Introduction To CNN

In neural networks, Convolutional neural networks (ConvNets or CNNs) is one of the main categories to do image recognition, images classifications. Objects detections, recognition faces etc., are some of the areas where CNNs are widely used. CNN image classifications takes an input image, process it and classify it under certain categories (Eg., Dog, Cat, Tiger, Lion). Computers see an input image as an array of pixels and it depends on the image resolution. Based on the image resolution, it will see h x w x d(h = Height, w = Width, d = Dimension). Eg., An image of 6 x 6 x 3 array of matrix of RGB (3 refers to RGB values) and an image of 4 x 4 x 1 array of matrix of grayscale image.

Technically, deep learning CNN models to train and test, each input image will pass it through a series of convolution layers with filters (Kernals), Pooling, fully connected layers (FC) and apply Softmax function to classify an object with probabilistic values between 0 and 1. The below figure is a complete flow of CNN to process an input image and classifies the objects based on values.

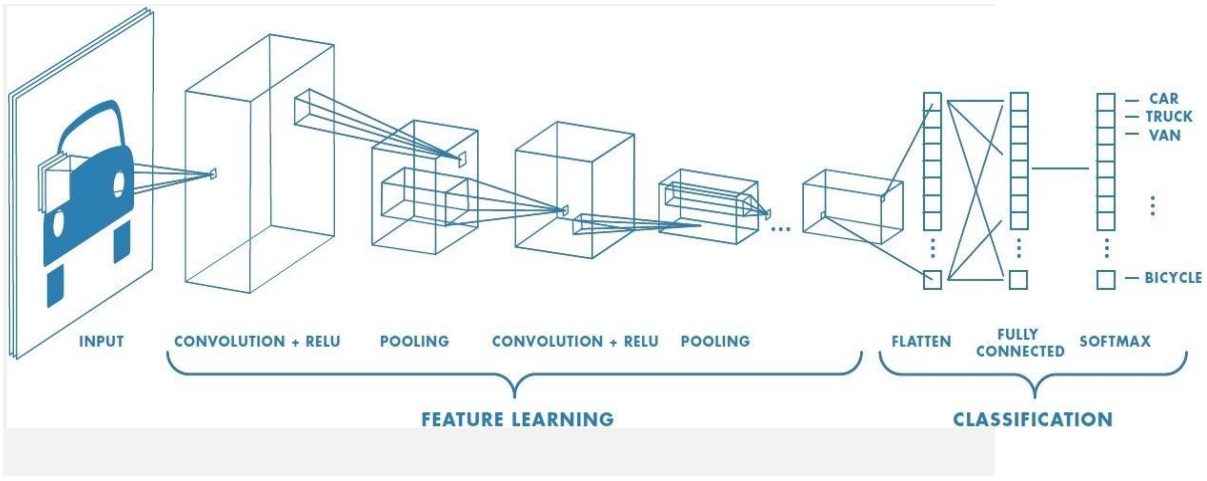


Figure 2 : Neural network with many convolutional layers

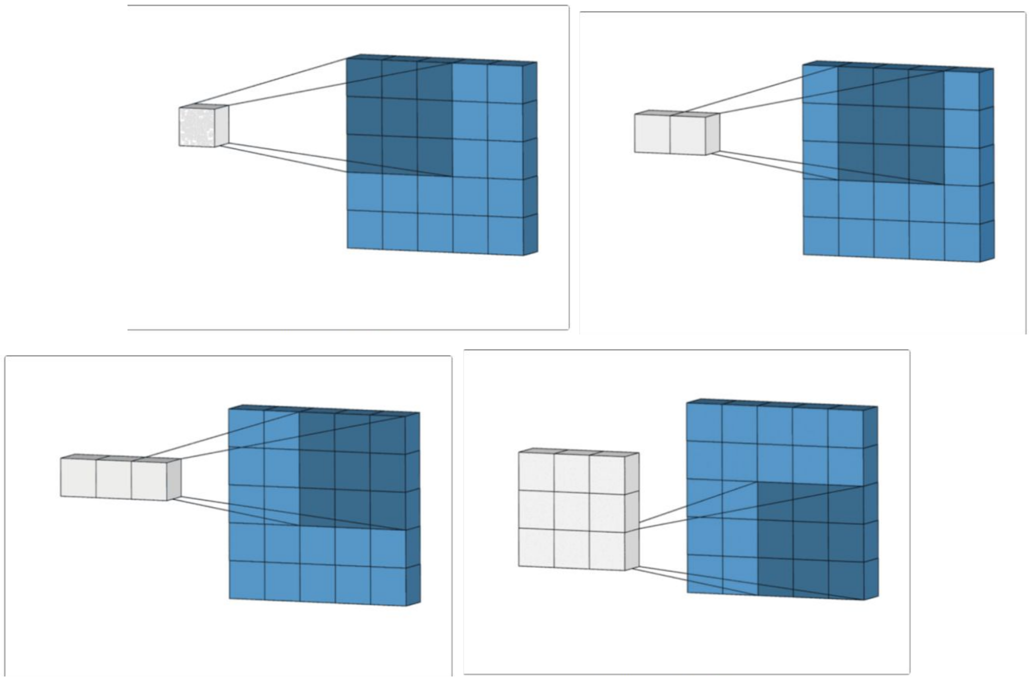
Understanding How CNN Works

1.Convolutional Layer(Feature Learning in Figure 2)

The convolution layer is the core building block of CNN. It carries the main portion of the network's computational load.

This layer performs a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a kernel, and the other matrix is the restricted portion of the receptive field. The kernel is spatially smaller than an image but is more in-depth. This means that, if the image is composed of three

(RGB) channels, the kernel height and width will be spatially small, but the depth extends up to all three channels.



During the forward pass, the kernel slides across the height and width of the image-producing image representation of that receptive region. This produces a two-dimensional representation of the image known as an activation map that gives the response of the kernel at each spatial position of the image. The sliding size of the kernel is called a stride.

