Convolutional Neural Networks (CNN, ConvNets).

* Designed for visual tasks.
  - → MNIST handwritten digit recognition.
  - → object detection.
  - → face recognition.

Some history of inspiration

1981 – Hubel & Weisel won Nobel Prize for fundamental research on how neuron system of vision works. for mammals. (cats, monkeys).

Key findings

① Some neurons in the visual cortex fire electrical impulses when specific presented light at some specific angle.

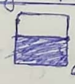
② The visual cortex is composed of sub-areas (v_1, v_2 -- etc) each of which has some specific functionality.

e.g. v_1: primary visual cortex → edges detection.
v_2: depth and motion, & so on.
some will be for color, some for facial recognition itself, etc.

③ There is a hierarchical structure among these sub-areas building from simpler to more complex and finally in the end the decision making.

ConvNets → leverages great research of such kind made for over 50yrs.
* Inspired from biology.

Convolution: Edge detection

sample image – (grayscale).  6×6 has a horizontal edge, which we'll detect.

* Edge – our eyes see an edge wherever they find an interface of difference in colors.

$$\begin{bmatrix} 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 255&255&255&255&255&255\\ 255&255&255&255&255&255\\ 255&255&255&255&255&255 \end{bmatrix}_{6\times6} * \begin{bmatrix} 1&2&1\\ 0&0&0\\ -1&-2&-1 \end{bmatrix}_{3\times3} = \begin{bmatrix} 0&0&0&0\\ -1020&-1020&-1020&-1020\\ -1020&-1020&-1020&-1020\\ 0&0&0&0 \end{bmatrix}_{4\times4}$$

input grayscale image. — convolution operation — kernel/filter

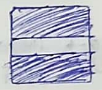
* One complete super-impose corresponds to 1-cell in O/P image.

* convolution operation – dot-product generalization over matrices.

Now, grayscale image ∈ [0, 255].

So, normalizing the image ⇒

255	255	255	255
0	0	0	0
0	0	0	0
255	255	255	255

⇒ 

So, the convolution operatⁿ on our image with sobel hor. edge detector kernel helped in detecting the edge.

There are tons of such kernels for different edges detection.

Sobel vertical edge detector = $\begin{bmatrix} 1&0&-1\\ 2&0&-2\\ 1&0&-1 \end{bmatrix}$ 3×3.

wiki – Sobel Operator. (Images)

To get both horizontal & vertical edges detected.
① do convolution using vertical detector
② do convolution using hor. detector.
③ Add both of them.

Padding

In the above example,
size of original image = 6×6.
size of resulting image = 4×4.
what if we want to preserve size?

↳ $(n\times n) \xrightarrow{(k\times k)} (n-k+1)\times(n-k+1)$
input — kernel — output.

Suppose we do a padding of p, i.e.
p extra row on top & bottom,
p extra column on left & right.

Then new dim. becomes $(n+2p)$ for original i/p image.
for o/p image, dimension $\rightarrow (n+2p-k+1) \times (n+2p-k+1)$

To maintain original dimensions,
$(n+2p-k+1) = n$.
$$\Rightarrow \boxed{p = \frac{(k-1)}{2}}$$
* usually k is taken as an odd number for divisibility.

Zero padding → The newly padded rows and columns are filled with 0.

This is most popular out of all the padding techniques.

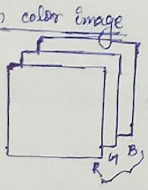
Strides

Amount of shift in right/down direction to perform convolution over the next area.

If stride = s,
$$n \times n \xrightarrow[\text{stride}=s]{(k\times k)\text{ kernel}} \left(\left\lfloor\frac{n-k}{s}\right\rfloor+1\right) \times \left(\left\lfloor\frac{n-k}{s}\right\rfloor+1\right)$$

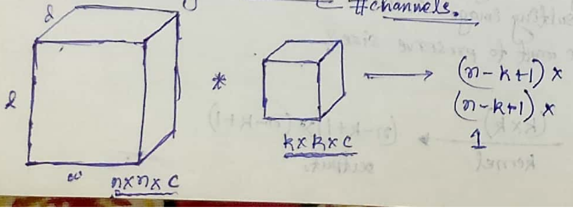
* stride reduces the o/p image size dramatically.

