Convolutional Neural Network (CNN) workflow

Network Arch. O/P dim.

0-Input : N, 50x50 (batch size, input image)

1-Conv2D : 64, 3x3, 48x48, Relu

2-Conv2D : 32, 3x3, 46x46, Relu

3-MaxPool2D : 32, 3x3, 15x15

4-Conv2D : 16, 3x3, 13x13, Relu

5-MaxPool2D : 16, 3x3, 4x4

6-Flatten : 16 x 4x4 =256

7-FC : 128, Relu

8-FC : 64, Relu

9-FC : 6, Softmax

Forward Prop.

Step1:

C represents conv op.; I represents image (50x50 single greyscale image);

$k^1_{1,p}$ – superscript means the hierarchal order of operation, and p represents no. of filters;

1,p- represents 1 input channel, p channels

b^1_p – bias corresponding to each of p kernels

* Represents the convolution (element wise product n addition

f represents Relu activation

$$C^1_p = f(I * k^1_{1,p} + b^1_p); \text{ p=1 to 64}$$

Step2:

$$C^2_q = f(\sum_p C^1_p * k^2_{p,q} + b^2_q); \text{ p=1 to 64; q= 1 to 32}$$

Step3:

$$M^1_q(i,j) = \max\big(C^2_q [\, 3(\text{i} - 1) + 1 \,:\, 3(\text{i} - 1) + 3 \,, 3(\text{j} - 1) + 1 \,:\, 3(\text{j} - 1) + 3 \,]\big)$$

$$\text{i, j=1 to 15; q= 1 to 32;}$$

Step4:

$$C^3_r = f(\sum_q M^1_q * k^3_{q,r} + b^3_r); \text{ q= 1 to 32; r = 1 to 16}$$

Step5:

$$M_r^2(i,j) = \max(\, C_r^3[\, 3(\mathrm{i}-1)+1 \,:\, 3(\mathrm{i}-1)+3\,, 3(\mathrm{j}-1)+1 \,:\, 3(\mathrm{j}-1)+3\,])$$

i, j=1 to 4; r= 1 to 16;

Step6:

$$g \;= G(\,\{M_r^2\}\,; \mathrm{r} = 1 \text{ to } 4\;);\quad \text{(flatten/vectorization)}$$

Rest of the steps for the fully connected layers are same as in Feedforward Multi-Layer Perceptron (MLP).

Backpropagation Path:

Let L be the loss at the end of the epoch during the training phase.

All the equations for backpropagation for the fully connected layers are the same as in MLP till the FC-7 layer.

Step1:

Those computed gradients from layer FC-7 simply need to be vectorized into matrix 4x4x16

Reshape output error vector Δg (256x1) into (4x4x16)

$$\{\Delta M_r^2\}_{\mathrm{r=1\ to\ 4}} = G^{-1}(\,\Delta g\,);\quad \text{(flatten/vectorization)}$$

Since it is a max-pool op., it doesn't involve any parameters, the stored mask in forward pass is used as ref. for prop. Updates into respective paths, giving ΔC_r^3.

Step2: (Inverse of step 4 in forward path)

Calc. $\Delta k_{q,r}^3$:

$$\Delta k_{q,r}^3(u,v) = \frac{\partial L}{\partial k_{q,r}^3(u,v)} = \sum_i^{13}\sum_j^{13} \frac{\partial L}{\partial C_r^3(i,j)}\,\frac{\partial C_r^3(i,j)}{\partial k_{q,r}^3(u,v)}$$

$$= \sum_i^{13}\sum_j^{13} \Delta C_r^3(i,j)\,.\frac{\partial C_r^3(i,j)}{\partial t}\,\frac{\partial \mathrm{t}}{\partial k_{q,r}^3(u,v)}; t = (\sum_q M_q^1 * k_{q,r}^3 \; + b_r^3)$$

$$= \sum_i^{13}\sum_j^{13} \Delta C_r^3(i,j)\,.f'.\;\frac{\partial\left(\sum_q M_q^1 * k_{q,r}^3 \; + b_r^3\right)}{\partial k_{q,r}^3(u,v)}$$

$$= \sum_i^{13}\sum_j^{13} \Delta C_r^3(i,j)\,.f'.\; M_q^1\,(i-u, j-v)$$

$$\text{let } \Delta C^3_{r,f}(i,j) = \Delta C^3_r(i,j) \,.\, f'$$

hence what we get is basically a deconvolution, hence:

$$\Delta k^3_{q,r}(u,v) = \sum_i^{13}\sum_j^{13} M^1_q(i-u, j-v) \,.\, \Delta C^3_{r,f}(i,j)$$

Here, $M^{1\,inv}_q$ which is eqv. To convolving with inverse of M^1_q i.e. rotating M^1_q by 180º (flipped)

$$\Delta k^3_{q,r} = M^{1\,inv}_q * \Delta C^3_{r,f}$$

Calculating ΔM^1_q:

