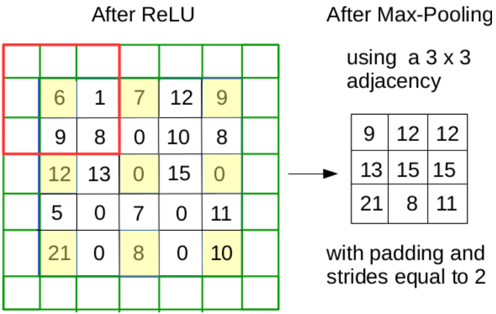
$$P_k(p) \quad = \quad \max_{q \in \mathcal{B}(p)} \{R_k(q)\},$$

where $\mathcal{B}$ is an adjacency relation.

The widest component among the proposed regions is the plate.

Pooling with stride

It is also common to apply padding and down-sampling on the input image with displacements $s_x \geq 1$ and $s_y \geq 1$, called strides.



For a $w \times h$ rectangular adjacency and image domain D_I with $n_x \times n_y$ pixels, the image domain D_P will have $\lfloor \frac{2n_x - w}{2s_x} \rfloor \times \lfloor \frac{2n_y - h}{2s_y} \rfloor$ pixels without padding and $\lceil \frac{n_x}{s_x} \rceil \times \lceil \frac{n_y}{s_y} \rceil$ pixels with padding.

Pooling

Other examples that create $\hat{P} = (D_I, \mathbf{P})$ by pooling are min-pooling and average pooling.

- Min-pooling:

$$P_k(p) \quad = \quad \min_{q \in \mathcal{B}(p)} \{R_k(q)\}.$$

- Average pooling:

$$P_k(p) \quad = \quad \frac{1}{|\mathcal{B}(p)|} \sum_{q \in \mathcal{B}(p)} R_k(q).$$

Indeed, any other image filtering could be used here to eliminate undesirable features and/or aggregate the desirable ones for better image analysis.

Normalization

Normalizations may be applied to any image $\hat{I} = (D_I, \mathbf{I})$ or to a batch $\mathcal{I} = \{\hat{I}_j\}_{j=1}^{B}$ with B images.

They are important to avoid discrepancies among local features along the network.

They create a new image $\hat{N} = (D_I, \mathbf{N})$ or a new batch $\mathcal{N} = \{\hat{N}_j\}_{j=1}^{B}$ with the same number of channels per image.

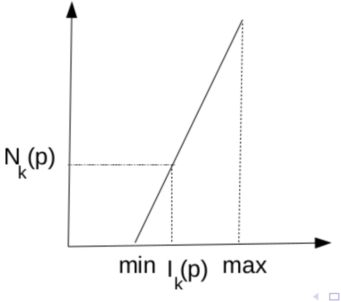
Linear normalization

For $\hat{N} = (D_I, \mathbf{N})$, $\mathbf{N}(p) = (N_1(p), N_2(p), \ldots, N_m(p))$,

$$N_k(p) = \frac{I_k(p) - \min_{q \in D_I}\{I_k(q)\}}{\max_{q \in D_I}\{I_k(q)\} - \min_{q \in D_I}\{I_k(q)\}},$$

$$N_k(p) = \frac{I_k(p) - \min_{j=1}^{B}\{I_{j,k}(p)\}}{\max_{j=1}^{B}\{I_{j,k}(p)\} - \min_{j=1}^{B}\{I_{j,k}(p)\}},$$

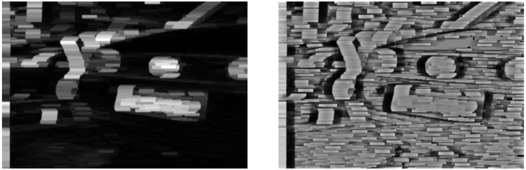
$k \in [1, m]$ and $p \in D_I$, we have a linear normalization.



Divisive normalization

Divisive normalization can enhance subtle and isolated activations within an adjacency $\mathcal{C}$, creating $\hat{N} = (D_I, \mathbf{N})$, with $\mathbf{N}(p) = (N_1(p), N_2(p), \ldots, N_m(p))$. For $k \in [1, m]$ and $p \in D_I$,

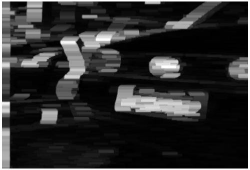
$$N_k(p) \quad = \quad \frac{I_k(p)}{\sqrt{\sum_{q \in \mathcal{C}(p)} I_k^2(q)}}.$$

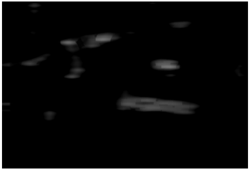


Images $\hat{P}$ (left) and $\hat{N}$ (right) – divisive normalization of $\hat{P}$ using $\mathcal{C}$ with $w = 25$ and $h = 5$.