Condution on color image

color image –


— 3D tensor.
→ channels (old signal processing terminology).

So, color-image = $n \times m \times c$ ← #channels.

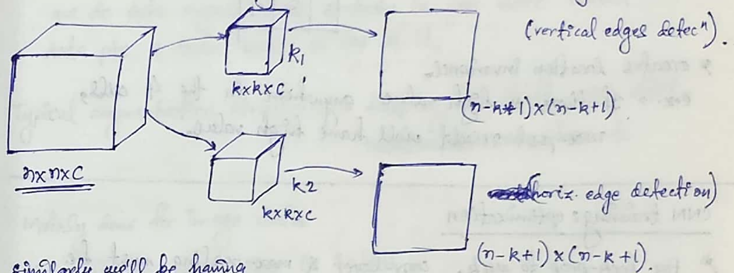


* #channels of kernel & image must be same to perform convolution operation.

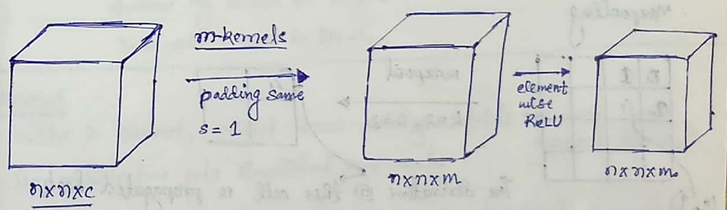
Just as in 2D, the 3D super-impose of kernel on the image corresponds to a single cell in output.

convolutional layer

In CNN, we learn the weights in kernel matrices using back prop.



* similarly we'll be having lots of different kernels to learn diff feature detection.
* We'll learn these kernel via back-prop rather than hard-coding them.
* The o/p of each kernel (a 2D matrix) can be combined to form a 3D tensor.

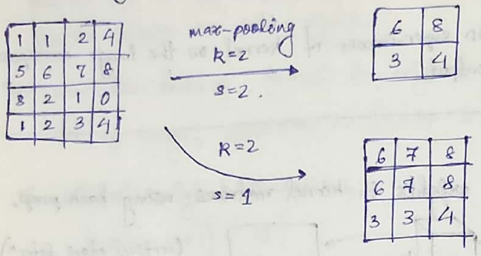


convolutional layer

* We can build a network by adding more & more conv. layers.
* similar to visual cortex, here also.
→ the earlier layer kernels detect basic edges and all.
→ the later layer compute more complicated detection of shapes etc.

So, deep ML- ConvNet ⟶ similar to visual cortex working. (hierarchical working).

Max-pooling



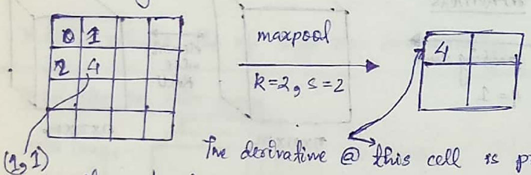
* creates location invariance.
  e.x. → If there is high value anywhere in the 4 cells, max-pool result will have high value.

CNN training: optimization

* For back-prop to work, conv-layer & max-pooling must be differentiable.
* ReLU is differentiable, no need to check.
* convolution → element-wise multiplication + Addition = Generalization of dot product over matrices.

  Hence, convolution is differentiable.

Maxpooling



The derivative @ this cell is propagated back as it is to the max. value cell i.e. (1,1).
For the remaining 3 cells, the derivate propagated will be 0.
Because the max value determines the value in that cell. no-one else.

Ref:- quora – How are parameters of max pooling represented in the weights nodes of NN.

Remaining all remains same as in MLP. – AdamOptimz, dropout, regularzn, etc,

LeNet (1998)

Ref:- world4jason.gitbooks.io – LeNet

subsampling – average pooling.

