N8104: Artificial Neural Networks

Convolutional Neural Networks

Sachchida Nand Chaurasia

Assistant Professor

Department of Computer Science

Banaras Hindu University

Varanasi

Email id: snchaurasia@bhu.ac.in, sachchidanand.mca07@gmail.com

June 14, 2023

Cross-Entropy Loss Function I

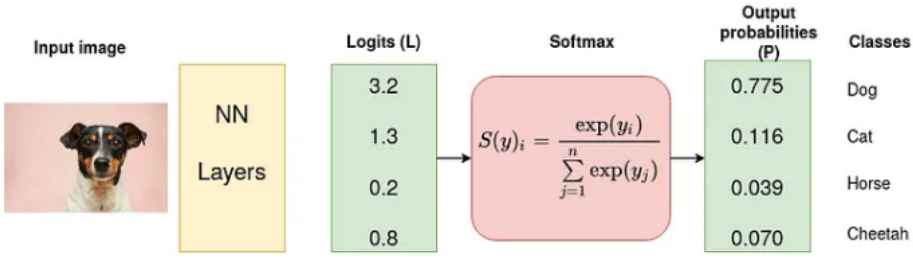
✓ When working on a Machine Learning or a Deep Learning Problem, loss/cost functions are used to optimize the model during training.

✓ The objective is almost always to minimize the loss function. The lower the loss the better the model.

✓ Cross-Entropy loss is a most important cost function. It is used to optimize classification models. The understanding of Cross-Entropy is pegged on understanding of Softmax activation function.

✓ Consider a 4-class classification task where an image is classified as either a dog, cat, horse or cheetah.

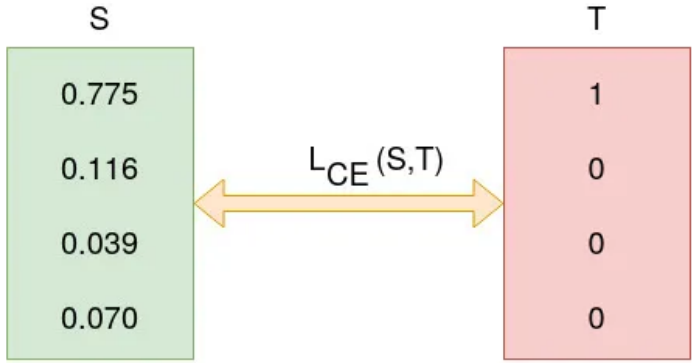
Cross-Entropy Loss Function II



Input image source: Photo by Victor Grabarczyk on Unsplash . Diagram by author.

✓ In the above Figure, Softmax converts logits into probabilities. The purpose of the Cross-Entropy is to take the output probabilities (P) and measure the distance from the truth values (as shown in Figure below).

Cross-Entropy Loss Function III



Cross-Entropy Loss Function IV

✓ For the example above the desired output is [1,0,0,0] for the class dog but the model outputs [0.775, 0.116, 0.039, 0.070] .

✓ The objective is to make the model output be as close as possible to the desired output (truth values).

✓ During model training, the model weights are iteratively adjusted accordingly with the aim of minimizing the Cross-Entropy loss. The process of adjusting the weights is what defines model training and as the model keeps training and the loss is getting minimized, we say that the model is learning.

Entropy

✓ Entropy of a random variable X is the level of uncertainty inherent in the variables possible outcome.

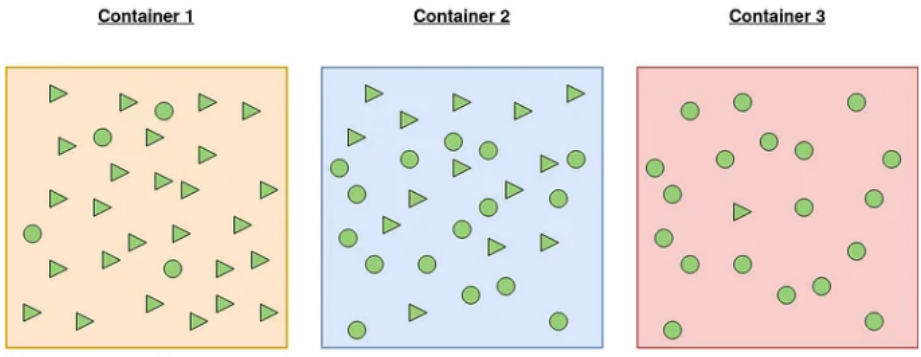
✓ For $p(x)$ — probability distribution and a random variable X, entropy is defined as follows:

$$H(X) = \begin{cases} -\int_x p(x) \log p(x) & \text{X is continous} \\ \sum_x p(x) \log p(x) & \text{X is discrete} \end{cases}$$

The greater the value of entropy, $H(x)$, the greater the uncertainty for probability distribution and the smaller the value the less the uncertainty.

Cross-Entropy Loss Function VI

Example



3 containers with triangle and circle shapes. (Source: Author).

Cross-Entropy Loss Function VII

✓ *Container 1: The probability of picking a triangle is 26/30 and the probability of picking a circle is 4/30. For this reason, the probability of picking one shape and/or not picking another is more certain.*

✓ *Container 2: Probability of picking the a triangular shape is 14/30 and 16/30 otherwise. There is almost 50–50 chance of picking any particular shape. Less certainty of picking a given shape than in 1.*

✓ *Container 3: A shape picked from container 3 is highly likely to be a circle. Probability of picking a circle is 29/30 and the probability of picking a triangle is 1/30. It is highly certain than the shape picked will be circle.*

Let us calculate the entropy so that we ascertain our assertions about the certainty of picking a given shape.