If we have an input of size W x W x D and Dout number of kernels with a spatial size of F with stride S and amount of padding P, then the size of output volume can be determined by the following formula:

$$W_{out} = \frac{W - F + 2P}{S} + 1$$

Formula for Convolution Layer

This will yield an output volume of size *Wout* x *Wou*t x *Dout*.

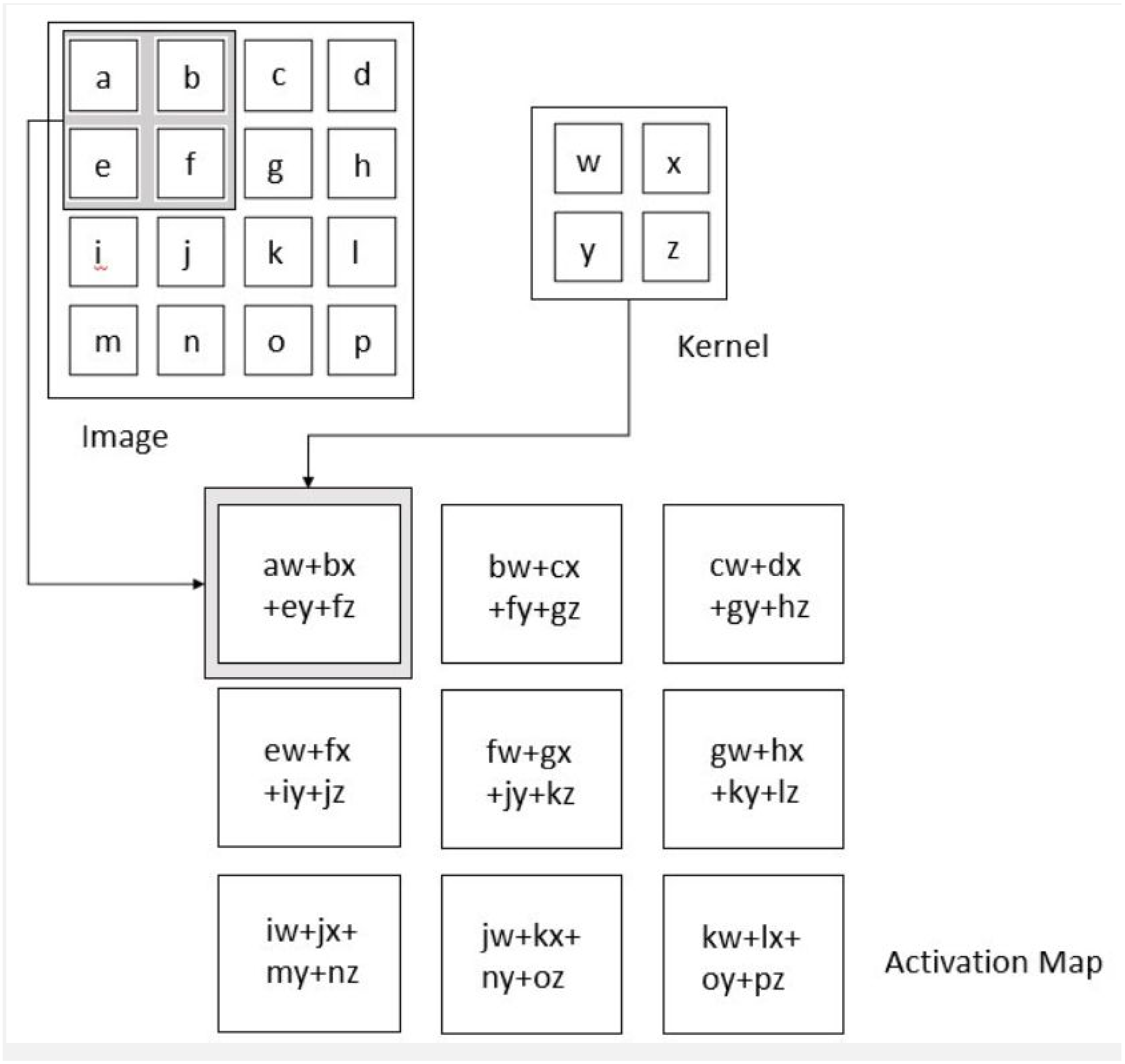


Figure 3: Convolution Operation (Source: Deep Learning by Ian Goodfellow, Yoshua Bengio, and Aaron Courville)

Making the kernel smaller than the input e.g., an image can have millions or thousands of pixels, but while processing it using the kernel we can detect meaningful information that is of tens or hundreds of pixels. This means that we need to store fewer parameters that not only reduces the memory requirement of the model but also improves the statistical efficiency of the model.

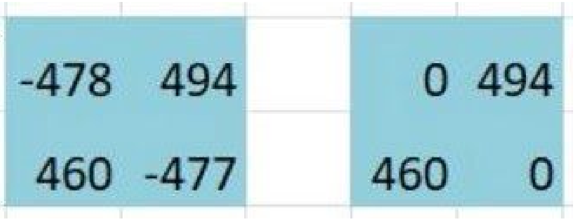
The weight values within filters are learnable during the training phase of a CNN. The output dimension of the convolutional layer has a depth component, if we partition each segment of the output we will obtain a 2D plane of a feature map.

The filter used on a single 2D plane contains a weight that is shared across all filters used across the same plane. The advantage of this is that we maintain the same feature detector used in one part of the input data across other sections of the input data.The advantage of this is that we maintain the same feature detector used in one part of the input data across other sections of the input data.

The output of a convolutional layer is a set of feature maps, where each feature map is the result of a convolution operation between the fixed weight parameters within the unit and the input data. One of the essential characteristics of the convolutional neural network layer is its ability for the feature map to reflect any affine transformations that are made to the input image that is fed through the input layer.