Data Augmentation

* To train CNN models robust to rotation, scale, cropping, etc, we do data augmentation ~~and train~~ to get more varied data pts & train model on top of it.

Typical augmentation strategies → flipping, hor. shift, vert. shift, rotation, zoom, shear etc.

* Mainly done for image data.
* Keras library does this in 1-line

Keras code:→ github.com/100~/mnist-lenet-keras

AlexNet

└ The model which was used on ImageNet dataset, in 2012.

Refer:- original research paper.
Andrew Ng slides for diagm.
Yale univ. – code in keras.

VGGNet

* Unlike in AlexNet, VGGNet fixes padding & stride for conv. and maxpool layers.
* The architecture gets simplified very much.

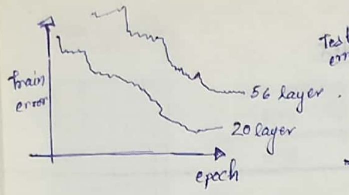
Refer:- Quora ans – What is the VGG neural network? By Yugandhar Nanda.

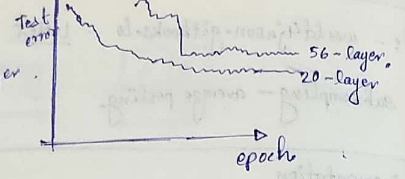
Code:- github.com/keras-team/... keras-appn/vgg16.py.

ResNet (MLP as well as CNN).

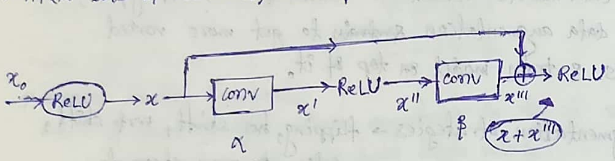
Even after using interesting ideas like – dropout, ReLU, etc.

* Training and Test error for a 56-layer NN is worse than that of 20 layer NN.





* After lots of research, came up with idea of "skip connection."



* x is o/p fm a ReLU unit. Hence it can be 0 or (+)ve.

x : +ve $\longrightarrow$ $\text{ReLU}(x) = x$.
x : 0 $\longrightarrow$ $\text{ReLU}(x) = x$.

* say we're using regularization.
* Regularization will remove all the useless weights.
* say α & β layers are useless.
* so their weights would be ≈ 0.

So, $x' \approx x'' \approx x''' \approx 0$

So, $\text{ReLU}(x + x''') = \text{ReLU}(x) = x$.

* Hence, if some of the intermediate layers are learning some useless feature, we can skip through them easily with help of skip-connection, like an identify funct.
* And if the new layers are useful then good.
* So, now adding new layer won't hurt performance ever.
* Important → The weights/kernels of α & β must be such that x & x''' have same dimensions, hence could be added.

Code :- github.com/keras-team/keras/resnet50.py.

Inception Network

Ref :- ashukumar27.io/CNN-Inception-Network/. — great detailed & concise as well.

Code :- github.com/keras-team../inception_v3.py.

Transfer Learning

Task :- classify images into dogs and cats. (2-class).

Idea :- Instead of building a NN from scratch to solve our task, we can re-use existing models (VGG-16) trained on a different dataset.

* Keras has VGG-16 trained on ImageNet.

Case I :- use VGG-16 as a feature engineering tool.

* Keep till the flatten layer of VGG-16.
* Now, we have $D = \{x_i, y_i\}_{i=1}^{n}$
* For each x_i, pass it through the NN. to get x_i' fm flatten layer.
* $D' = \{x_i', y_i\}$ – Build a simple linear regression model on this data.

Case II :- use VGG16 as base NN & modify last few layers.

* Except the last few layers, freeze the remaining layers, i.e. their weights shouldn't change during back-prop.
* use the given dataset D to fine-tune the last few layers using backprop. Keep learning rate small.

Case III :- use VGG16 as base NN & modify it using given dataset without freezing any of the layers.

Ref :- cs231.github.io/transfer-learning.