Cross-Entropy Loss Function VIII

Entropy for container 1:

$$
\begin{aligned}
H(X) &= -\sum_{x} p(x) \log(p(x)) \\
&= -\left[p(x_1) \log_2 \left(p(x_1)\right) + p(x_2) \log_2 \left(p(x_2)\right)\right] \\
&= -\left[\frac{26}{30} \log_2 \left(\frac{26}{30}\right) + \frac{4}{30} \log_2 \left(\frac{4}{30}\right)\right] \\
&= 0.5665
\end{aligned}
$$

Cross-Entropy Loss Function IX

Entropy for container 2:

$$
\begin{aligned}
H(X) &= -\sum_{x} p(x) \log(p(x)) \\
&= -\left[p(x_1) \log_2 \left(p(x_1)\right) + p(x_2) \log_2 \left(p(x_2)\right)\right] \\
&= -\left[\frac{14}{30} \log_2 \left(\frac{14}{30}\right) + \frac{16}{30} \log_2 \left(\frac{16}{30}\right)\right] \\
&= 0.9968
\end{aligned}
$$

Entropy for container 3:

$$
\begin{aligned}
H(X) &= -\sum_{x} p(x) \log(p(x)) \\
&= -\left[p(x_1) \log_2 \left(p(x_1)\right) + p(x_2) \log_2 \left(p(x_2)\right)\right] \\
&= -\left[\frac{1}{30} \log_2 \left(\frac{1}{30}\right) + \frac{29}{30} \log_2 \left(\frac{29}{30}\right)\right] \\
&= 0.2108
\end{aligned}
$$

✓ As expected the entropy for the first and third container is smaller than the second one. This is because probability of picking a given shape is more certain in container 1 and 3 than in 2.

Cross-Entropy Loss Function (logarithmic loss, log loss or logistic loss):

✓ Each predicted class probability is compared to the actual class desired output 0 or 1 and a score/loss is calculated that penalizes the probability based on how far it is from the actual expected value.

✓ The penalty is logarithmic in nature yielding a large score for large differences close to 1 and small score for small differences tending to 0.

✓ Cross-entropy loss is used when adjusting model weights during training. The aim is to minimize the loss, i.e, the smaller the loss the better the model. A perfect model has a cross-entropy loss of 0.

Cross-entropy is defined as:

$$L_{CE} = -\sum_{i=1}^{n} t_i \log(p_i)$$

where t_i is the truth label and p_i is the Softmax probability for the i^{th} class. n is numbe of classes.

Binary Cross-Entropy Loss

✓ For binary classification (a classification task with two classes — 0 and 1), we have binary cross-entropy defined as:

$$\begin{aligned} L &= -\sum_{i=1}^{2} t_i \log(p(i) \\ &= -\left[t_1 \log(p_1) + t_2 \log(p_2)\right] \\ &= -\left[t \log(p) + (1-t) \log(1-p)\right] \end{aligned}$$

where t_i is the truth value taking a value 0 or 1 and p_i is the Softmax probability for the i^{th} class. Since we have two classess 1 and 0 we can have $t_1 = 1$ and $t_2 = 0$ and since p's are

Cross-Entropy Loss Function XIV

probabilities then $p_1 + p_2 = 1 \longrightarrow p_1 = 1 - p_2$. For the convenience of notation, we can then let $t_1 = t$, $t_2 = 1 - t$, $p_1 = p$ and $p_2 = 1 - p$.

✓ Binary cross-entropy is often calculated as the average cross-entropy across all data examples, that is,

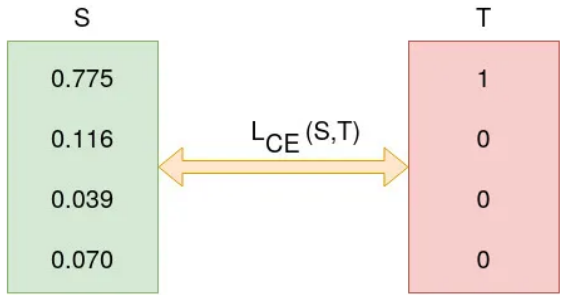
$$L = -\frac{1}{N}\left[\sum_{j=1}^{N}[t_j \log(p_j) + (1 - t_j)\log(1 - p_j)]\right]$$

for N data points where t_i is the truth value taking a value 0 or 1 and p_i is the Softmax probability for the i^{th} data point.

Cross-Entropy Loss Function XV

Example:

Consider the classification problem with the following Softmax probabilities (S) and the labels (T). The objective is to calculate for cross-entropy loss given these information.



Logits(S) and one-hot encoded truth label(T) with Categorical Cross-Entropy loss function used to measure the 'distance' between the predicted probabilities and the truth labels. (Source: Author)

Cross-Entropy Loss Function XVI

The categorical cross-entropy is computed as follows:

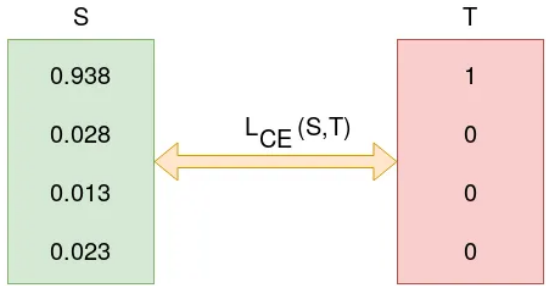
$$
\begin{aligned}
L_{CE} &= -\sum_{i=1} T_i \log(S_i) \\
&= -\left[1 * \log_2(0.775) + 0 * \log_2(0.126) + 0 * \log_2(0.039) + 0 * \log_2(0.070)\right] \\
&= -\log(0.775) \\
&= 0.3677
\end{aligned}
$$

✓ Softmax is continuously differentiable function. This makes it possible to calculate the derivative of the loss function with respect to every weight in the neural network.

Cross-Entropy Loss Function XVII

✓ This property allows the model to adjust the weights accordingly to minimize the loss function (model output close to the true values).

✓ Assume that after some iterations of model training the model outputs the following vector of logits



Cross-Entropy Loss Function XVIII

$$\begin{aligned} L_{CE} &= -1 * \log_2(0.938) + 0 + 0 + 0 \\ &= 0.095 \end{aligned}$$

0.095 is less than previous loss, that is, 0.3677 implying that the model is learning. The process of optimization (adjusting weights so that the output is close to true values) continues until training is over.

Softmax I

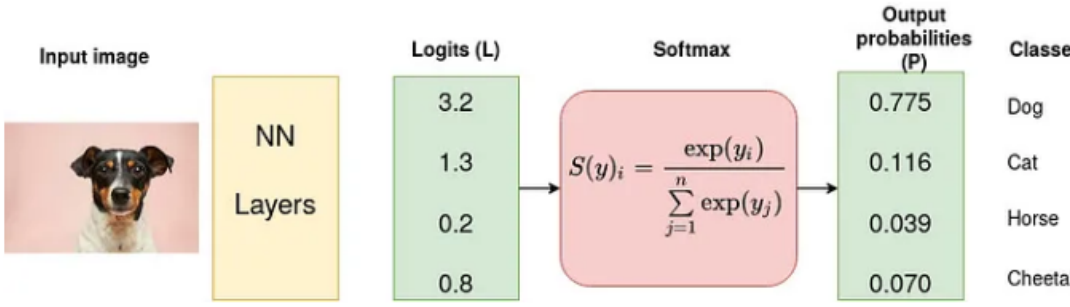
✓ It is often used as the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted output classes.