2.Activation Layer

$$RELU(x) = \begin{cases} 0 \; if \;\; x < 0 \\ x \; if \;\; x >= 0 \end{cases}$$



-478	494
460	-477

0	494
460	0

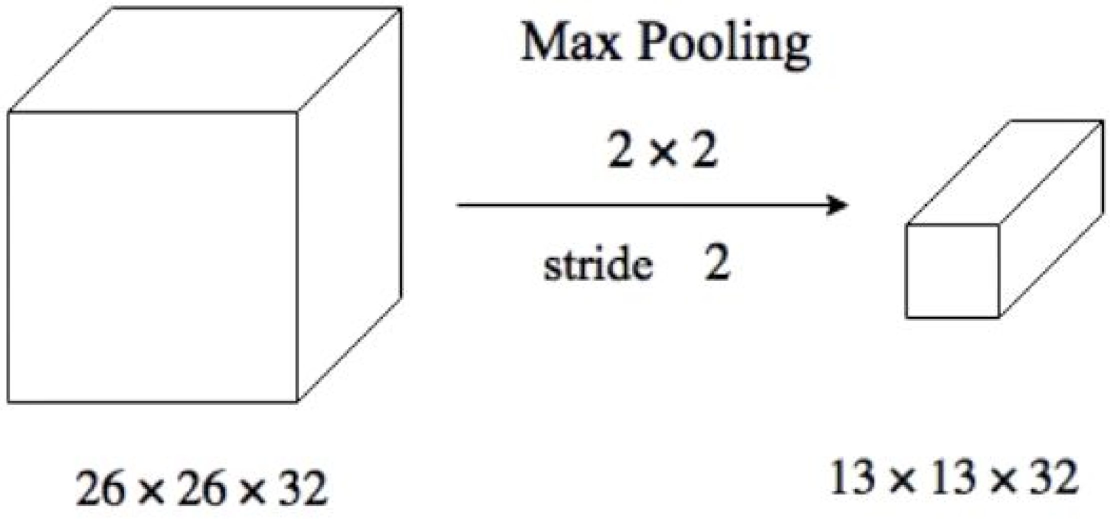
In this step we apply the rectifier function to increase non-linearity in the CNN. Images are made of different objects that are not linear to each other.Relu function converts all negative numbers in the feature maps to 0.

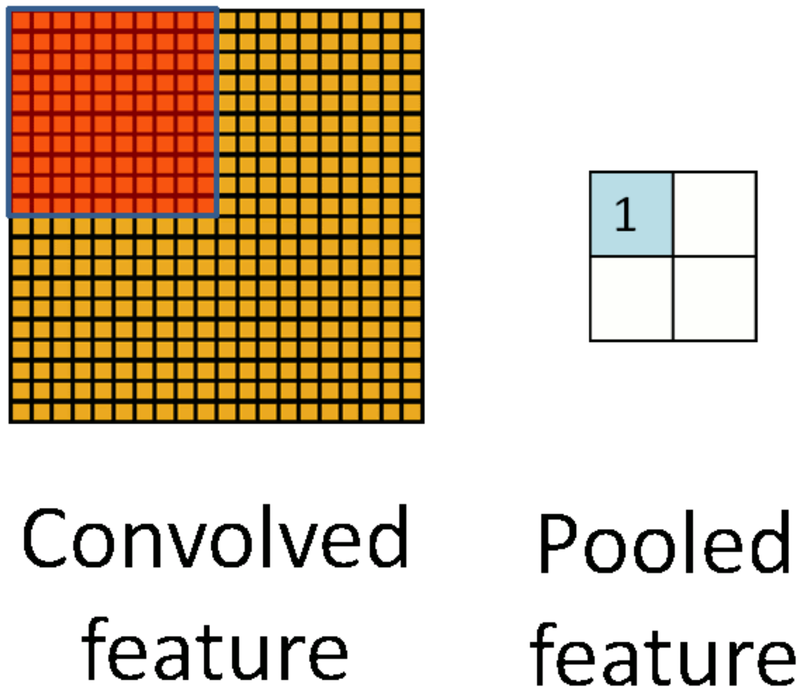
3.Pooling Layer

Third, is the POOLING LAYER, which involves downsampling of features. It is applied through every layer in the 3d volume. Typically there are hyperparameters within this layer:

The dimension of spatial extent: which is the value of n which we can take N cross and feature representation and map to a single value

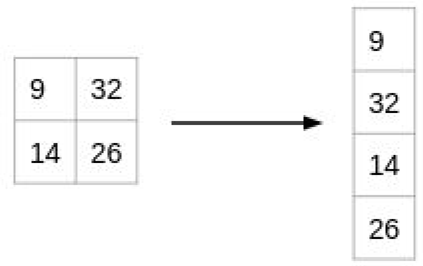
Stride: which is how many features the sliding window skips along the width and height





A common **POOLING LAYER** uses a 2 cross 2 max filter with a stride of 2, this is a non-overlapping filter. A max filter would return the max value in the features within the region. Example of max pooling would be when there is 26 across 26 across 32 volume, now by using a max pool layer that has 2 cross 2 filters and astride of 2, the volume would then be reduced to 13 crosses, 13 crosses 32 feature map.