Divisive normalization

To be useful, we are interested in reducing spurius regions and enhancing the plate by applying ReLU activation on the residue $\hat{P} - \hat{N}$ (right).





When thresholding the image on the right, it should facilitate the detection of the plate.

Batch normalization

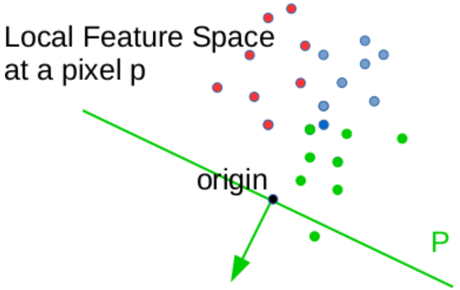
Batch normalization is very useful to standardize local features and eliminate the need of bias learning by creating an image $\hat{N} = (D_I, \mathbf{N})$, $\mathbf{N}(p) = (N_1(p), N_2(p), \ldots, N_m(p))$,

$$
\begin{aligned}
N_k(p) &= \frac{I_k(p) - \mu_k(p)}{\sigma_k(p) + \epsilon}\gamma_k + \beta_k, \\
\mu_k(p) &= \frac{1}{n}\sum_{j=1}^{n} I_{j,k}(p), \\
\sigma_k^2(p) &= \frac{1}{n-1}\sum_{j=1}^{n}(I_k(p) - \mu_k(p))^2,
\end{aligned}
$$

for $k \in [1, m]$, $p \in D_I$, $\epsilon = 10^{-5}$, and $\gamma_k, \beta_k \in \Re$ are parameters that can be learned and even undo the operation. We set $\gamma_k = 1$ and $\beta_k = 0$, for all $k \in [1, m]$, by default.

Batch normalization

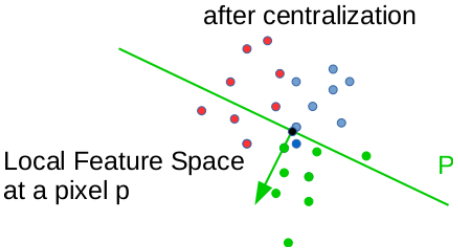
Batch normalization affects the local feature space with points $X_j(p)$ from an image set $\mathcal{I} = \{\hat{I}_j\}_{j=1}^{n}$ for all pixels $p \in D_I$.



Just the centralization of the point cloud already shows that training can adjust a kernel to select more features from a given class with no need of bias.

Batch normalization

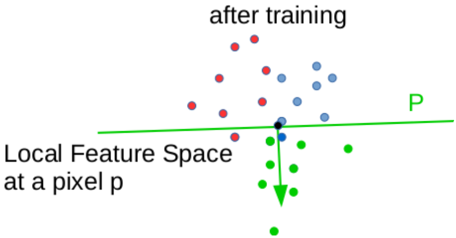
Batch normalization affects the local feature space with points $X_j(p)$ from an image set $\mathcal{I} = \{\hat{I}_j\}_{j=1}^{n}$ for all pixels $p \in D_I$.



Just the centralization of the point cloud already shows that training can adjust a kernel to select more features from a given class with no need of bias.

Batch normalization

Batch normalization affects the local feature space with points $X_j(p)$ from an image set $\mathcal{I} = \{\hat{I}_j\}_{j=1}^{n}$ for all pixels $p \in D_I$.



Just the centralization of the point cloud already shows that training can adjust a kernel to select more features from a given class with no need of bias.

Applications to image analysis

- By concatenating all channels I_j, $j = 1, 2, \ldots, m$, of an image (flattening), one creates a global feature vector $\mathbf{x}$ as image representation.

- By concatenating all channels I_j, $j = 1, 2, \ldots, m$, of an image (flattening), one creates a global feature vector $\mathbf{x}$ as image representation.

- Unfortunately, such representation is not usually suitable for image classification, as it was the case for the fashion mnist dataset.

Applications to image analysis

- By concatenating all channels I_j, $j = 1, 2, \ldots, m$, of an image (flattening), one creates a global feature vector $\mathbf{x}$ as image representation.

- Unfortunately, such representation is not usually suitable for image classification, as it was the case for the fashion mnist dataset.

- In the next lecture, we will see how Convolutional Neural Networks (CNNs) use sequence of layers containing convolution, activation, pooling and normalization to create image feature spaces suitable for MLP classifiers.

Applications to image analysis

- By concatenating all channels I_j, $j = 1, 2, \ldots, m$, of an image (flattening), one creates a global feature vector $\mathbf{x}$ as image representation.
- Unfortunately, such representation is not usually suitable for image classification, as it was the case for the fashion mnist dataset.
- In the next lecture, we will see how Convolutional Neural Networks (CNNs) use sequence of layers containing convolution, activation, pooling and normalization to create image feature spaces suitable for MLP classifiers.
- The applications go much beyond image classification – image synthesis, object detection, semantic/instance segmentation.