Convolutional NN: Deep Learning Part-3

NN can handle complex problems by harnessing the combined power of several neurons. ↳ uncover patterns in data.

Also NN primarily Solves :- Regression / Classification Problems.

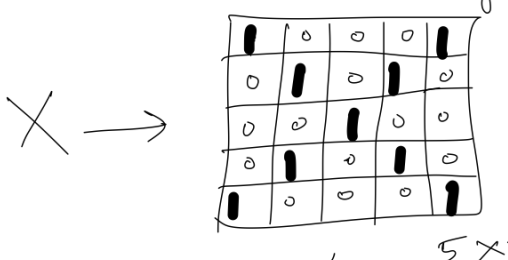
CNN are specialized models designed for image recognition tasks.

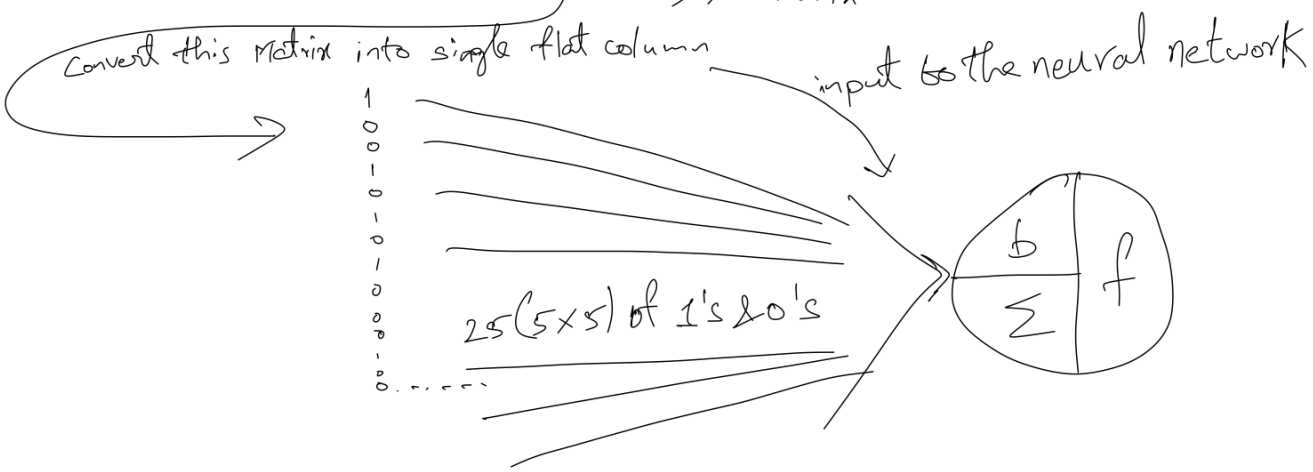
Now Lets build CNN Model to find if image represents X or not X

"This is an "X"

definitly not an "X"

Sol:- When we zoom into an image, we'll see bunch of pixels





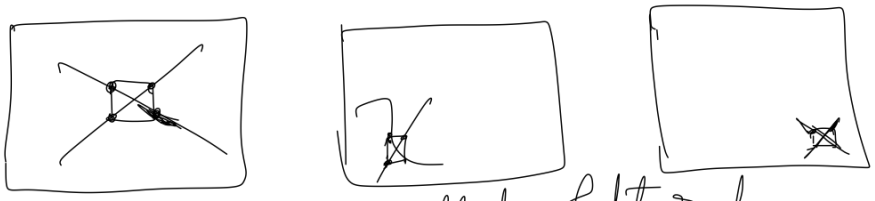
In realworld the x will be in different shapes, Also the above example will use lot of computation if we go with larger pixels image like 256×256 = 65536 weights

So, 2 issues → Reducing the inputs we feed into neural network.

→ Finding a way to detect the patterns in images

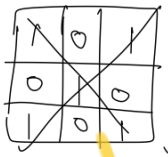
Answer:-

Filter: Finding some consistent pattern in all the 'X' images.



→ This small 'X' pattern is called filter here.

→ These filters are commonly of 3×3 pixels, although the size can vary.

