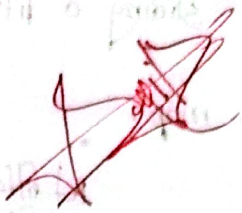
Convolutional neural network (CNN)

CNN

CNNs are specialized deep learning architectures designed primarily for image and spatial data. They mimic the process how human brain process the visual informations.



Structure

=> Convolutional Layers : Extract features using filter (Mostly edges)

=> Pooling Layers : Downsample feature maps to reduce dimentionality.

=> fully connected Layers : Combine extracted features for classification.

Why important and advantages

=> Fewer parameters → less overfitting

=> Able to capture spatial hierarchy like edges → Shape → Objects.

Applications

=> Image classification

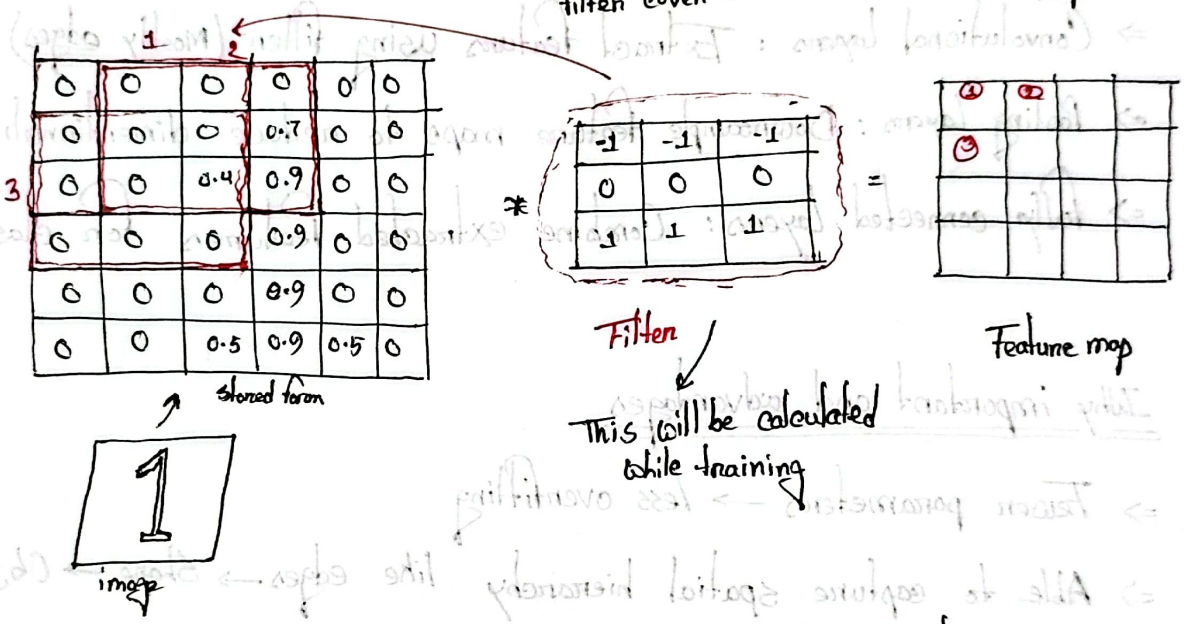
=> Object detection

=> Medical imaging

=> Face recognition and many more.

Convolution operation

Convolution is a mathematical operation that combines two functions. in CNNs, this means sliding a filter (kernel) over the input image to produce a feature map.



