CONVOLUTION NETWORKS

Why? (not Fully Connected Networks)

Cant do Fully Connected Networks for images, too many params

⇒ Need a network that is more memory efficient while still preserves image structure.

⇒ Enter CNN

How to design a CNN?

- Basic structure:

img ⟶ Conv ⟶ Relu ⟶ Conv ⟶ ...

- Stride
  - Trade channels for spatial resolution
  - Larger receptive field
  - More global patterns

⟶ After stride, we get: ↗ channels, ↘ resolution

For example:
- Before stride: $C \times H \times W$
- After stride: $2C \times H/2 \times W/2$

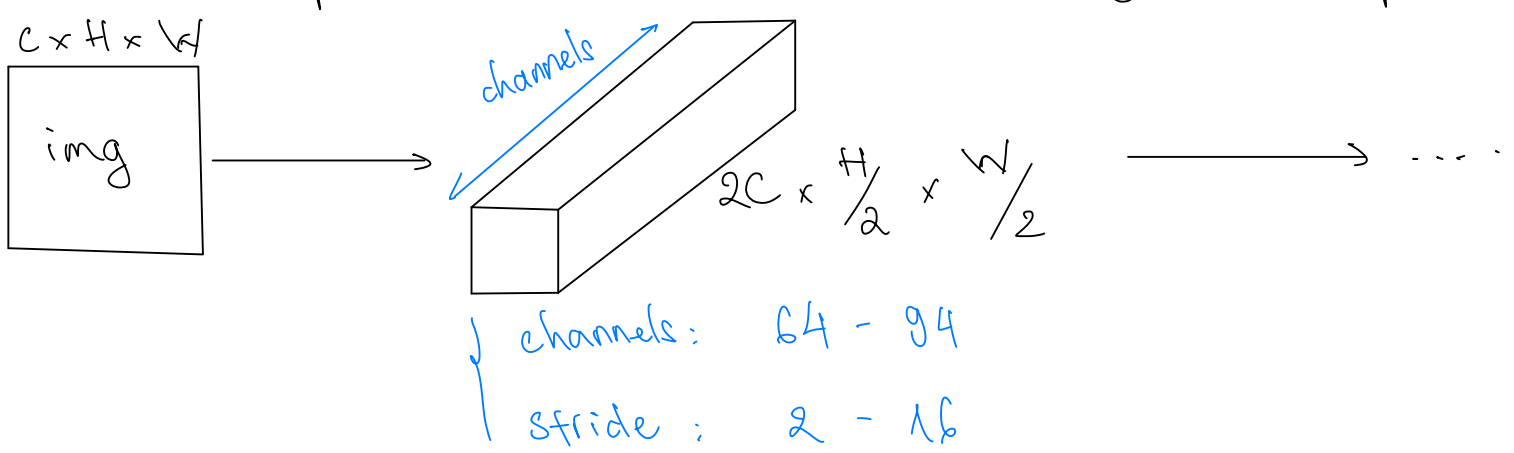
The decision of where and when to stride matters most in a CNN, since other hyperparams dont have much room to change

FAQs:

1. Should you stride throughout the network?

   No, because you'll loose information

   ↳ Instead, make the first layer "special". Meaning "blow up" the channels in exchange for spatial



2. Stride in Convolution Layer vs Pooling Layer ?

- They serve the same purpose : downsampling
- The difference is : strided Convolution Layer is trainable while strided Pooling Layer is not

For example

1. Strided Convolution Layer with Pooling

```
def simple_cnn_with_pool():
  inputs = Input(shape=(32,32,3))

  conv1 = Conv2D(filters=32, kernel_size=(3,3), strides=(1,1), activation='relu')(input
  pool1 = MaxPooling2D(pool_size=2)(conv1)

  return Model(inputs=inputs, outputs=pool1)
```

Let's see the `summary` of this model.

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 32, 32, 3)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 30, 30, 32)        896       <= Trainable parameters
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0         <= Non trainable
=================================================================
Total params: 896
Trainable params: 896
Non-trainable params: 0
_________________________________________________________________
```

2. Strided Convolution Layer without Pooling

```
def simple_cnn_without_pool():
  inputs = Input(shape=(32,32,3))

  conv1 = Conv2D(filters=32, kernel_size=(3,3), strides=(2,2), activation='relu')(input

  return Model(inputs=inputs, outputs=conv1)
```

Notice the use of `strides=(2,2)` and no pooling layer. Let's see the `summary` of this model.

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 32, 32, 3)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 15, 15, 32)        896
=================================================================
Total params: 896                                      <= Same trainable parameters
Trainable params: 896
Non-trainable params: 0
_________________________________________________________________
```

Additional question : If having pooling doesn't change the trainable parameters, why not just remove it ?

↳ Answer : It depends, based on these scenarios

- When no pooling is better :

  In the first layer of ResNet, where having 3 kernels size 3x3 have the same receptive field a 1 kernel size 7x7 with stride of 2. The latter significantly reduces the computation required.

- When having pooling is better:

  For residual block in ResNet, which requires 1x1 kernel size convolution layer, replacing the convolution layer with pooling makes it easier to backpropagate.

- Kernel size

  Except for the first layer, you want to keep your kernel size small, often 3x3 or 1x1

  Why? . Saves computation

  . More layers in sequence often better

- Repeat Patterns

  Once you found a pattern that works well, you want to keep reuse that pattern

  Why? . Save time developing, debugging, tuning

- Avoid having Linear Layer inside the network

  Why? . Linear Layer has too many parameters

- Average Pooling in the end

  Why? . Average all the features learned using CNN

  FAQs:-

  1. Average pooling before or after Classifier ?

	Average → Classifier	Classifier → Average
Pros	. Less params in Classifier . Regularization → ↓ overfitting . Easier to understand	. Learn spatial info of each class . Preserve more spatial info
Cons	. lose spatial information . classifier operates more abstract (because it use the average)	. Increase params . More prone to overfitting due to increase params