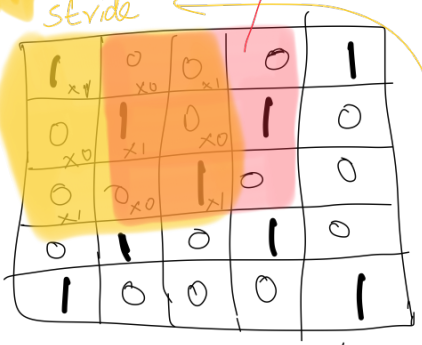


3×3

→ To apply a filter to an image for pattern detection, we slide the 3×3 filter over each section, and calculate the dot product of the filter & section it covers.

→ For First section over the filter over the first section

stride ← → stride=1, if moves to Right 1 column the stride=2, so on



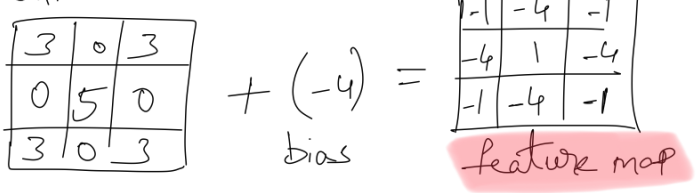
And then Multiply & add the products

1×1 + 0×0 + 0×1 + 0×0 + 1×1 + ... = 3

→ By computing the dot product b/w image & filter, we can say that the filter is convolved with the image & that's what gives Convolutional neural networks

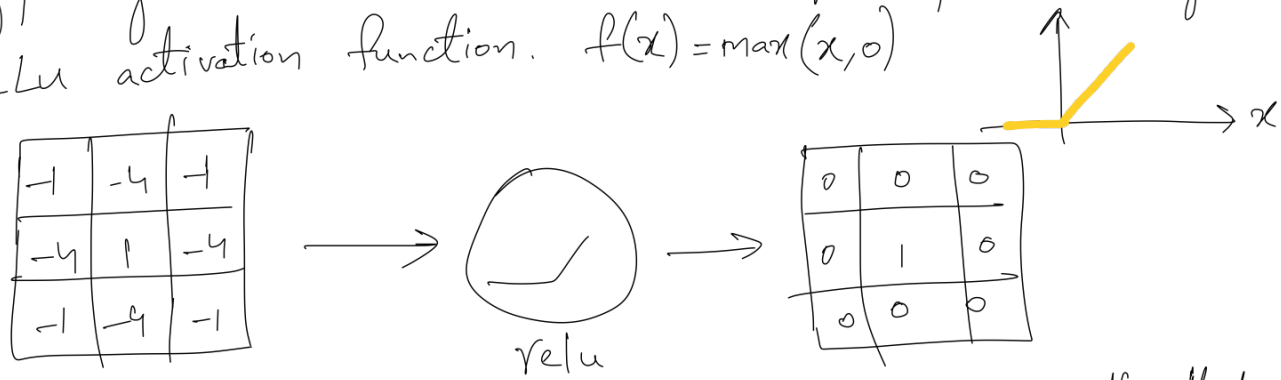
→ We do this to all the sections by sliding this fil depending on something called stride.

→ Usually the stride is set to 2, In this case lets set it to stride=1 & store all the dot products.



Note:- The larger our strides → the smaller our feature map will be.
In this example the stride=1, resulting in a relatively large feature map.

