# 3RCNN- ALGORITHM DERIVATION

**Vikas Sarvasya**

**30 May 2023 to 8 June 2023**

# Contents

# 1 Loss to Filter Calculation

**Theorem 1.1.** Each node is a filter matrix.

**Theorem 1.2.** Each image is treated as a matrix of grayscale intensities.

**Theorem 1.3.** Convolutions are performed with same-padding methods to keep matrix size constant.

**Theorem 1.4.** Conv2D layers take in image matrices and output feature maps, which could also be considered image matrices. This allows for multiple Conv2D layers to be joined together.

## 1.1 Individual Filter Adjustment

Call the input image matrix $I$ and define it to be

$$I = \begin{bmatrix} I_{1,1} & I_{1,2} & ... & I_{1,23} & I_{1,24} \\ I_{2,1} & I_{2,2} & ... & I_{2,23} & I_{2,24} \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ I_{23,1} & I_{23,2} & ... & I_{23,23} & I_{23,24} \\ I_{24,1} & I_{24,2} & ... & I_{24,23} & I_{24,24} \end{bmatrix} \tag{1}$$

For our given layer, define 16 filters, with one such filter

$$F = \begin{bmatrix} F_{1,1} & F_{1,2} & F_{1,3} & F_{1,4} & F_{1,5} \\ F_{2,1} & F_{2,2} & F_{2,3} & F_{2,4} & F_{2,5} \\ F_{3,1} & F_{3,2} & F_{3,3} & F_{3,4} & F_{3,5} \\ F_{4,1} & F_{4,2} & F_{4,3} & F_{4,4} & F_{4,5} \\ F_{5,1} & F_{5,2} & F_{5,3} & F_{5,4} & F_{5,5} \end{bmatrix} \tag{2}$$

Then, we add two layers of same-padding to the outside of the image matrix before applying the convolution operation to get:

$$O = Conv(I, F) \tag{3}$$

where $O$ represents a feature map.
We have

$$O_{i,j} = \Sigma F_{k,l} * I_{(i+k-3),(j+l-3)} \quad for \quad k, l \in (1, 2, 3, 4, 5) \tag{4}$$

We then define $P = MaxPooling(O)$, so

$$P_{i,j} = max(O_{m,n}) \quad for \quad m \in (2i-1, 2i) \quad and \quad n \in (2j-1, 2j) \tag{5}$$

We then get to the resultant matrix, $R = AvgPooling(P)$. This is done by splitting $P$ into chunks with the same dimensions as the filter $5x5$ in this case. Since the initial image has dimensions $24x24$, we add a layer of same-padding at the bottom and the right to make it $25x25$ and then divide it into chunks.

All of the values in these chunks will be averaged to create a resultant matrix with the same dimensions as the filter. This can be represented as follows:

$$R_{i,j} = \frac{(\Sigma P_{o,p})}{r_F c_F} \tag{6}$$

for

$$o \in \left( \lceil \frac{r_P}{r_F} \rceil i - r_F + 1, \lceil \frac{r_P}{r_F} \rceil i \right) \tag{7}$$

and

$$p \in \left( \lceil \frac{c_P}{c_F} \rceil j - c_F + 1, \lceil \frac{c_P}{c_F} \rceil j \right) \tag{8}$$

where $r_F$ is the number of rows in the filter and $c_F$ is the number of columns in the filter.

We can depict $R$ as a single column of vectors, as shown in the equation below:

$$R = \begin{bmatrix} \overrightarrow{r_1} \\ \overrightarrow{r_2} \\ \overrightarrow{r_3} \\ \overrightarrow{r_4} \\ \overrightarrow{r_5} \end{bmatrix} \tag{9}$$

We then use the flatten function on $R$ to get

$$\overrightarrow{f} = flatten(R) = (\overrightarrow{r_1}, \overrightarrow{r_2}, \overrightarrow{r_3}, \overrightarrow{r_4}, \overrightarrow{r_5}) \tag{10}$$

We then apply the softmax function to $\overrightarrow{f}$.