4.FULLY CONNECTED LAYER

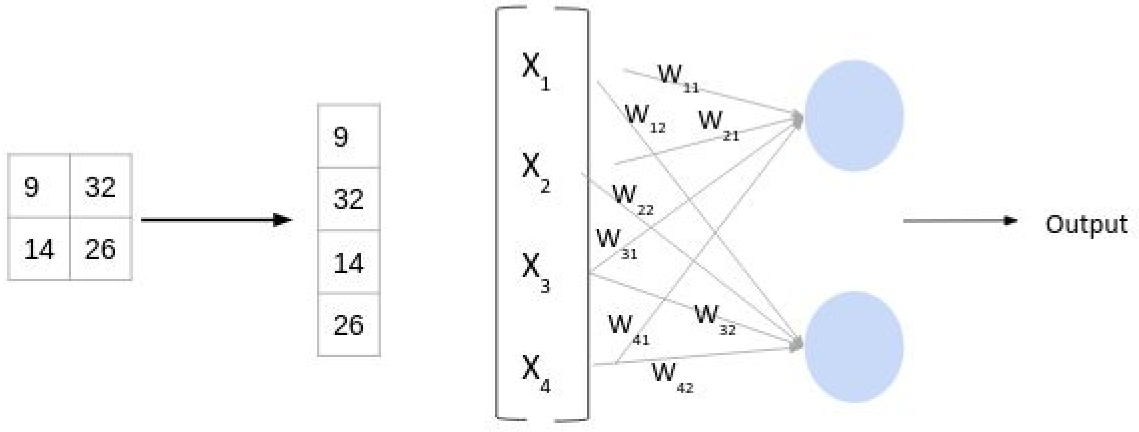


In this layer involves **Flattening**. This involves transforming the entire pooled feature map matrix into a single column which is then fed to the neural network for processing. With the fully connected layers, we combined these features together to create a model. Finally, we have an activation function such as softmax or sigmoid to classify the output.

We first perform a linear transformation on this data. The equation for linear transformation is:

$Z = W^T.X + b$

Here, X is the input, W is weight, and b (called bias) is a constant. Note that the W in this case will be a matrix of (randomly initialized) numbers. Can you guess what would be the size of this matrix?



Considering the size of the matrix is (m, n) – m will be equal to the number of features or inputs for this layer. Since we have 4 features from the convolution layer, m here would be

4. The value of n will depend on the number of neurons in the layer. For instance, if we have two neurons, then the shape of weight matrix will be (4, 2):

$$X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} \qquad W = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \\ W_{41} & W_{42} \end{bmatrix} \qquad b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

Input Data — Randomly Initialized Weight Matrix — Randomly Initialized bias Matrix

Having defined the weight and bias matrix, let us put these in the equation for linear transformation:

$$Z = W^T . X + b$$

$$Z = \begin{bmatrix} W_{11} & W_{21} & W_{31} & W_{41} \\ W_{12} & W_{22} & W_{32} & W_{42} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

$$Z_{2x2} = \begin{bmatrix} W_{11}X_1 + W_{21}X_2 + W_{31}X_3 + W_{41}X_4 \\ W_{12}X_1 + W_{22}X_2 + W_{32}X_3 + W_{42}X_4 \end{bmatrix}$$

Now, there is one final step in the forward propagation process – the non-linear transformations. Let us understand the concept of non-linear transformation and it’s role in the forward propagation process.

Non-Linear transformation

The linear transformation alone cannot capture complex relationships. Thus, we introduce an additional component in the network which adds non-linearity to the data. This new component in the architecture is called the **activation function**.

TThese activation functions are added at each layer in the neural network. The activation function to be used will depend on the type of problem you are solving.

We will be working on a binary classification problem and will use the Sigmoid activation function. Let's quickly look at the mathematical expression for this:

$f(x) = 1/(1+e^{-x})$

The range of a Sigmoid function is between 0 and 1. This means that for any input value, the result would always be in the range (0, 1). A Sigmoid function is majorly used for binary classification problems and we will use this for both convolution and fully-connected layers.

Let's quickly summarize what we've covered so far.

Forward Propagation Summary

Step 1: Load the input images in a variable (say X)

Step 2: Define (randomly initialize) a filter matrix. Images are convolved with the filter

$Z_1 = X * f$

Step 3: Apply the Sigmoid activation function on the result

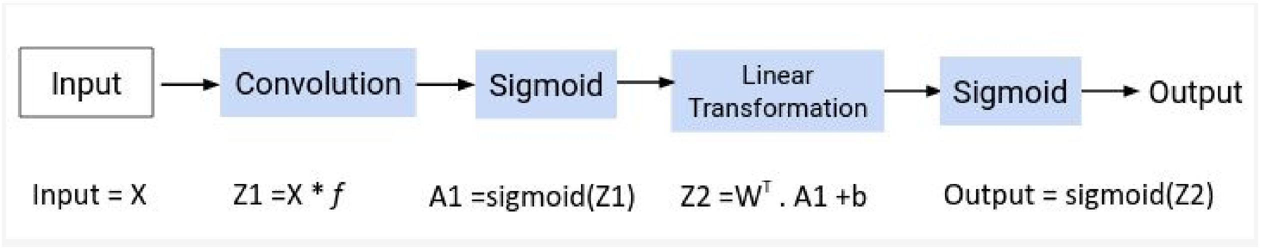
$A = \text{sigmoid}(Z_1)$

Step 4: Define (randomly initialize) weight and bias matrix. Apply linear transformation on the values

$Z_2 = W_T.A + b$

Step 5: Apply the Sigmoid function on the data. This will be the final output

$O = \text{sigmoid}(Z_2)$



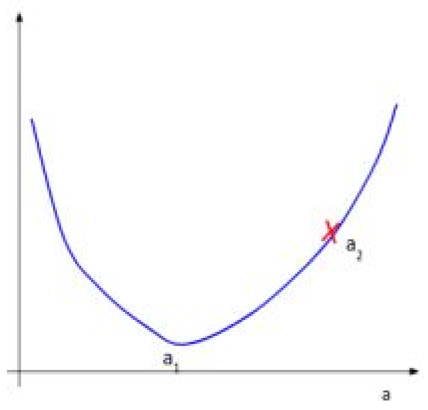
Now the question is – how are the values in the filter decided? The CNN model treats these values as parameters, which are randomly initialized and learned during the training process. We will answer this in the next section.

Convolutional Neural Network (CNN): Backward Propagation

During the forward propagation process, we randomly initialized the weights, biases and filters. These values are treated as parameters from the convolutional neural network algorithm. In the backward propagation process, the model tries to update the parameters such that the overall predictions are more accurate.