$$\text{output size} = \frac{n - f + 2p}{s} + 1$$

- n = input size
- f = filter size
- p = padding
- s = stride

Visit deeplizard website to understand practically

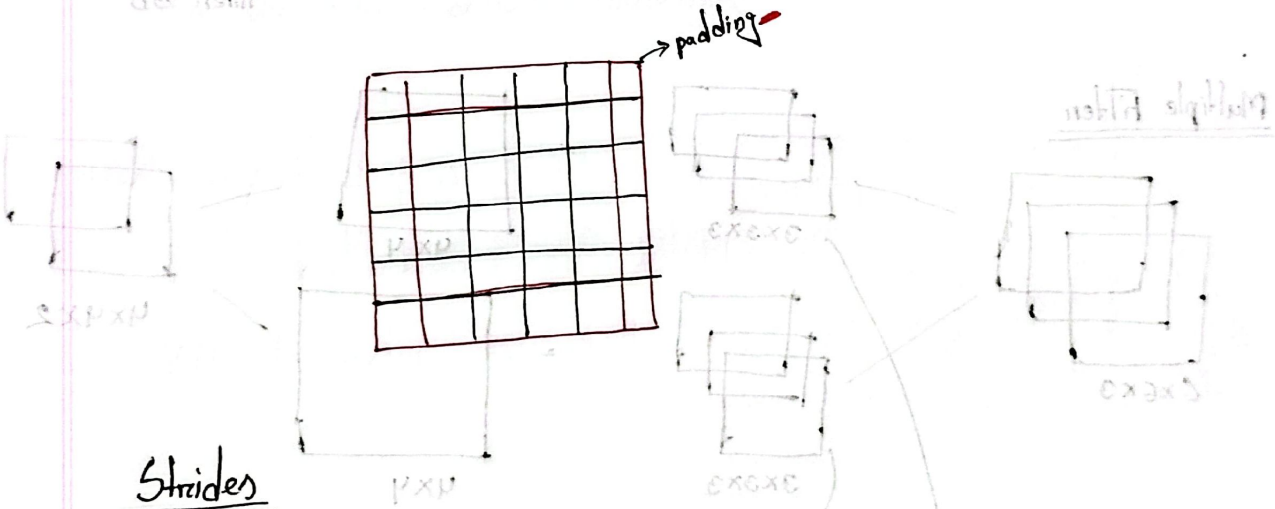
Padding and Strides

Padding

While appling filter on an image its dimension reduces. For this, we may loss information. So, to avoide this reduction, padding is introduced, which add some extra pixels (usually zeros) around the images edges.

=> Valid padding → No padding, output smaller

=> Same padding → Input size = output size



Strides

It determines how fast the filter moves in each steps.

» stride = 1 : Detailed, slow

» Stride > 1 : Faster but less detailed

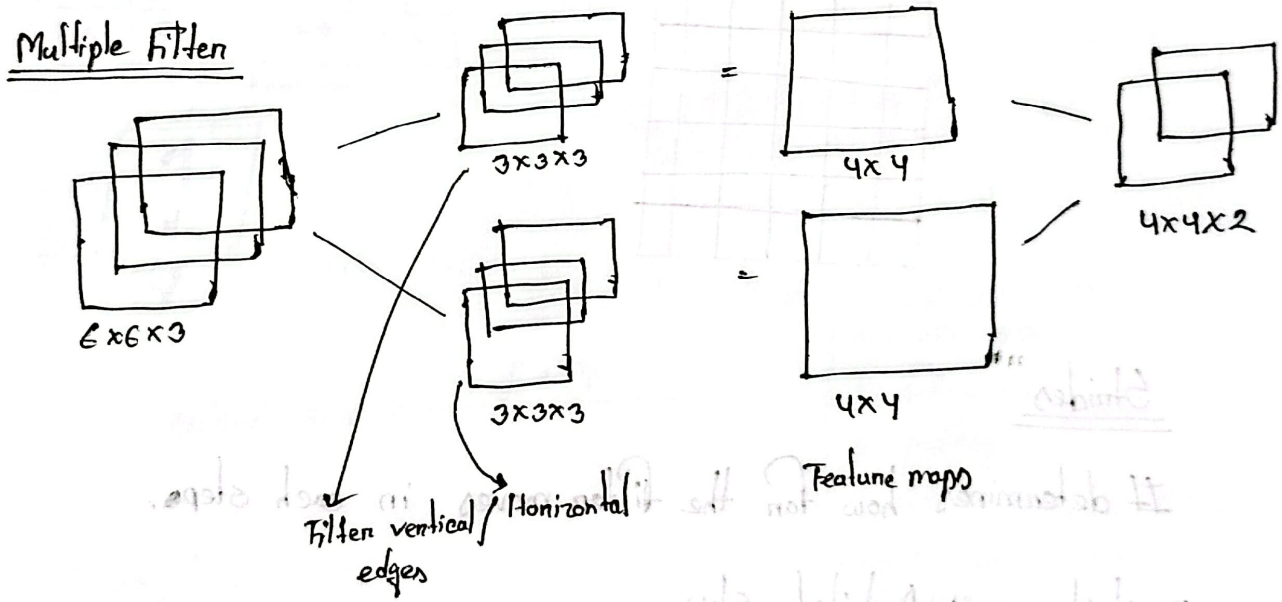
Working with RGB image

As digital images are RGB that can form any color in the picture, so, in 2D input image cannot do the thing in CNNs. Thats why we make three ~~chanels~~ channels each for R, G and B.