The softmax function takes in a vector $\overrightarrow{z}$ and returns a vector of probabilities, as shown below

$$softmax(\overrightarrow{z}) = \frac{e^{z_i}}{\Sigma e^{z_j}} \quad for \quad j \in (1, 2, ..., K) \tag{11}$$

We define $S_{\overrightarrow{f}} = softmax(\overrightarrow{f}) - \frac{1}{r_F c_F}$ to normalize the distribution of probabilities to add to 0.

We can break $S_{\overrightarrow{f}}$ into its five component vectors again, making

$$S_{\overrightarrow{f}} = (\overrightarrow{S_1}, \overrightarrow{S_2}, \overrightarrow{S_3}, \overrightarrow{S_4}, \overrightarrow{S_5}) \tag{12}$$

We can use these vectors to create a matrix of softmax probabilities, as shown below:

$$S = \begin{bmatrix} \overrightarrow{S_1} \\ \overrightarrow{S_2} \\ \overrightarrow{S_3} \\ \overrightarrow{S_4} \\ \overrightarrow{S_5} \end{bmatrix} \tag{13}$$

We multiply $R$ and $S$ element-wise to get the $\Delta$ matrix:

$$\Delta_{i,j} = R_{i,j} S_{i,j} \tag{14}$$

We can use this to alter the filter individually:

$$F_{final} = F_{initial} - \alpha\Delta \tag{15}$$

where $\alpha$ is the hyperparameter learning rate of the algorithm.

## 1.2 Overall Filter Adjustment

While the previous filter adjustment method is probability based to keep nuance throughout the filter, this method adjusts the filter as a whole and is based on loss calculations.

First, we define the image loss matrix. Each image comes with a set of coordinates labeling $(x_i, y_i, x_f, y_f)$ to determine the bounding box. Every pixel on the image can be assigned four values, one corresponding to the mean-squared error results of the bounding box.

As a reminder, mean squared error is defined as

$$MSE = \frac{1}{n}\Sigma(Y_i - \hat{Y_i})^2 \quad for \quad i \in (1, 2, ...n) \tag{16}$$

For bounding box regression, we define the mean-squared error of a predicted bounding box to be

$$min\left((x_i - \hat{x_i})^2, (x_f - \hat{x_f})^2\right) + min\left((y_i - \hat{y_i})^2, (y_f - \hat{y_f})^2\right) \tag{17}$$

to normalize the loss distribution and make sure that a zero-loss situation is actually possible.

Using the above calculations, we can define the loss matrix $L$. For each element, we then find $\frac{\partial L}{\partial x}$ and $\frac{\partial L}{\partial y}$ using the tabular method on $L$. The top and left of these new matrices are padded with zeros, giving us matrices $\frac{\partial L}{\partial x}$ and $\frac{\partial L}{\partial y}$ with the same dimensions as the original image.

We then go layer by layer to apply the loss gradient to all the filters. As we remember from the forward pass, each filter of each layer makes its own feature map, $O = Conv(I, F)$.

We apply the tabular method on $O$ to get $\frac{\partial O}{\partial x}$ and $\frac{\partial O}{\partial y}$.

We then calculate $\frac{\partial L}{\partial O}$ as

$$\frac{\partial L}{\partial O} = \left(\frac{\partial L}{\partial x} \div \frac{\partial O}{\partial x}\right) + \left(\frac{\partial L}{\partial y} \div \frac{\partial O}{\partial y}\right) \tag{18}$$

Then, we calculate $\frac{\partial O}{\partial F}$.