✓ Softmax is an activation function that scales numbers/logits into probabilities.

✓ The output of a Softmax is a vector (say v) with probabilities of each possible outcome. The probabilities in vector v sums to one for all possible outcomes or classes.

$$S(y)_i = \frac{\exp(y_i)}{\sum_{j=1}^{n} \exp(y_i)}$$

Softmax II



Input image source: Photo by Victor Grabarczyk on Unsplash . Diagram by author.

Softmax III

$$\exp(3.2) = 24.5325$$
$$\exp(1.3) = 3.6693$$
$$\exp(0.2) = 1.2214$$
$$\exp(0.8) = 2.2255$$

and therefore,

$$\begin{aligned} S(3.2) &= \frac{\exp(3.2)}{\exp(3.2) + \exp(1.3) + \exp(0.2) + \exp(0.8)} \\ &= \frac{24.5325}{24.5325 + 3.6693 + 1.2214 + 2.2255} \\ &= 0.775 \end{aligned}$$

Categorical Data into Numerical Data:

✓ The truth labels are categorical data: any particular image can be categorized into one of these groups: dog, cat, horse or cheetah.

✓ The computer however does not understand this kind of data and therefore we need to convert them into numerical data. There are two ways to do so:

1. Integer encoding
2. One-hot encoding

Integer Encoding (Also called Label Encoding):

✓ In this kind of encoding, labels are assigned unique integer values. For example in our case, we will have,

0 for “dog”, 1 for “cat”, 2 for “horse” and 3 for “cheetah”.

Softmax V

✓ When to use integer encoding: It is used when the labels are ordinal in nature, that is, labels with some order, for example, consider a classification problem where we want to classify a service as either poor, neutral or good, then we can encode these classes as follows

0 for “poor”, 1 for “neutral” and 2 for “good”.

Clearly, the labels have some order and the labels gives the weights to the labels accordingly.

✓ Conversely, we refrain from using integer encoding when the labels are nominal (names without specific ordering), for example, consider the flower classification problem where we have 3 classes: Iris setosa, Iris versicolor and Iris virginica,

0 for “Iris setosa”, 1 for “Iris versicolor” and 2 for “Iris virginica”.

Softmax VI

✓ The model may take a natural ordering of the labels (2>1>0) and give more weight to one class over another when in fact these are just labels with no specific ordering implied.

One-hot encoding:

✓ For categorical variables where no such ordinal relationship exists, the integer encoding is not enough. One-hot encoding is preferred.

✓ In one-hot encoding, the labels are represented by a binary variable (1 and 0s) such that for a given class a binary variable with 1 for position corresponding to that specific class and 0 elsewhere is generated, for example, in our case we will have the following labels for our 4 classes

[1, 0, 0, 0] for “dog”, [0, 1, 0, 0] for “cat”, [0, 0, 1, 0] for “horse” and [0, 0, 0, 1] for “cheetah”.

Softmax VII

But one may ask, why not use standard normalization, that is, take each logit and divide it by the sum of the all logits to get the probabilities? Why take the exponents? Here are some two reasons.

1. Softmax normalization reacts to small and large variation/change differently but standard normalization does not differentiate the stimulus by intensity so longest the proportion is the same, for example,
   # Softmax normalization
   softmax([2, 4]) = [0.119, 0.881]
   softmax([4, 8]) = [0.018, 0.982]
   # Standard normalization
   std_norm([2, 4]) = [0.333, 0.667]

Softmax VIII

std_norm([4, 8]) = [0.333, 0.667]

2. Another problem arises when there are negative values in the logits. In that case, you will end up with negative probabilities in the output. *The Softmax is not affected with negative values because exponent of any value (positive or negative) is always a positive value*.

Introduction I

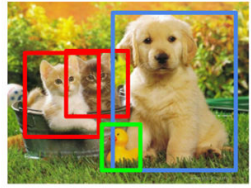
Computer Vision Problems

1. Image classifications 2. Object detection 3. Neural Style Transfer

Classification

CAT

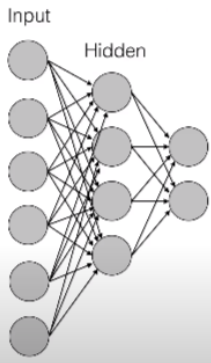
Object Detection



CAT, DOG, DUCK

Introduction II

Introduction III

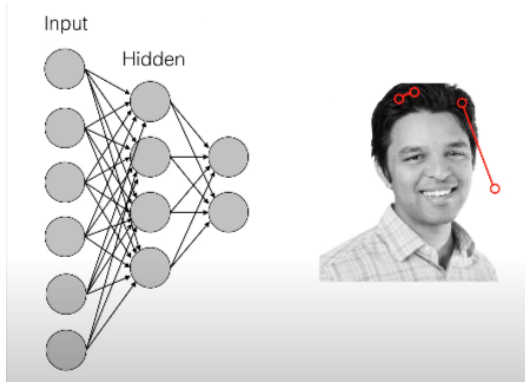


MNIST dataset: 28 x28 pixels (784 pixels)
First layer weights: ~78k parameters

Typical Image: 256 x 256 (56,000 pixels)
First layer weights: 560k parameters !

Too many parameters, unscalable to real images

Introduction IV



✓ These kind of networks do not take into account or understanding of relationship between space and pixels in the images.

Introduction V

Background of CNNs: CNN's were first developed and used around the 1980s. The most that a CNN could do at that time was recognize handwritten digits. It was mostly used in the postal sectors to read zip codes, pin codes, etc.

✓ The important thing to remember about any deep learning model is that it requires a large amount of data to train and also requires a lot of computing resources.

✓ This was a major drawback for CNNs at that period and hence CNNs were only limited to the postal sectors and it failed to enter the world of machine learning.