We can also, or we must do the same for filter. Three filter for three channels.

More formally, the dimension increased from 2D to 3D. | Input 3D
Filter 3D

Multiple Filter



Problems with convolution

① High Computing cost

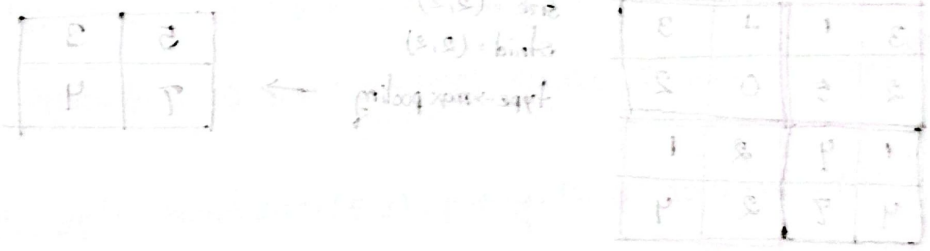
=> many parameters and multiplication → slow and memory heavy

② Overfitting

③ Translation variance

Small shift (transitions) in the input image causes different output -
it doesn't perfectly recognize the object if it moves slightly.

So, we need translation invariance.



Pooling Layers

Reduces the spatial size of a feature map → makes computation efficient and prevent overfitting.

Types

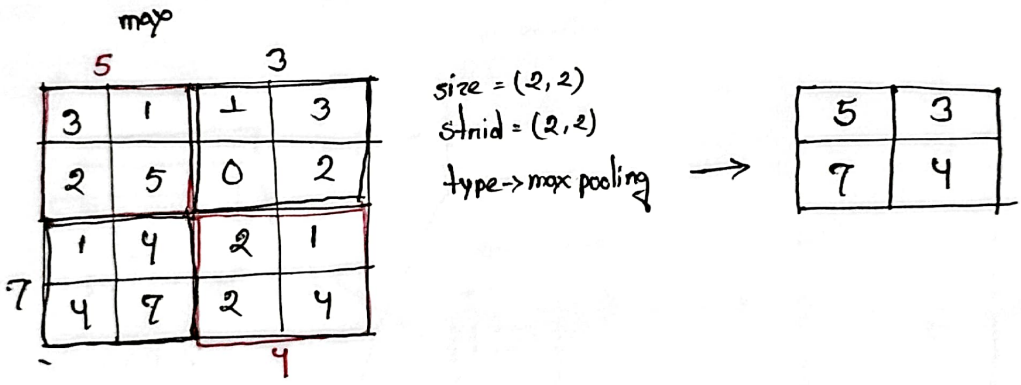
- Max pooling **
- Min pooling
- Avg pooling
- Global pooling