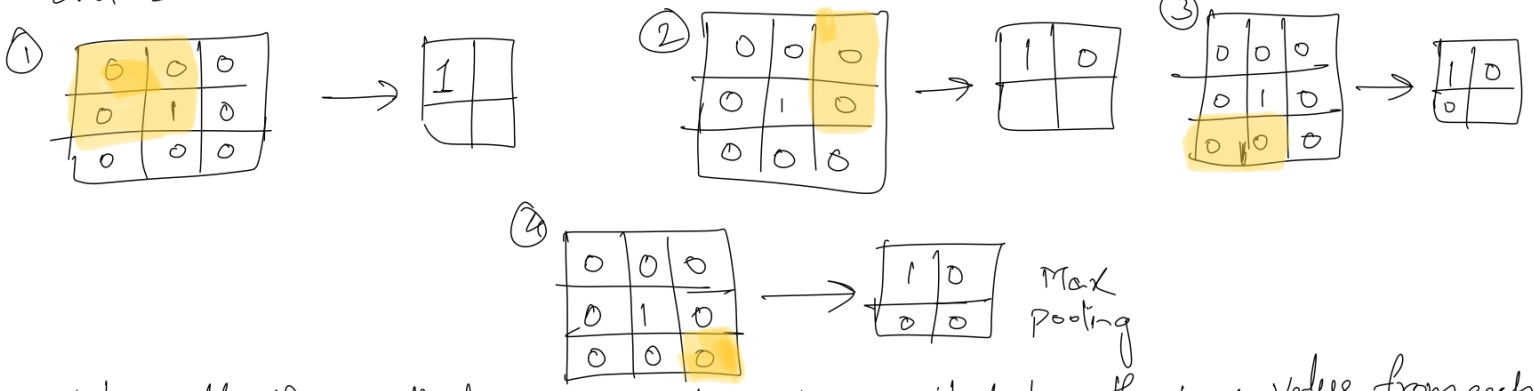
→ Typically each value in this feature map is passed through the ReLu activation function. $f(x) = \max(x, 0)$



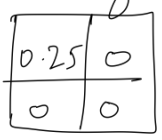
Here the middle cell has some value other than "0".

Pooling :-

With our feature map now ready, next step is pooling.
We simply scan the previously created feature map, selecting small 2x2 sections and choose the max value from each section.



→ We call this method max pooling, because it takes the max value from each section. Alternatively we could use mean pooling → calculating the average value for each region. The result looks like this:

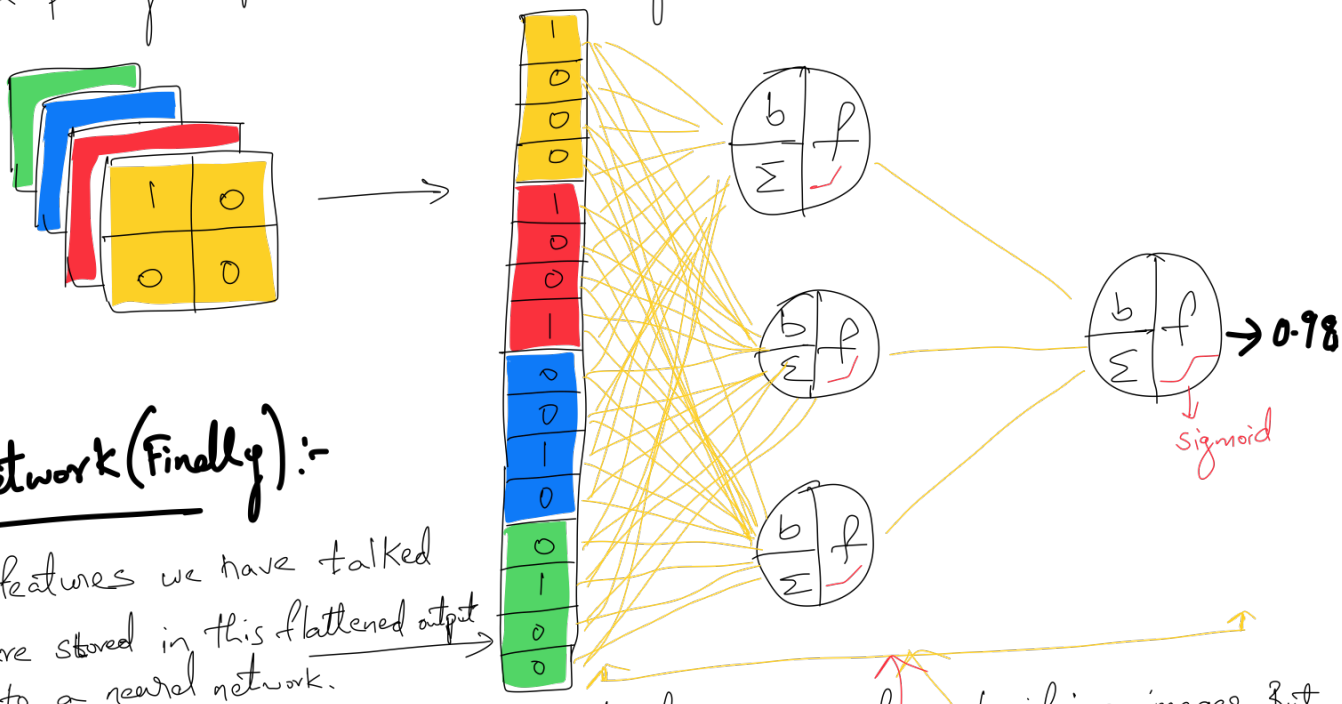


→ Sum pooling is another option by adding up all the values in each region. However Max pooling is the mostly used method.
↓
To further reduce noise in an image & its effectiveness becomes more apparent with larger images, as it identifies the area where the filter best matches the input image.