✓ In 2012 Alex Krizhevsky uses multi-layered neural networks. The availability of large sets of data, to be more specific ImageNet datasets with millions of labeled images and an abundance of computing resources enabled researchers to revive CNNs.

Introduction VI

What exactly is a CNN?

In deep learning, a convolutional neural network (CNN/ConvNet) is a class of deep neural networks, most commonly *applied to analyze visual imagery.*

✓ When we think of a neural network we think about matrix multiplications but that is not the case with ConvNet. It uses a special technique called *Convolution*.

Bottom line is that the role of the ConvNet is to reduce the images into a form that is easier to process, without losing features that are critical for getting a good prediction.

Introduction VII

✓ In neural networks, Convolutional neural network (ConvNets or CNNs) is one of the main categories to do images recognition, images classifications.

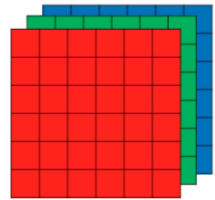
✓ Objects detections, recognition faces etc., are some of the areas where CNNs are widely used.

✓ CNN image classifications takes an input image, process it and classify it under certain categories (Eg., Dog, Cat, Tiger, Lion).

✓ Computers sees an input image as array of pixels and it depends on the image resolution. Based on the image resolution, it will see $h \times w \times d$ (h = Height, w = Width, d = Dimension).

✓ An image of $6 \times 6 \times 3$ array of matrix of RGB (3 refers to RGB values) and an image of $4 \times 4 \times 1$ array of matrix of grayscale image.

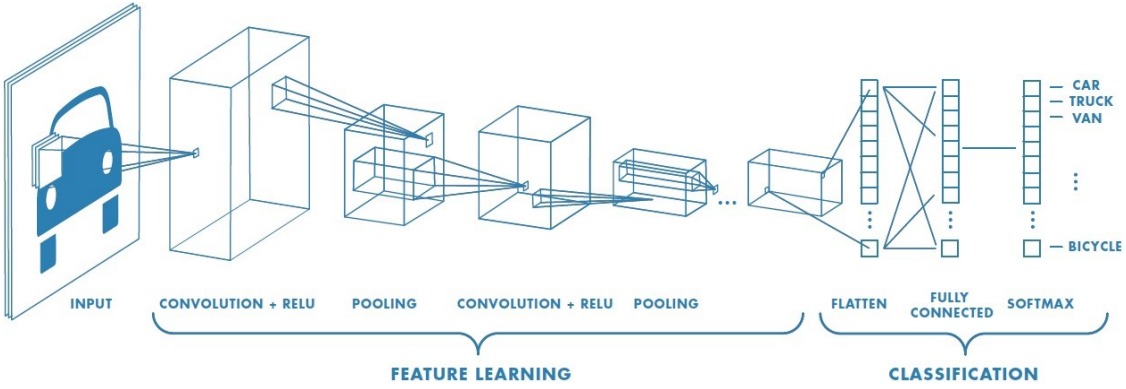
Introduction VIII



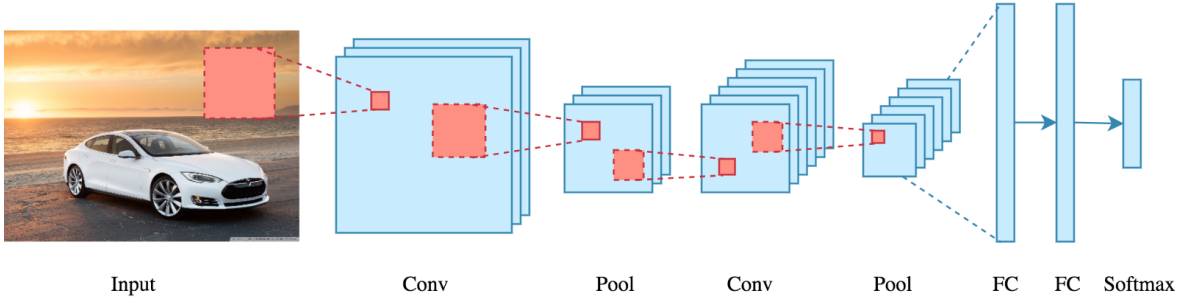
✓ Technically, deep learning CNN models to train and test, each input image will pass it through a series of convolution layers with filters (Kernels), Pooling, fully connected layers (FC) and apply Softmax function to classify an object with probabilistic values between 0 and 1.

✓ The below figure is a complete flow of CNN to process an input image and classifies the objects based on values.