Effects

=> Reduces dimentionality

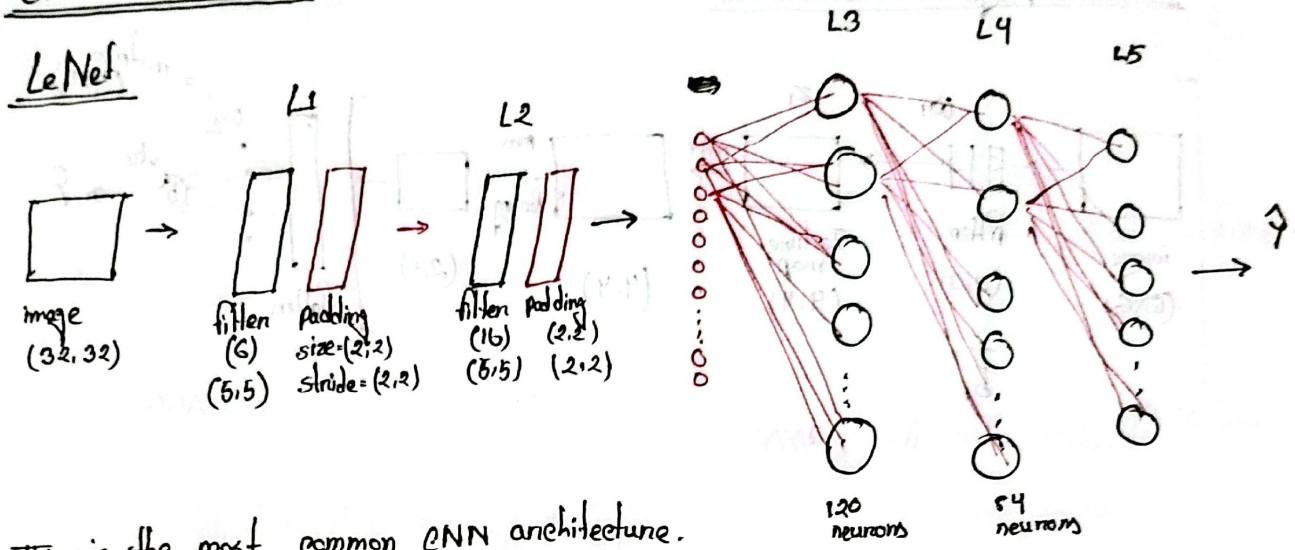
=> increase invariance to small shift

=> generalize



CNN architecture

LeNet



This is the most common CNN architecture.

Demo application on mnist

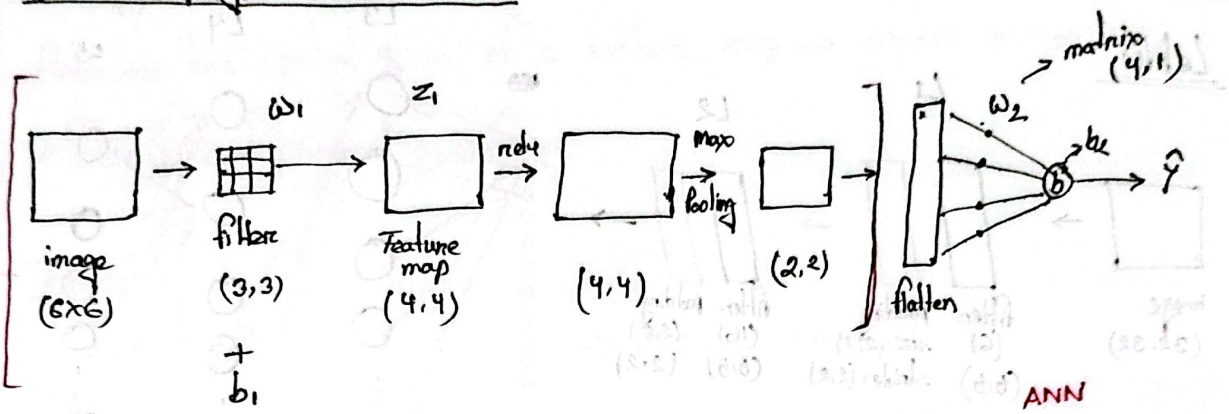
```
model = Sequential()
model.add(Conv2D(6, kernel_size=(5,5), padding='valid', activation='tanh', input_dim=(32,32,1)))
model.add(AveragePooling2D(pool_size=(2,2), strides=2, padding='valid'))

model.add(Conv2D(16, kernel_size=(5,5), padding='valid', activation='tanh'))
model.add(AveragePooling2D(pool_size=(2,2), strides=2, padding='valid'))

model.flatten
model.add(Flatten())

model.add(Dense(120, activation='tanh'))
model.add(Dense(84, activation='tanh'))
model.add(Dense(10, activation='softmax'))
```