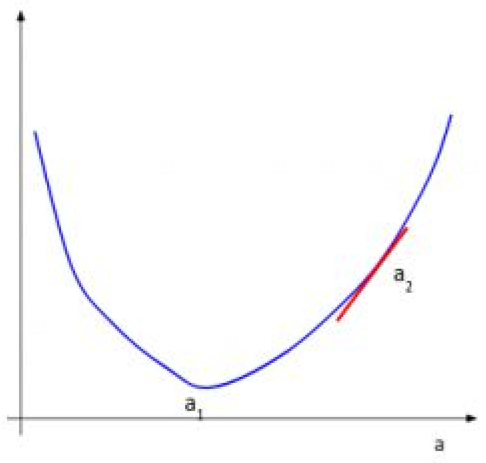
For updating these parameters, we use the gradient descent technique. Let us understand the concept of gradient descent with a simple example.

Consider that following in the curve for our loss function where we have a parameter a:



During the random initialization of the parameter, we get the value of a as a2. It is clear from the picture that the minimum value of loss is at a1 and not a2. The gradient descent technique tries to find this value of parameter (a) at which the loss is minimum.

We understand that we need to update the value a2 and bring it closer to a1. To decide the direction of movement, i.e. whether to increase or decrease the value of the parameter, we calculate the gradient or slope at the current point.



gradient descent cnn

Based on the value of the gradient, we can determine the updated parameter values. When the slope is negative, the value of the parameter will be increased, and when the slope is positive, the value of the parameter should be decreased by a small amount.

Here is a generic equation for updating the parameter values:

new_parameter = old_parameter - (learning_rate * gradient_of_parameter)

The learning rate is a constant that controls the amount of change being made to the old value. The slope or the gradient determine the direction of the new values, that is, should the values be increased or decreased. So, we need to find the gradients, that is, change in error with respect to the parameters in order to update the parameter values.

Understanding Chain Rule in Backpropagation:

Consider this equation

$f(x,y,z) = (x + y)z$

To make it simpler, let us split it into two equations.