→ Now we will use the results from Max pooling as inputs for Neural Network.

Flattening:

→ Before inputting the values into a neural network, we have to flatten the feature map matrix. For instance, if we have four filters, they would result in four feature maps. Leading to four 2×2 matrices from the max pooling step. This is what they will look like flattened:



Neural Network (Finally):-

→ All the features we have talked till now are stored in this flattened output as inputs to a neural network.

→ The features already provide a good level of accuracy for classifying images. But we want to improve the model's complexity and precision. The job of ANN is to take this data and use these features to make the image classification better, which is the main reason we're creating a convolutional neural network.

→ So we take these inputs and plug them into a fully connected neural network.

Note: This is called fully connected neural network because here we are ensuring that each input & each neuron is connected to another neuron.

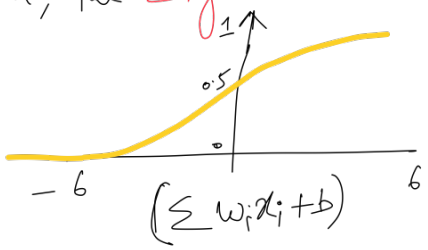
→ Now we need to select activation functions. We used ReLU activation function for all the neurons in our neural network for ice cream sales. The ReLU remains good choice for the inner layer. However, for outer neuron, its not suitable due to the different nature of the problem we are trying to solve.

→ The first scenario is was regression problem, while the current one is a classification problem. We can approach our current problem by calculating a probability. For instance, given an input image, we can determine how likely it

is that the image represents the 'x'. Here, we'll want the neural network to output values in the range of 0–1, where 1 indicates a high likelihood of being 'x' & 0 indicates it's probably not an 'x'.

→ To achieve this type of output, the Sigmoid function is a good choice.

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



→ The functions takes an input & squishes it into an S-shaped curve the outputs a value between 0 & 1. This is perfect for predicting Probabilities.

.... To get a prediction of 0.98!

↓

we need a way to check how good this 0.98 prediction is. In this case, we know our original image is an 'x', so we can say — "the CNN did a good job here!", but we need something that in math-y terms tells us the same thing.

→ In the previous model, we used the Mean Square Error (MSE) cost function to evaluate the accuracy of our prediction & used that for training process. Similarly we need to use a cost function here. But as we discussed earlier, since the kinds of predictions are different, we can't use MSE.

→ In this case we use something called Log Loss function.

$$\text{log loss} = \frac{-1}{n} \sum_{i=1}^{n} \left[\vec{y} \cdot \log(\hat{p}) + (1-y) \cdot \log(1-\hat{p}) \right]$$

actual value (0 or 1)

no. of images we want to evaluate

Predicted probability

Here, $y=1$ if the image is a 'x' & 0 if it's not.

→ For this example, $y=1$, $\hat{p} = 0.98$ & $n=1$

$$\text{log loss} = \frac{-1}{1} \sum_{i=1}^{1} \left[1 \cdot \log(0.98) + (1-1) \cdot \log(1-0.98) \right]$$

$$= -\log(0.98) = 0.0087.$$

Here, we see the cost function is very close to 0, which is good. The lower the cost function the better.

Training:

From previous article a neural network learns the optimal weights & bias through training process using gradient descent.

→ This involves running the training set through the network, making predictions & calculating costs. We keep doing this until we get the optimal values.

→ The same process happens when we train our Convolutional Neural Network, but with Two changes.

① Instead of using the MSE cost function, we use Log Loss.

② Besides finding the best weight & bias, we also look for the best filters (3×3 matrices) & bias terms in the convolution step.

→ The Filters, bias terms in the convolution step & weight, bias terms in the neural network.