Backpropagation in CNN

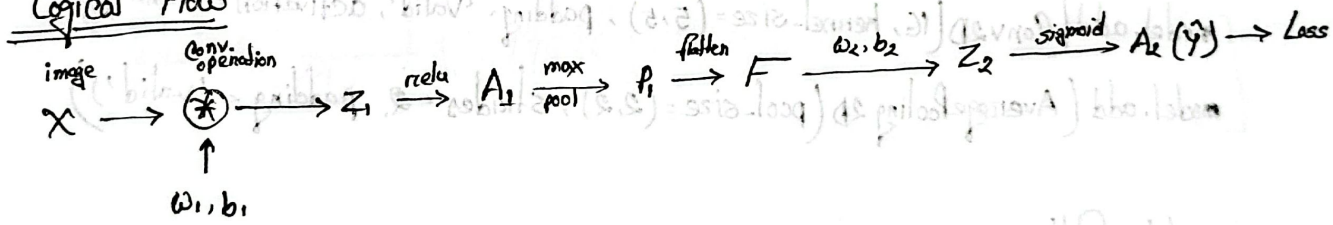


Trainable parameters

$w_1 = (3,3)$, $w_2 = (4,1)$ $= 15$

$b_1 = (1,1)$, $b_2 = (1,1)$

Logical Flow



Forword propagation

$Z_1 = conv(x, w_1 + b_1)$

$A_1 = relu(Z_1)$

$P_1 = maxpool(A_1)$

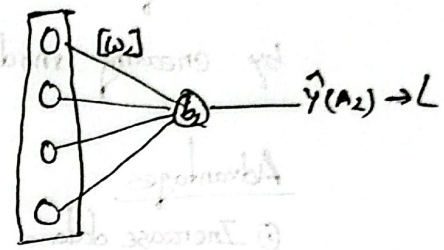
$F = flatten(P_1)$

$Z_2 = w_2 F^T + b_2$

$A_2 = \sigma(Z_2)$

From the ANN part, If we want to calculate ω_2 and b_2, process will be,

$$\omega_2 = \omega_2 - \eta\left\{\frac{\partial L}{\partial \omega_2}\right\} \quad , \quad b_2 = b_2 - \eta\left\{\frac{\partial L}{\partial b_2}\right\}$$



$$\frac{\partial L}{\partial \omega_2} = \frac{\partial L}{\partial A_2} \times \frac{\partial A_2}{\partial Z_2} \times \frac{\partial Z_2}{\partial \omega_2}$$

$$= (A_2 - \hat{y}) F^T$$

$$L = -y_i \log(\hat{y}_i) - (1-y_i)\log(1-\hat{y}_i)$$

y = actual output

$$\frac{\partial L}{\partial b_2} = \cancel{(A_2 - \hat{y})} \frac{\partial L}{\partial A_2} \times \frac{\partial A_2}{\partial Z_2} \times \frac{\partial Z_2}{\partial b_2}$$

$$= (A_2 - \hat{y})$$

For, the CNN part,

$$\omega_1 = \omega_1 - \eta\left\{\frac{\partial L}{\partial \omega_1}\right\} \quad , \quad b_1 = b_1 - \eta\left\{\frac{\partial L}{\partial b_1}\right\}$$

$$\frac{\partial L}{\partial \omega_1} = \frac{\partial L}{\partial A_2} \times \frac{\partial A_2}{\partial Z_2} \times \frac{\partial Z_2}{\partial F} \times \frac{\partial F}{\partial P_1} \times \frac{\partial P_1}{\partial A_1} \times \frac{\partial A_1}{\partial Z_1} \times \frac{\partial Z_1}{\partial \omega_1}$$

(Z_1 is a matrix)

$$= \text{conv}\left(x, \frac{\partial L}{\partial Z_1}\right)$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial A_2} \times \frac{\partial A_2}{\partial Z_2} \frac{\partial Z_2}{\partial F} \frac{\partial F}{\partial P_1} \frac{\partial P_1}{\partial A} \frac{\partial A_1}{\partial Z_1} \frac{\partial Z_1}{\partial b_1}$$

$$= \text{sum}\left(\frac{\partial L}{\partial Z_1}\right)$$

Data Augmentation

Data augmentation is the process of artificially expanding a training dataset by creating modified version of the existing images.

Advantages

① Increase data

② Reduce overfitting

```
from keras.layers import layers, models

data_augmentation = models.Sequential([
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.2),
        layers.RandomZoom(0.2),
        layers.RandomContrast(0.1),
])

model = models.Sequential()
model.add(data_augmentation)
model.add(layers.Rescaling(1./255))    // normalize

//Create model Now
```

Pretrained Models

Pretrained models are neural networks already trained on large benchmark dataset (like Imagenet) then reused to other tasks.

Adv

=> Faster training
=> Less data required
=> Higher accuracy
=> Avoide overfitting

Common pretrained CNN architectures

Model	Parameter	Use case
VGG 16 / VGG 19	~138 M	Easy to Fine-tune, Slower
ResNet50/101	~25 M	Very Stable and accurate
InceptionV3	~23M	High performance, Fast
MobileNet	~4M - 7M	For edge device

```
from keras.applications import VGG16, ResNet50, InceptionV3
from keras import layers, models

base_model = VGG16(weights='imagenet', include_top=False, input_shape=(255,255,3))
             ResNet50
             InceptionV3

base_model.trainable = False   # Freeze convolutional base

model = models.Sequential([
    base_model,
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
```

```
model = models.Sequential([
    data_augmentation,
    layers.Rescaling(1./255),
    base_model,
    .
    .
    .
])
```

Best practice

VGG → Flatten()

ResNet / Inception → GlobalAveragePooling2D()