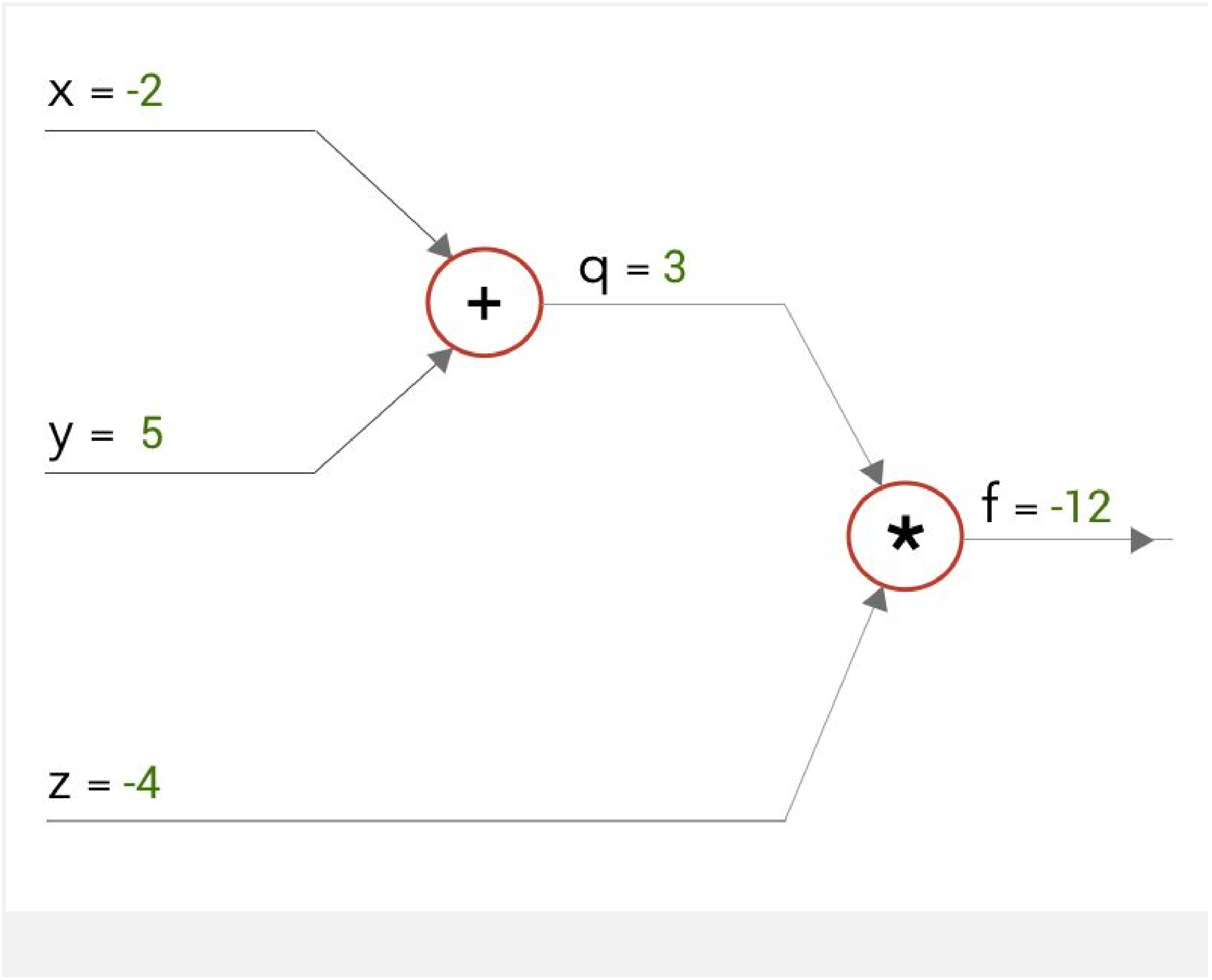
$$f(x,y,z) = (x+y)z$$

$$q = x + y$$

$$f = q * z$$

Now, let us draw a computational graph for it with values of x, y, z

as **x = -2, y = 5, z = 4.**



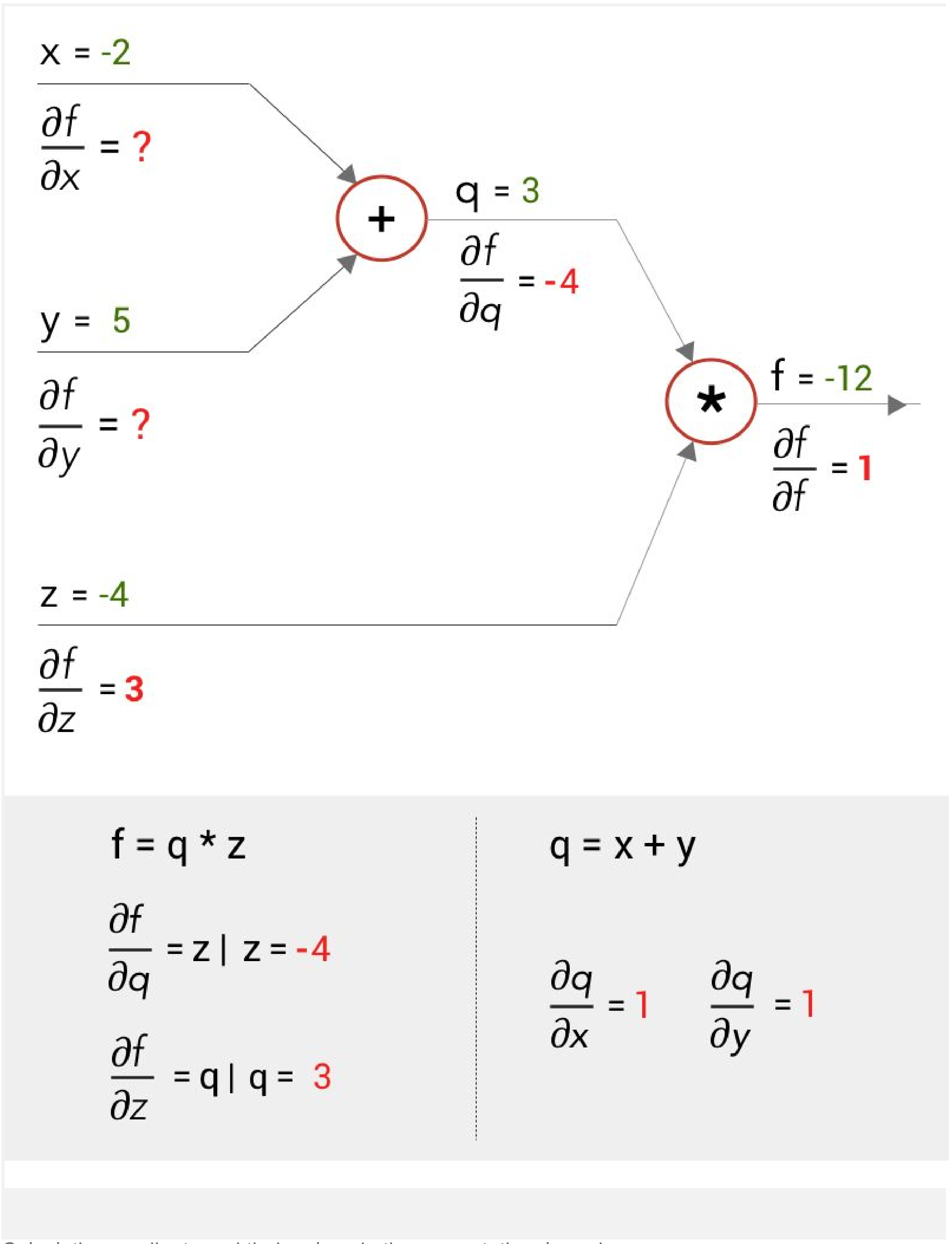
Computational Graph of f = q*z where q = x + y

When we solve for the equations, as we move from left to right, ('the forward pass'), we get an output of **f = -12**

Now let us do the backward pass. Say, just like in Backpropagations, we derive the gradients moving from right to left at each stage. So, at the end, we have to get the values of the gradients of our inputs x,y and z — ***∂*f/*∂*x** *and* ***∂*f/*∂*y** *and* ***∂*f/*∂*z**

(*differentiating function f in terms of x,y and z)*

Working from right to left, at the multiply gate we can differentiate ***f*** to get the gradients at ***q*** and ***z*** — ***∂*f/*∂*q** and ***∂*f/*∂*z** . And at the add gate, we can differentiate ***q*** to get the gradients at ***x*** and ***y*** — ***∂*q/*∂*x** and ***∂*q/*∂*y.**



$x = -2$
$\frac{\partial f}{\partial x} = ?$
$y = 5$
$\frac{\partial f}{\partial y} = ?$
$z = -4$
$\frac{\partial f}{\partial z} = 3$
+
$q = 3$
$\frac{\partial f}{\partial q} = -4$
*
$f = -12$
$\frac{\partial f}{\partial f} = 1$
$f = q * z$
$\frac{\partial f}{\partial q} = z \mid z = -4$
$\frac{\partial f}{\partial z} = q \mid q = 3$
$q = x + y$
$\frac{\partial q}{\partial x} = 1$
$\frac{\partial q}{\partial y} = 1$

Calculating gradients and their values in the computational graph

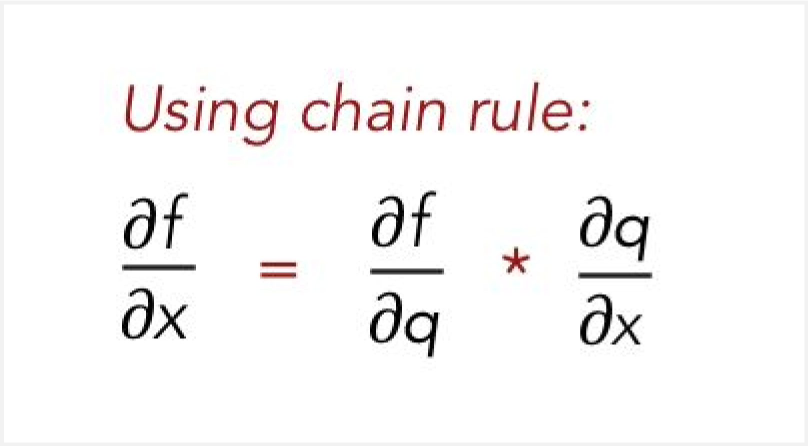
We have to find **∂f/∂x** and **∂f/∂y** but we only have got the values of **∂q/∂x** and **∂q/∂y.** So, how do we go about it?