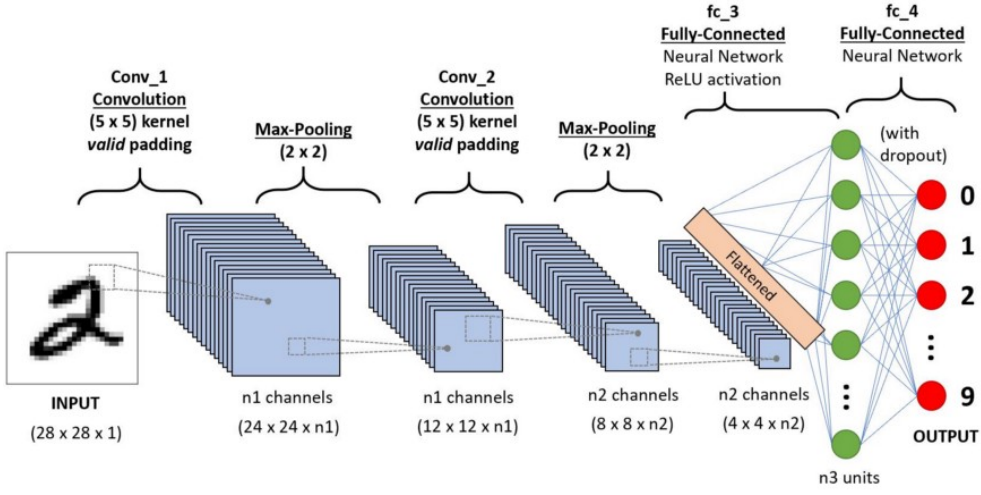
Introduction IX



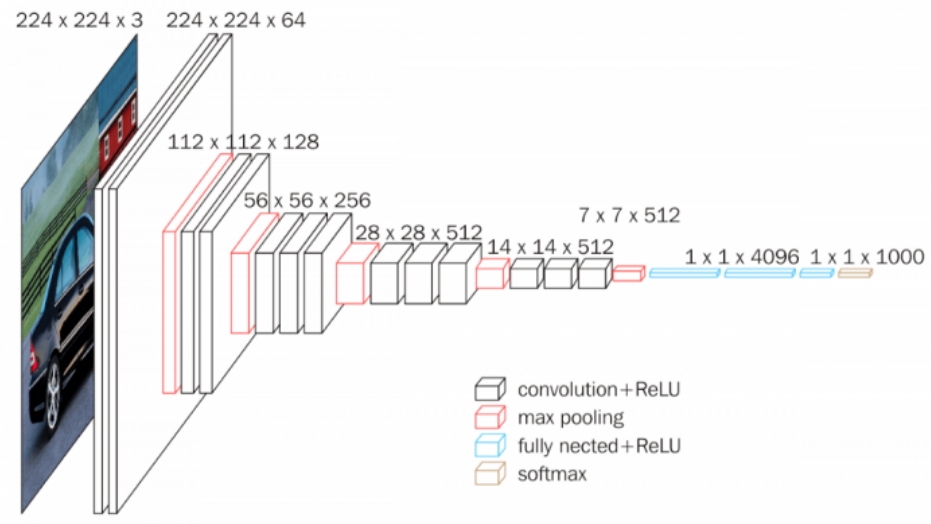
Introduction X



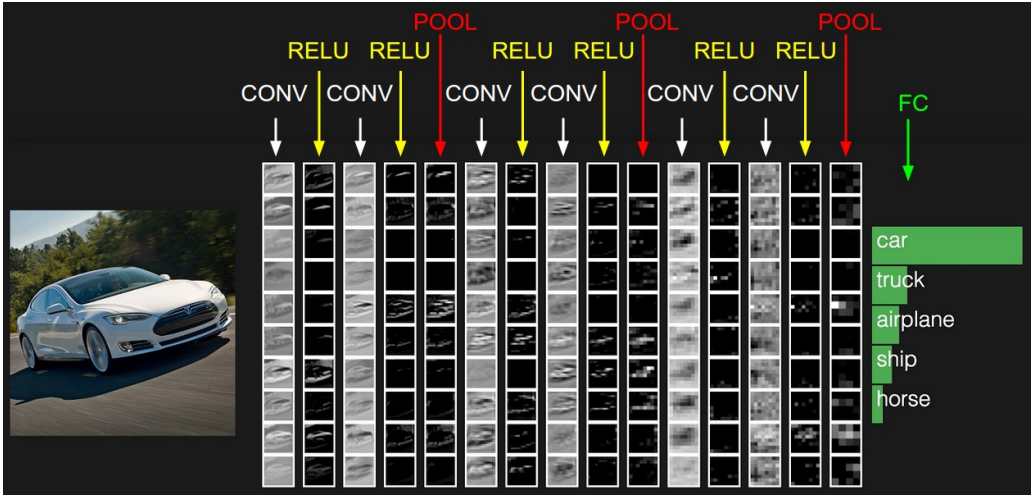
Introduction XI



Introduction XII



Introduction XIII



What is Convolution?

Convolution is a mathematical operation on two functions (f and g) that produces a third function ($f * g$) that expresses how the shape of one is modified by the other.

- The term convolution refers to both the result function and to the process of computing it.
- The fundamental difference between a densely connected layer and a convolution layer is this: Dense layers learn global patterns in their input feature space (for example, for a MNIST digit, patterns involving all pixels), whereas convolution layers learn local patterns: in the case of images, patterns found in small 2D windows of the inputs.

Introduction XV

- For example, a fully-connected first layer with 784 (28*28) input neurons, and 30 hidden neurons. Then every hidden neuron is connected to 784 input neurons and a total of 784*30=23,550 weights are involved.
- On the other hand, CNN makes connections in small localized regions of the input image, say for example 3*3 region corresponding to 9 input units. That region in the input image is called *local receptive field* for the hidden unit, and each connection learns a weight.
- This *sparse connectivity* in convolutional neural networks reduces the memory requirements of the model; require fewer operations to compute the output.

Introduction XVI

- An appropriate kernel convolved with input image can detect just a single kind of localized feature. To do image recognition, we need more than one feature maps obtained be selecting different kernels (each kernel defined by its own weight parameters) for each feature map.

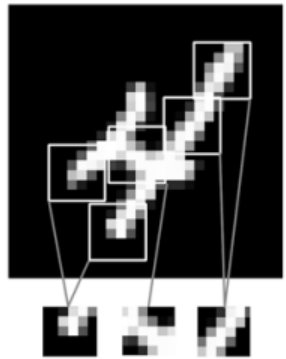


Figure 1: Images can be broken into local patterns such as edges, textures, and so on.

Introduction XVIII

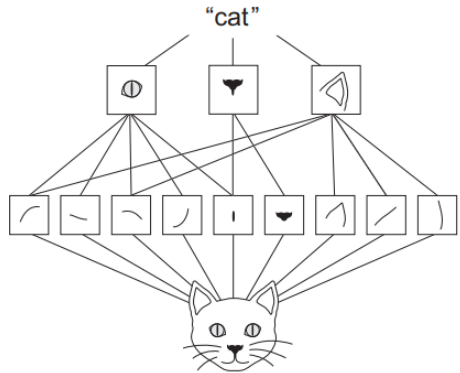


Figure 2: The visual world forms a spatial hierarchy of visual modules: hyperlocal edges combine into local objects such as eyes or ears, which combine into high-level concepts such as “cat.”

Introduction XIX

✓ The convolution of f and g is written $f * g$, denoting the operator with the symbol $*$. It is defined as the integral of the product of the two functions after one is reversed and shifted. As such, it is a particular kind of integral transform:

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau) g(t - \tau)\, d\tau \tag{0.1}$$

An equivalent definition is :

$$(f * g)(t) := \int_{-\infty}^{\infty} f(t - \tau) g(\tau)\, d\tau \tag{0.2}$$

Introduction XX

A common engineering notational convention is:

$$f(t) * g(t) := \underbrace{\int_{-\infty}^{\infty} f(\tau) g(t-\tau)\, d\tau}_{(f*g)(t)} \tag{0.3}$$

1. Express each function in terms of a dummy variable τ.
2. Reflect one of the functions:$g(\tau) \rightarrow g(-\tau)$.
3. Add a time-offset, t, which allows $g(t-\tau)$ to slide along the τ -axis.
4. Start t at $-\infty$ and slide it all the way to $+\infty$. Wherever the two functions intersect, find the integral of their product. In other words, at time t, compute the area under the function $f(\tau)$ weighted by the weighting function $g(t-\tau)$.