The partial derivative of a matrix is a reverse of the convolution function, which functions as a partial integral by taking the sum of products in an area. So, we have

$$\frac{\partial O_{1,1}}{\partial F} = I_F = \begin{bmatrix} I_{1,1} & I_{1,2} & I_{1,3} & I_{1,4} & I_{1,5} \\ I_{2,1} & I_{2,2} & I_{2,3} & I_{2,4} & I_{2,5} \\ I_{3,1} & I_{3,2} & I_{3,3} & I_{3,4} & I_{3,5} \\ I_{4,1} & I_{4,2} & I_{4,3} & I_{4,4} & I_{4,5} \\ I_{5,1} & I_{5,2} & I_{5,3} & I_{5,4} & I_{5,5} \end{bmatrix} \tag{19}$$

We want to find $\Delta O_{1,1}$, so we do

$$\Delta O_{1,1} = \frac{\partial O_{1,1}}{\partial F} * F = \Sigma(I_F \cdot F) \tag{20}$$

As shown in the equation above, we take the dot product of the initial image segment and the filter before summing all of the elements to get $\Delta O_{1,1}$.

Then, we can make the new $\frac{\partial O}{\partial F}$ matrix, as shown below.

$$\left(\frac{\partial O}{\partial F}\right)_{i,j} = \Delta O_{i,j} \tag{21}$$

Then, we can make

$$\frac{\partial L}{\partial F} = \frac{\partial L}{\partial O} \cdot \frac{\partial O}{\partial F} \tag{22}$$

With this loss gradient in terms of the filter, we alter the filter values as shown:

$$F_{final} = F_{initial} - \alpha \frac{\partial L}{\partial F} \tag{23}$$

# 2 Expansion to 3D

## 2.1 Individual Filter Adjustment- 3D

The only difference in this section is that when $R$ is flattened, it has not only a univariate vector for each row of that filter, but for each row of the filters above it. This means that this flatten function would work on 25 vectors and 125 elements (since there are 5 layers) instead of 5 vectors and 25 elements.

This is done to adjust the softmax values and to ensure that modification of the filters is controlled and evenly spread out. After, the $S_{\vec{f}}$ vector is recreated with all 25 sub-vectors and then redistributed to their respective filters, with the $\Delta$ matrix being created the same way as the original algorithm.

## 2.2 Overall Filter Adjustment- 3D

The main difference for this section is that there are now five different loss matrices, one for each layer. This means that wildly different $\frac{\partial L}{\partial O}$'s can be created, allowing for more variety and a fine-tuning of the filters to their respective image. This introduces the idea of precedence, which is that the filters need to be applied in a certain order, since they are applied on different images but produce the same features.

There are no inbuilt algorithmic differences for this section.

# 3 Image Set Techniques

## 3.1 Events with States

Events with States is a mathematical technique used to determine how sets affect each other.

This technique deals with the entire image as a singular set for each layer. Since we have five rnlayers, we can define this as $Images = (A, B, C, D, E)$.

As we move from set to set, we calculate a coefficient of similarity, $s$.

We can calculate $s_{A,B}$ with the method shown below.

$$s_{A,B} = \sum \left( ELU \left( \frac{ln\left(\frac{A_{i,j}+B_{i,j}}{2}\right)}{ln(255)} - \sqrt{\left(\frac{abs(A_{i,j} - B_{i,j})}{2^8}\right)} \right) \right) \div ij \tag{24}$$

In this equation, we first take the average of the two values of a certain pixel and take the natural logarithm. We then divide by the natural logarithm of 255 to normalize these values to operate in a range from 0 to 1. This is used to give higher precedence to brighter areas of the image. This is important since many images have similar dark areas, but it is more important to depict similar features. Especially in CT scans, this also helps find the ratios of white and gray areas in each image for segmentation purposes.

The second part is where differing pixel values are subtracted. We take the absolute difference between the two values and divide by $2^8$ to normalize these values down to a range between 0 and 1. We then take the square root to ensure that higher differences are emphasized more.

Lastly, we apply the ELU function. ELU is a variant of the ReLU (Rectified Linear Unit) function, which is as follows:

$$\begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$

ReLU is often used as an activation function, and it is used to ensure that negative values are discarded.

However, we do want to include them to make sure that highly variant images have a low coefficient while still ensuring that they do not sway the calculation too much.

So, we use the Exponential Linear Unit (ELU), shown below.

$$\begin{cases} e^x - 1 & x \leq 0 \\ x & x > 0 \end{cases}$$

Once applying this function, we sum the values over all of the pixels in the image and divide by the number of pixels to get the coefficient of similarity.