Similarly, Following the above process for ΔM^1_q, we get similar deconvolution:

$$C^3_r = f(\textstyle\sum_q M^1_q * k^3_{q,r} + b^3_r);$$

$$\Delta M^1_q(i,j) = \frac{\partial L}{\partial M^1_q(i,j)}$$

$$\Delta M^1_q = \sum_r^{15} \Delta C^3_{r,f} * k^{3\,inv}_{q,r}$$

Calculating Δb^3_r:

$$\Delta b^3_r = \frac{\partial L}{\partial b^3_r} = \sum_i^{13}\sum_j^{13} \frac{\partial L}{\partial C^3_r(i,j)} \frac{\partial C^3_r(i,j)}{\partial b^3_r}$$

$$= \sum_i^{13}\sum_j^{13} \Delta C^3_r(i,j) \,.\, \frac{\partial C^3_r(i,j)}{\partial t} \frac{\partial \mathrm{t}}{\partial b^3_r}; t = (\sum_q M^1_q * k^3_{q,r} + b^3_r)$$

$$= \sum_i^{13}\sum_j^{13} \Delta C^3_r(i,j) \,.\, f' .\, \frac{\partial\left(\sum_q M^1_q * k^3_{q,r} + b^3_r\right)}{\partial b^3_r}$$

$$= \sum_i^{13}\sum_j^{13} \Delta C^3_r(i,j) \,.\, f'$$

$$\Delta b^3_r = \sum_i^{13}\sum_j^{13} \Delta C^3_{r,f}(i,j)$$

Step3: (Inverse of step 3 in forward path)

Since it is a max-pool operation, it doesn't involve any parameters, the stored mask in forward pass is used as ref. for prop. Updates from ΔM^1_q will be propagated into respective ΔC^2_q.

Step4: (Inverse of step 2 in forward path)

Calculating $\Delta k_{p,q}^{2}$:

$$\Delta k_{p,q}^{2}(u,v) = \frac{\partial L}{\partial k_{p,q}^{2}(u,v)} = \sum_{i}^{46}\sum_{j}^{46} \frac{\partial L}{\partial C_{q}^{2}(i,j)} \frac{\partial C_{q}^{2}(i,j)}{\partial k_{p,q}^{2}(u,v)}$$

$$= \sum_{i}^{46}\sum_{j}^{46} \Delta C_{q}^{2}(i,j) . \frac{\partial C_{q}^{2}(i,j)}{\partial t} \frac{\partial \mathrm{t}}{\partial k_{p,q}^{2}(u,v)}; t = (\sum_{p} C_{p}^{1} * k_{p,q}^{2} + b_{q}^{2})$$

$$= \sum_{i}^{46}\sum_{j}^{46} \Delta C_{q}^{2}(i,j) . f' . \frac{\partial \left(\sum_{p} C_{p}^{1} * k_{p,q}^{2} + b_{q}^{2}\right)}{\partial k_{p,q}^{2}(u,v)}$$

Equivalently,

$$\Delta k_{p,q}^{2} = C_{p}^{1\,inv} * \Delta C_{q,f}^{2}$$

Calculating ΔC_{p}^{1}:

Similarly, Following the above process for ΔM_{q}^{1}, we get similar deconvolution:

$$\Delta C_{p}^{1}(i,j) = \frac{\partial L}{\partial C_{p}^{1}(i,j)}$$

$$\Delta C_{q}^{1} = \sum_{q}^{48} \Delta C_{q,f}^{2} * k_{p,q}^{2\,inv}$$

Calculating Δb_{q}^{2}:

$$\Delta b_{q}^{2} = \frac{\partial L}{\partial b_{q}^{2}} = \sum_{i}^{46}\sum_{j}^{46} \frac{\partial L}{\partial C_{q}^{2}(i,j)} \frac{\partial C_{q}^{2}(i,j)}{\partial b_{q}^{2}}$$

$$\Delta b_{q}^{2} = \sum_{i}^{46}\sum_{j}^{46} \Delta C_{q,f}^{2}(i,j)$$

Step5: (inv. Of step 1 in fwd path)

Calc. $\Delta k_{1,p}^{1}$:

$$\Delta k_{1,p}^{1}(u,v) = \frac{\partial L}{\partial k_{1,p}^{1}(u,v)} = \sum_{i}^{50}\sum_{j}^{50} \frac{\partial L}{\partial C_{p}^{1}(i,j)} \frac{\partial C_{p}^{1}(i,j)}{\partial k_{1,p}^{1}(u,v)}$$

Equivalently,

$$\Delta k_{1,p}^{1} = \mathrm{I}^{inv} * \Delta C_{p,f}^{1}$$

Calc. Δb_{p}^{1}:

$$\Delta b_p^1 = \frac{\partial L}{\partial b_p^1} = \sum_i^{50} \sum_j^{50} \frac{\partial L}{\partial C_p^1(i,j)} \frac{\partial C_p^1(i,j)}{\partial b_p^1}$$

$$\Delta b_p^1 = \sum_i^{50} \sum_j^{50} \Delta C_{p,f}^1(i,j)$$

Gaussian Noise

Probability density for the Gaussian distribution is:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