Properties of convolutions

For every piecewise continuous functions f, g, and h, hold:

1. Commutativity: $f * g = g * f$;
2. Associativity: $f * (g * h) = (f * g) * h$;
3. Distributivity: $f * (g + h) = f * g + f * h$;
4. Neutral element: $f * 0 = 0$;
5. Identity element: $f * \delta = f$;

Applications of convolution:

Introduction XXIII

Introduction XXIV

- In image processing:
  - ✓ In digital image processing convolutional filtering plays an important role in many important algorithms in edge detection and related processes.
  - ✓ In optics, an out-of-focus photograph is a convolution of the sharp image with a lens function.
  - ✓ In image processing applications such as adding blurring.

Convolution Layer: Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data.

✓ It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel.

✓ Initially use fewer filters and gradually increase and monitor the error rate to see how it is varying.

✓ Very small filter sizes will capture very fine details of the image. On the other hand having a bigger filter size will leave out minute details in the image.

✓ Kernel size defines the field of view of the convolution and provides a measure for how close the filed of view of input resembles a feature.

✓ Convolution with single kernel can only detect one kind of feature at many locations of the input.

✓ We want each convolution layer of our network to extract many kind of features. So a convolutional layer has several different kernels/filters consisting of trainable parameters which convolve on the input feature map spatically and detect different spatial feature based on learned weights.

✓ The size of the kernel and number of kernels are two important hyper-parameters.

Number of CL in CNN model:

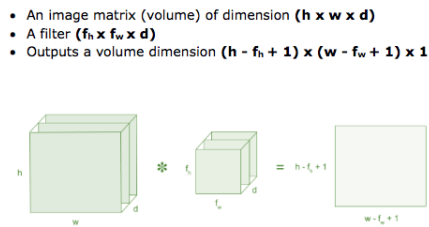
✓ The more convolutional layers the better; however, with very large numbers, the training time becomes a constraint.

✓ After about three or four layers, the gain in accuracy becomes rather small.

Introduction XXVII

✓ Since all image recognition tasks are different, the best way is to increment the number of layers one at a time until satisfactory compromise between generalization performance and training time is achieved.

- An image matrix (vloume) of dimension $(h \times w \times d)$
- A filter $(f_h \times f_w \times d)$
- Outputs a volume dimension $(h - f_h + 1) \times (w - f_w + 1) \times 1$

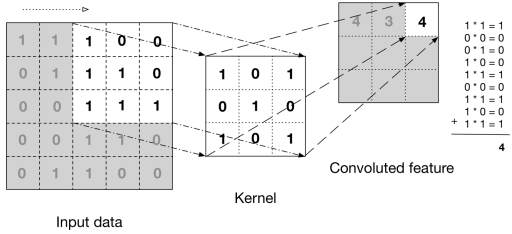


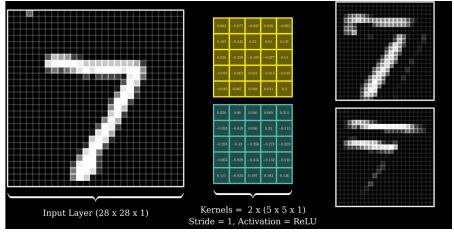
Introduction XXVIII

✓ Consider a 5×5 whose image pixel values are 0, 1 and filter matrix 3×3:

✓ Then the convolution of 5×5 image matrix multiplies with 3×3 filter matrix which is called “Feature Map” as output shown in below:

Introduction XXIX





✓ Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters. The below example shows various convolution image after applying different types of filters (Kernels).

Feature detection

- Edge detection: Canny, Deriche, Differential, Sobel, Prewitt, Roberts cross
- Corner detection: Harris operator, Shi and Tomasi, Level curve curvature, Hessian feature strength measures, SUSANFAST
- Blob detection: Laplacian of Gaussian (LoG), Difference of Gaussians (DoG), Determinant of Hessian (DoH), Maximally stable extremal regions, PCBR
- Ridge detection: Hough transform, Hough transform, Generalized Hough transform
- Structure tensor: Structure tensor, Generalized structure tensor

Introduction XXXI

- Affine invariant feature detection: Affine shape adaptation, Harris affine, Hessian affine
- Feature description: SIFT, SURF, GLOH, HOG
- Scale space: Scale-space axioms, Implementation details, Pyramids

Introduction XXXII

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

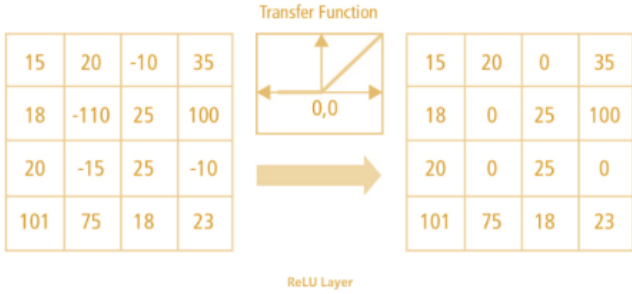
Introduction XXXIII

Non Linearity: After the convolution has been performed on the input image map, a non-linearity is included. The reason for including non-linearity in the network is because the input-output mapping that needs to be learnt is usually non-linear.

ReLU stands for Rectified Linear Unit for a non-linear operation. The output is $(x) = \max(0, x)$.

Why ReLU is important : ReLU's purpose is to introduce non-linearity in our ConvNet. Since, the real world data would want our ConvNet to learn would be non-negative linear values.

Introduction XXXIV



Introduction XXXV

Strides: Stride is the number of pixels shifts over the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on. The below figure shows convolution would work with a stride of 2.