$$\frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1 \qquad \frac{\partial f}{\partial q} = -4$$

$$\frac{\partial f}{\partial x} = ? \qquad \frac{\partial f}{\partial y} = ?$$

How do we find ∂f/∂x and ∂f/∂y

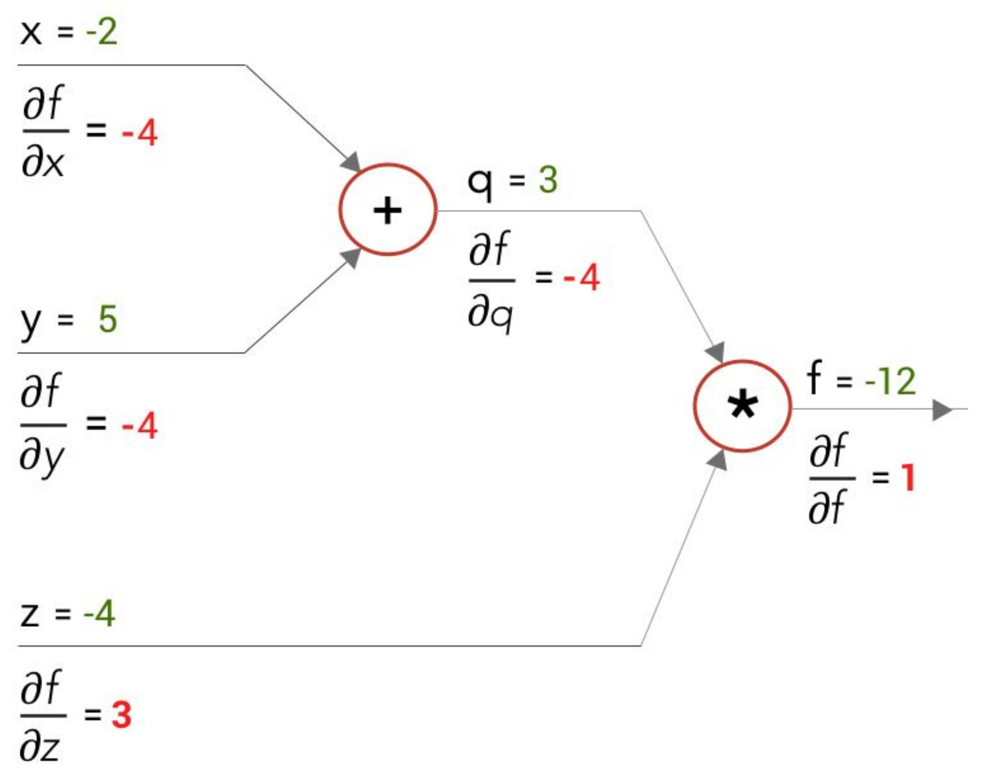
This can be done using the chain rule of differentiation. By the chain rule, we can find **∂f/∂x** as



Using chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} * \frac{\partial q}{\partial x}$$

And we can calculate **∂f/∂x** and **∂f/∂y** as:



$x = -2$
$\frac{\partial f}{\partial x} = -4$
$y = 5$
$\frac{\partial f}{\partial y} = -4$
$z = -4$
$\frac{\partial f}{\partial z} = 3$
+
$q = 3$
$\frac{\partial f}{\partial q} = -4$
*
$f = -12$
$\frac{\partial f}{\partial f} = 1$

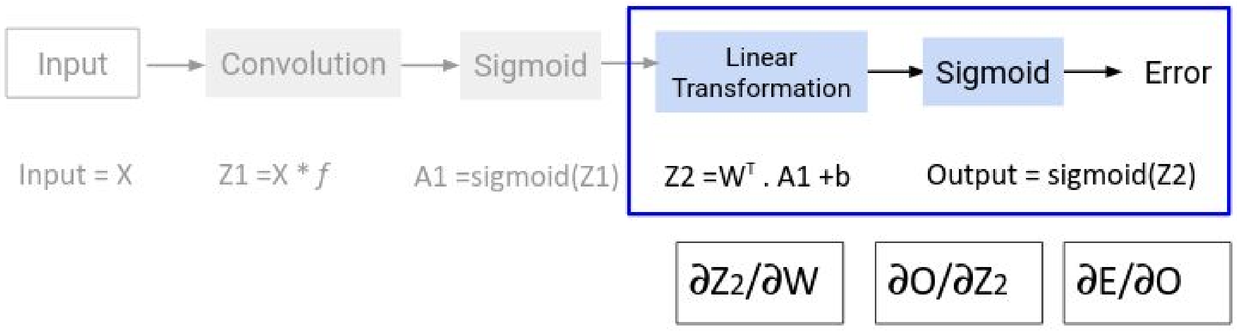
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} * \frac{\partial q}{\partial x} = -4 * 1 = -4$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} * \frac{\partial q}{\partial y} = -4 * 1 = -4$$

Backward Propagation: Fully Connected Layer

As discussed previously, the fully connected layer has two parameters – weight matrix and bias matrix. Let us start by calculating the change in error with respect to weights – ∂E/∂W.

Since the error is not directly dependent on the weight matrix, we will use the concept of chain rule to find this value. The computation graph shown below will help us define ∂E/∂W:



Backward Propagation (Fully Connected layer)

$\partial E/\partial W = \partial E/\partial O \, . \, \partial O/\partial Z_2 . \, \partial z/\partial W$

We will find the values of these derivatives separately.