$\mu\ is\ the\ mean$

$\sigma\ is\ \ the\ standard\ deviation\ and\ \sigma^2$is the variance

p(x) gives the gaussian distribution.

$$\tilde{z} = z + n$$

z is the noise free image

n is the noise with gaussian distribution with desired variance (noise)

Traditional denoising methods

Gaussian filtering:

Gaussian filter is used for noise removal. The source image is convolved with selected Gaussian filter for noise removal.

The Gaussian distribution in 1-D has the form:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-x^2}{2\sigma^2}}$$

where σ is the standard deviation of the distribution. We have also assumed that the distribution has a mean of zero

> **Library used** – OpenCV (cv2)

> **Parameters used** – kernel size – 3*3 (standard)

> **Link to library-** More information available on this link

https://docs.opencv.org/master/d4/d86/group__imgproc__filter.html#gaabe8c836e97159a9193fb0b11ac52cf1

Average filtering:

Average (or mean) filtering is a method of 'smoothing' images by reducing the amount of intensity variation between neighbouring pixels. The average filter works by moving through the image pixel by pixel, replacing each value with the average value of neighbouring pixels, including itself.

The function smooths an image using the kernel:

$$K = \frac{1}{kernel\ heigth\ \times \text{kernerl width}} \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$$

> **Library used** – OpenCV (cv2)

> **Parameters used** – kernel size – 3*3

> **Link to library-** More information available on this link-

https://docs.opencv.org/master/d4/d86/group__imgproc__filter.html#ga8c45db9afe636703801b0b2e440fce37

Median filtering:

Median filtering is a nonlinear method used to remove noise from images. It is widely used as it is very effective at removing noise while preserving edges. It is particularly effective at removing 'salt and pepper' type noise. Image is smoothened using the median filter with the kernel size × kernel size aperture. Each channel of a multi-channel image is processed independently.

> **Library used** – OpenCV (cv2)

> **Parameters used** – kernel size – 3*3

> **Link to library-** More information available on this link-

https://docs.opencv.org/master/d4/d86/group__imgproc__filter.html#ga564869aa33e58769b4469101aac458f9

Bilateral filtering:

The basic idea underlying bilateral filtering is to do in the range of an image what traditional filters do in its domain. Two pixels can be *close* to one another, that is, occupy nearby spatial location, or they can be *similar* to one another, that is, have nearby values, possibly in a perceptually meaningful fashion. Bilateral Filter can reduce unwanted noise very well while keeping edges fairly sharp.

> **Library used** – OpenCV (cv2)

> **Parameters used** – kernel size =3, sigmaColor=3, sigmaSpace=3

> **Link to library-** More information available on this link

https://docs.opencv.org/master/d4/d86/group__imgproc__filter.html#ga9d7064d478c95d60003cf839430737ed

bm3d filtering:

Bm3d is based on an enhanced sparse representation in transform-domain. The enhancement of the sparsity is achieved by grouping similar 2D image fragments (e.g. blocks) into 3D data arrays which we call "groups".

Image fragments are grouped together based on similarity, but unlike standard k-means clustering and such cluster analysis methods, the image fragments are not necessarily disjoint. This block-matching algorithm is less computationally demanding

> **Library used** – bm3d

> **Parameters used** – sigma_psd= 15

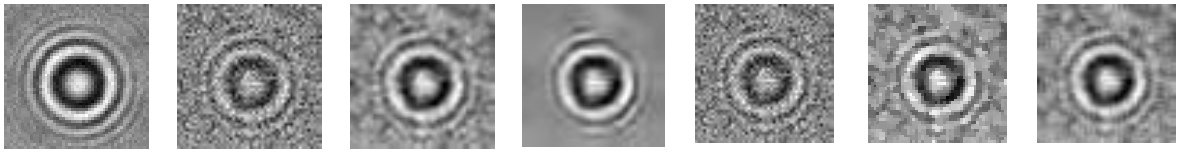
> **Link to library**- More information available on these links-

https://pypi.org/project/bm3d/

http://www.cs.tut.fi/~foi/GCF-BM3D/

Comparison:

RBC



$20\mu m$

Original	Noisy	Gaussian filtering output	Bm3d filtering output	Bilateral filtering output	Average filtering output	Median filtering output

Reference:

1. K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising with block-matching and 3D filtering," *Proc. SPIE Electronic Imaging '06*, no. 6064A-30, San Jose, California, USA, January 2006.
2. K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3D transform-domain collaborative filtering," IEEE Trans. Image Process., vol. 16, no. 8, pp. 2080-2095, August 2007.