✓ A general relationship

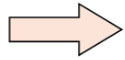
$$Output\ dimension = \left[\frac{n + 2p - k}{s}\right] + 1$$

✓ *Pooling and strading is to reject the local spatial details and preserve the most representative local spatial information.*

Introduction XXXVI

1	2	3	4	5	6	7
11	12	13	14	15	16	17
21	22	23	24	25	26	27
31	32	33	34	35	36	37
41	42	43	44	45	46	47
51	52	53	54	55	56	57
61	62	63	64	65	66	67
71	72	73	74	75	76	77

Convolve with 3x3 filters filled with ones



108	126	
288	306	

Padding: Another issue that demands consideration in designing a CNN model is that input pixels near borders influence feature output pixels than the input pixels near the center.

✓ This can make the border pixels somewhat underrepresented in the model.

✓ However, This is not a big issue in applications wherein most of the important pieces of information are situated in the middle of the image.

✓ This would have a bigger consequences if we did have meaningful data around the edge of the input. We may lose valuable data by throwing away information around the edges of the input.

✓ Padding is a technique that can be used to offset or mitigate the loss of information due to reduction in size, and border pixels getting less opportunity to interact with the kernel.

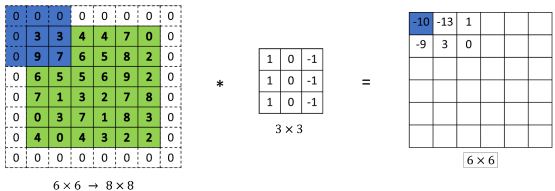
✓ Zero-padding is a kind of padding which occurs when we add a border of pixels all with value zero around the input map along both axes.

Introduction XXXVIII

✓ *This i also a hyper-parameter.*

✓ Sometimes filter does not fit perfectly fit the input image. We have two options:

- Pad the picture with zeros (zero-padding) so that it fits
- Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid part of the image.



Introduction XXXIX

Pooling Layer: Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called subsampling or downsampling which reduces the dimensionality of each map but retains important information.

✓ This is to decrease the computational power required to process the data by reducing the dimensions.

✓ Spatial pooling can be of different types:

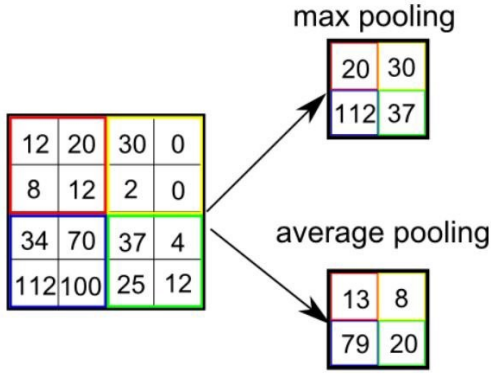
- Max Pooling
- Average Pooling
- Sum Pooling

Introduction XL

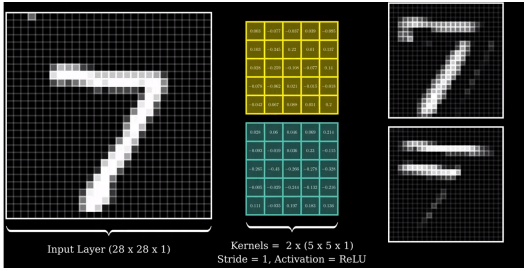
✓ Max pooling takes the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map call as sum pooling.

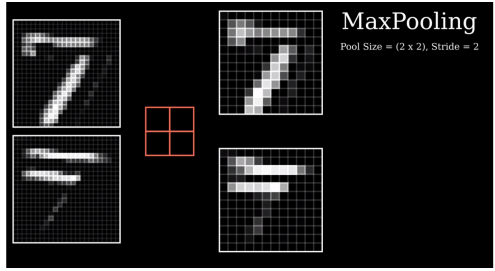
✓ Average Pooling returns the average of all the values from the portion of the image covered by the Kernel. Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism.

✓ We can say that Max Pooling performs a lot better than Average Pooling.


max pooling
20 30
112 37
12 20 30 0
8 12 2 0
34 70 37 4
112 100 25 12
average pooling
13 8
79 20

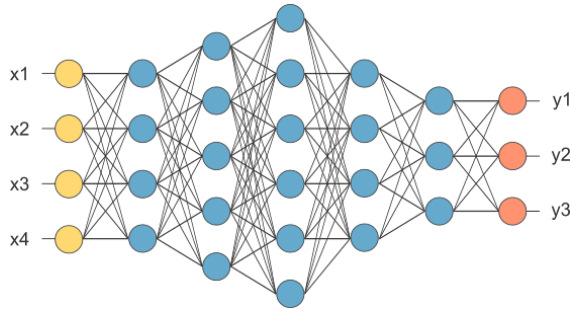
Introduction XLII





Fully Connected Layer

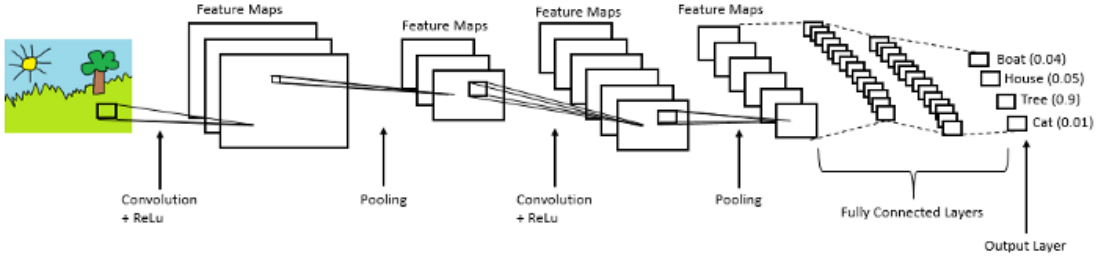
The layer we call as FC layer, we flattened our matrix into vector and feed it into a fully connected layer like a neural network.



Introduction XLIV

✓ In the above diagram, the feature map matrix will be converted as vector $(x_1, x_2, x_3, \dots)$. 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 outputs as cat, dog, car, truck etc.



Cross-correlation:

Given an input image *I* and a filter (kernel) K of dimensions $k_1 \times k_2$, the cross-correlation operation is given by:

$$(I \otimes K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i+m, j+n) K(m,n) \tag{1}$$

$$(I * K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i-m, j-n) K(m,n) \tag{0.4}$$

$$= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i+m, j+n) K(-m,-n) \tag{0.5}$$

From the equation it is easy to see that convolution is the same as cross-correlation with a flipped kernel i.e: for a kernel K where $K(-m,-n) == K(m,n)$.