This is where the recurrent element of the network comes in. For the forward pass of the network, instead of just the resultant feature vector from the previous node being passed, a compilation of all of the similarity coefficients is provided.

For example, for $A$, the forward pass is as shown:

$$\frac{R_A + R_B s_{A,B} + R_C s_{A,C} + R_D s_{A,D} + R_E s_{A,E}}{\sum s_{A,v}} \tag{25}$$

### 3.2 DNN Correction of Image Distortion

With chest CTs, the heart area is often extremely bright and cannot be seen separately.

To correct this issue, we first train a DNN to recognize only the heart using a separate bounding box algorithm for identifying circles.

The backpropagation algorithm for this network would be the same as described in the Overall Filter Adjustment 3D section, but instead of applying the loss matrix to the bounding box points, it would be infinitely summed over the circle's circumference as an integral.

Once this subset has been acquired, it would be selectively dimmed through the following algorithm:

$$J = I - (min(i, j) > 50) \tag{26}$$

$$K = J * \frac{255}{max(X) - 25 - (min(i, j) > 50)} \tag{27}$$

$$L = Conv\left(K, \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}\right) \tag{28}$$

$$M = K - Dilation(Dilation(L)) \tag{29}$$

$$N = ZeroPadding(M, min((h \pm r)^2 - i^2 + (k \pm r)^2 - j^2)) \tag{30}$$

With this new image $N$, which has been transformed into a square from circle $M$, we feed it into the 3D RCNN detailed in the previous section.

While this dimming process is enough to process this image, it messes up the calculation of CAC scores with the calcium intensity values.

To rectify this, we do the following procedure.

$$O = \left((N - 170) * \frac{70}{avg(N_{i,j} > 215) - 170}\right) + 170 \tag{31}$$

We then use the same scoring procedure that was used for CAC scoring of gated cardiac CTs, which is

$$CAC = \Sigma\,(1 \quad if \quad Z_{i,j} > 215) + \Sigma\,(1 \quad if \quad Z_{i,j} > 240) \tag{32}$$

### 3.3 Saving Filters

Usually, in CNNs, only the last filter is used to determine features. All of the previous filters are used simply to modify the image and pass it on. By keeping a time-record of all the filters/nodes, we can store the final value of each filter.

Instead of building a final feature vector using only the last set, we apply each and every filter to the image separately and compile the results in a much larger multivariant vector and pass that on to the last Dense layer.

### 3.4 Multiple Resets

Since the backpropagation is a form of abstract gradient descent, it will only find a local minimum for loss. To find a possibly better value and a global minimum, we use a modified randomization method.

At this point, we would have already run a full training session on the given initial filter values. We know that most of the points in that area would converge to the same local minimum, so we want to find a randomized set far away from this set.

So, we close off options to choose filters between $0.5F$ and $2F$ from the initial filter to force the new filter values away from what they were before.

We will continue doing this if difference criteria are met. There are two options, as listed below.

- Option 1: Loss decreases by at least 5%
- Option 2: Loss increases by at least 10% and less than 100%

The first condition is fairly straightforward: if a non-trivial improvement is found, the algorithm keeps randomizing to find new values.

The second condition allows for the loss to increase within a certain range. If the loss increases too much, then obviously the algorithm should stop since it is not improving. However, this condition also requires a minimum loss change to ensure that the algorithm is not static and returning to similar loss values.

### 3.5 Plane Recurrence

While we have mostly focused on the passing forward of information between network layers, it is important to remember that there are multiple nodes in each of the five sections. Even though each node is a filter between nodes by calculating the determinant of the filter matrix.

We can ensure that the nodes on different layers are connected with a procedure using the determinants, as shown below.

$$F_{l,adjusted} = \frac{F_l}{||F_l||}\left(\sum_{k=1}^{5}(-1)^{l-k}\frac{||F_k||}{2^{l-k}}\right) \tag{33}$$