1. Change in error with respect to output

Suppose the actual values for the data are denoted as y’ and the predicted output is represented as O. Then the error would be given by this equation:

$E = (y' - O)^2/2$

If we differentiate the error with respect to the output, we will get the following equation:

$\partial E/\partial O = -(y'-O)$

2. Change in output with respect to Z_2 (linear transformation output)

To find the derivative of output O with respect to Z_2, we must first define O in terms of Z_2. If you look at the computation graph from the forward propagation section above, you would see that the output is simply the sigmoid of Z_2. Thus, $\partial O/\partial Z_2$ is effectively the derivative of Sigmoid. Recall the equation for the Sigmoid function:

f(x) = 1/(1+e^-x)

The derivative of this function comes out to be:

$f'(x) = (1+e_{-x})_{-1}[1-(1+e_{-x})_{-1}]$

f'(x) = sigmoid(x)(1-sigmoid(x))

$\partial O/\partial Z_2 = (O)(1-O)$

You can read about the complete derivation of the Sigmoid function here.

3. Change in Z_2 with respect to W (Weights)

The value Z_2 is the result of the linear transformation process. Here is the equation of Z_2 in terms of weights:

$Z_2 = W_T.A_1 + b$

On differentiating Z_2 with respect to W, we will get the value A_1 itself:

$\partial Z_2/\partial W = A_1$

Now that we have the individual derivations, we can use the chain rule to find the change in error with respect to weights:

$\partial E/\partial W = \partial E/\partial O \ . \ \partial O/\partial Z_2. \ \partial Z_2/\partial W$

$\partial E/\partial W = -(y'-o) \ . \ sigmoid'. \ A_1$

The shape of ∂E/∂W will be the same as the weight matrix W. We can update the values in the weight matrix using the following equation:

W_new = W_old - lr*∂E/∂W

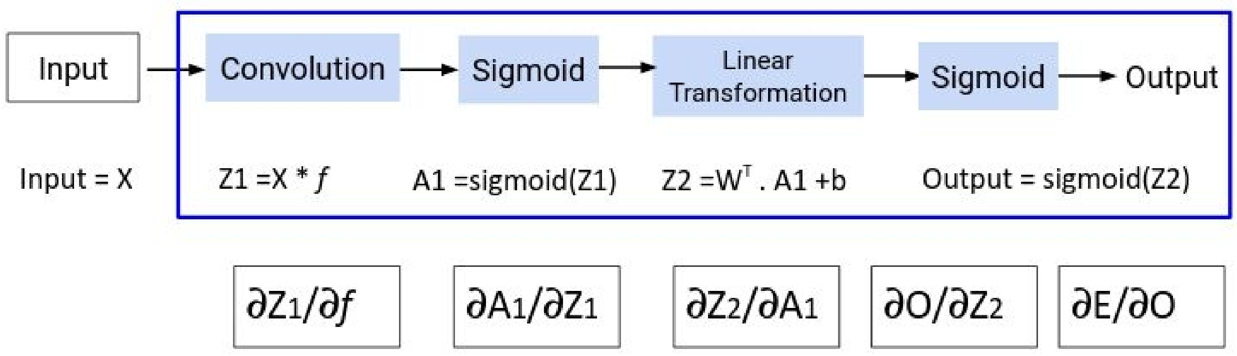
Updating the bias matrix follows the same procedure. Try to solve that yourself and share the final equations in the comments section below!

Backward Propagation: Convolution Layer

For the convolution layer, we had the filter matrix as our parameter. During the forward propagation process, we randomly initialized the filter matrix. We will now update these values using the following equation:

*new_parameter = old_parameter - (learning_rate * gradient_of_parameter)*

To update the filter matrix, we need to find the gradient of the parameter – dE/df. Here is the computation graph for backward propagation:



Backward Propagation (Convolution layer)

From the above graph we can define the derivative $\partial E/\partial f$ as:

$\partial E/\partial f = \partial E/\partial O . \partial O/\partial Z_2 . \partial Z_2/\partial A_1 . \partial A_1/\partial Z_1 . \partial Z_1/\partial f$

We have already determined the values for $\partial E/\partial O$ and $\partial O/\partial Z2$. Let us find the values for the remaining derivatives.

1. Change in Z_2 with respect to A_1

To find the value for $\partial Z_2/\partial A_1$, we need to have the equation for Z_2 in terms of A_1:

$Z_2 = W_T.A_1 + b$

On differentiating the above equation with respect to A_1, we get W_T as the result:

$\partial Z_2/\partial A_1 = W_T$

2. Change in A_1 with respect to Z_1

The next value that we need to determine is $\partial A_1/\partial Z_1$. Have a look at the equation of A_1

$A_1 = \text{sigmoid}(Z_1)$

This is simply the Sigmoid function. The derivative of Sigmoid would be:

$\partial A_1/\partial Z_1 = (A_1)(1-A_1)$

3. Change in Z_1 with respect to filter *f*

Finally, we need the value for $\partial Z_1/\partial f$. Here's the equation for Z_1

$Z_1 = X * f$

Differentiating Z with respect to X will simply give us X:

$\partial Z_1/\partial f = X$

Now that we have all the required values, let’s find the overall change in error with respect to the filter:

$\partial E/\partial f = \partial E/\partial O . \partial O/\partial Z_2 . \partial Z_2/\partial A_1 . \partial A_1/\partial Z_1 * \partial Z_1/\partial f$

Notice that in the equation above, the value ($\partial E/\partial O . \partial O/\partial Z2 . \partial Z2/\partial A1 . \partial A1/\partial Z$) is convolved with $\partial Z1/\partial f$ instead of using a simple dot product. Why? The main reason is that during forward propagation, we perform a convolution operation for the images and filters.

This is repeated in the backward propagation process. Once we have the value for $\partial E/\partial f$, we will use this value to update the original filter value:

$f = f - lr*(\partial E/\partial f)$

This completes the backpropagation section for convolutional neural networks