✓ CNNs consists of convolutional layers which are characterized by an input map I, a bank of filters K and biases b.

Introduction XLVII

✓ In the case of images, we could have as input an image with height H, width W and $C = 3$ channels (red, blue and green) such that $I \in \mathbb{R}^{H \times W \times C}$. Subsequently for a bank of D filters we have $K \in \mathbb{R}^{k_1 \times k_2 \times C \times D}$ and biases $b \in \mathbb{R}^D$, one for each filter.

The output from this convolution procedure is:

$$(I * K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \sum_{c=1}^{C} K_{m,n,c} \cdot I_{i+m,j+n,c} + b \tag{0.6}$$

For a grayscale image:

$$(I * K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} K_{m,n} \cdot I_{i+m,j+n} + b \tag{0.7}$$

Notation

1. l is the l^{th} layer where l=1 is the first layer and $l = L$ is the last layer.
2. Input x is of dimension $H \times W$ and has i by j as the iterators.
3. Filter or kernel w is of dimension $k_1 \times k_2$ has m by n as the iterators.
4. $w^l_{m,n}$ is the weight matrix connecting neurons of layer l with neurons of layer $l-1$.
5. b^l is the bias unit at layer l.

Introduction XLIX

6. $x_{i,j}^l$ is the convolved input vector at layer l plus the bias represented as

$$x_{i,j}^l = \sum_m \sum_n w_{m,n}^l o_{i+m,j+n}^{l-1} + b^l \tag{0.8}$$

7. $o_{i,j}^l$ is the output at layer l given by

$$o_{i,j}^l = f(x_{i,j}^l) \tag{0.9}$$

8. $f(\cdot)$ is the activation function. Application of the activation layer to the convolved input vector at layer l is $f(x_{i,j}^l)$

Forward Propagation

The convolution equation of the input at layer l is given by:

$$x_{i,j}^{l} = \text{rot}_{180^\circ}\left\{w_{m,n}^{l}\right\} * o_{i,j}^{l-1} + b_{i,j}^{l} \tag{0.10}$$

$$x_{i,j}^{l} = \sum_{m}\sum_{n} w_{m,n}^{l} o_{i+m,j+n}^{l-1} + b_{i,j}^{l} \tag{0.11}$$

$$o_{i,j}^{l} = f(x_{i,j}^{l}) \tag{0.12}$$

Error

For a total of P predictions, the predicted network outputs y_p and their corresponding targeted values t_p the the mean squared error is given by:

$$E = \frac{1}{2} \sum_{p} (t_p - y_p)^2 \tag{0.13}$$

✓ Learning will be achieved by adjusting the weights such that y_p is as close as possible or equals to corresponding t_p.

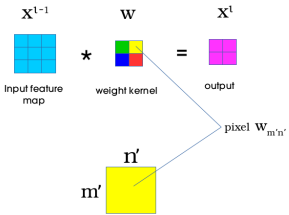
✓ In the classical backpropagation algorithm, the weights are changed according to the gradient descent direction of an error surface E.

Backpropagation

For backpropagation there are two updates performed, for the weights and the deltas. Lets begin with the weight update.

Gradient w.r.t weights is the measurement of how the change in a single pixel $w_{m',n'}$ in the weight kernel affects the loss function E.

$$\frac{\partial E}{\partial w^l_{m',n'}}$$



Introduction LIII

During forward propagation, the convolution operation ensures that the yellow pixel $w_{m',n'}$ in the weight kernel makes a contribution in all the products (between each element of the weight kernel and the input feature map element it overlaps).
This means that pixel $w_{m',n'}$ will eventually affect all the elements in the output feature map.
Convolution between the input feature map of dimension $H \times W$ and the weight kernel of dimension $k_1 \times k_2$ produces an output feature map of size $(H - k_1 + 1)$ by $(W - k_2 + 1)$.
The gradient component for the individual weights can be obtained by applying the chain rule in the following way:

$$\frac{\partial E}{\partial w^l_{m',n'}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \frac{\partial E}{\partial x^l_{i,j}} \frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} \tag{0.14}$$

$$= \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta^l_{i,j} \frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} \tag{0.15}$$

In the above equation $x^l_{i,j}$ is equivalent to

$$\left\{w^l_{m,n}\right\} o^{l-1}_{i+m,j+n} + b^l \tag{0.16}$$

$$\frac{\partial x_{i,j}^{l}}{\partial w_{m',n'}^{l}} = \frac{\partial}{\partial w_{m',n'}^{l}} \left(\sum_{m} \sum_{n} w_{m,n}^{l} o_{i+m,j+n}^{l-1} + b^{l} \right) \tag{0.17}$$

Further expanding the summations in the above equation and taking the partial derivatives for all the components results in zero values for all except the components where $m = m'$ and $n = n'$ in $w_{m,n}^{l} o_{i+m,j+n}^{l-1}$ as follows:

$$\frac{\partial x_{i,j}^{l}}{\partial w_{m',n'}^{l}} = \frac{\partial}{\partial w_{m',n'}^{l}} \left(w_{0,0}^{l} o_{i+0,j+0}^{l-1} + \cdots + w_{m',n'}^{l} o_{i+m',j+n'}^{l-1} + \cdots + b^{l} \right) \tag{0.18}$$

$$= \frac{\partial}{\partial w_{m',n'}^{l}} \left(w_{m',n'}^{l} o_{i+m',j+n'}^{l-1} \right) \tag{0.19}$$

$$= o_{i+m',j+n'}^{l-1} \tag{0.20}$$

Substituting ?? in ?? gives us the following results:

$$\frac{\partial E}{\partial w^l_{m',n'}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta^l_{i,j} o^{l-1}_{i+m',j+n'} \tag{13}$$

$$= \text{rot}_{180^\circ} \left\{ \delta^l_{i,j} \right\} * o^{l-1}_{m',n'} \tag{14}$$

Summary

- Provide input image into convolution layer.
- Choose parameters, apply filters with strides, padding if requires. Perform convolution on the image and apply ReLU activation to the matrix.
- Perform pooling to reduce dimensionality size.
- Add as many convolution layers until satisfied.
- Flatten the output and feed into a fully connected layer (FC layer).
- Output the class using an activation function (Logistic Regression with cost functions) and classifies images.