IEEE FCI - Helwan Student Branch

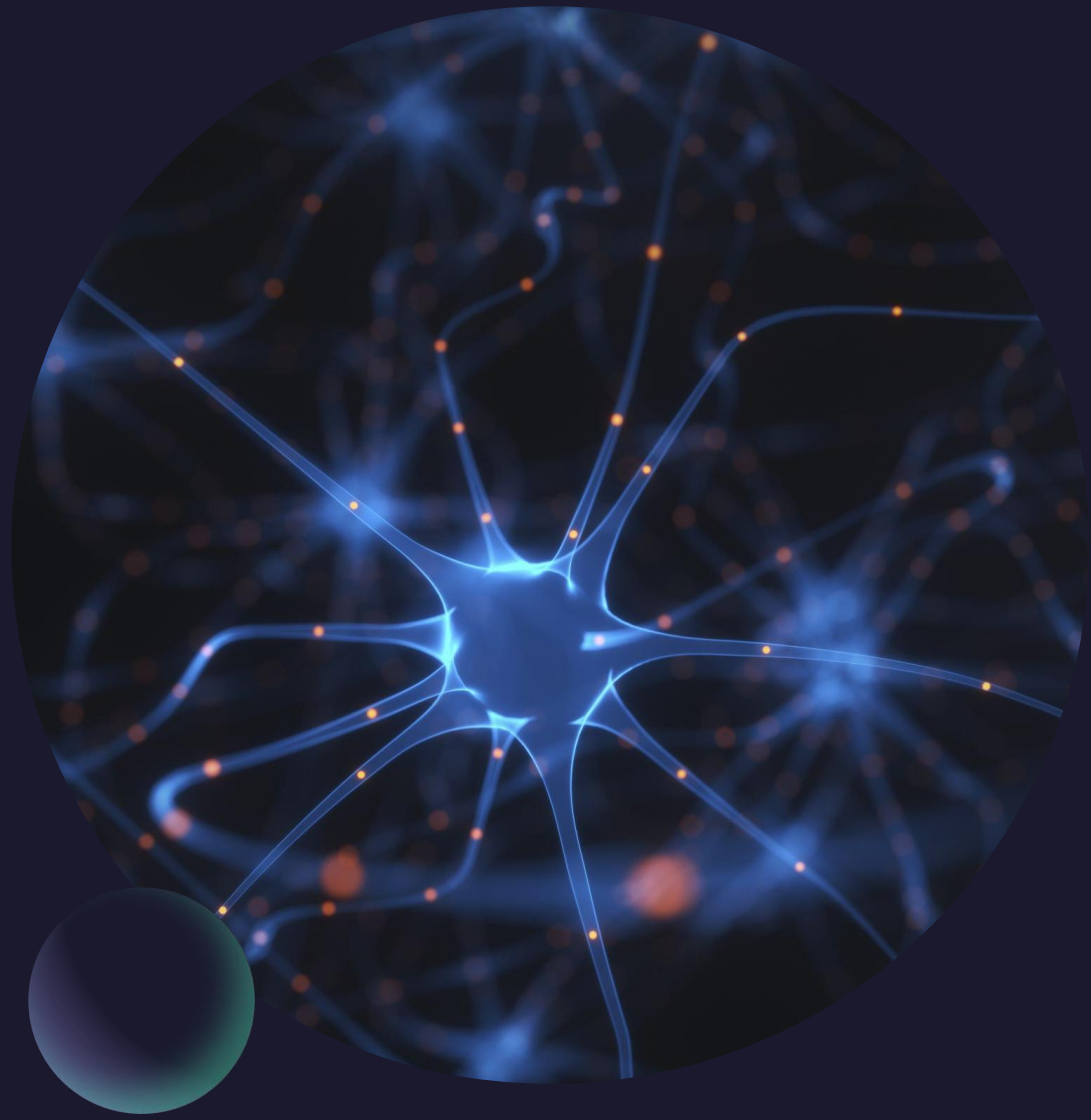
Convolution Neural Networks (CNN)

Hossam Ahmed

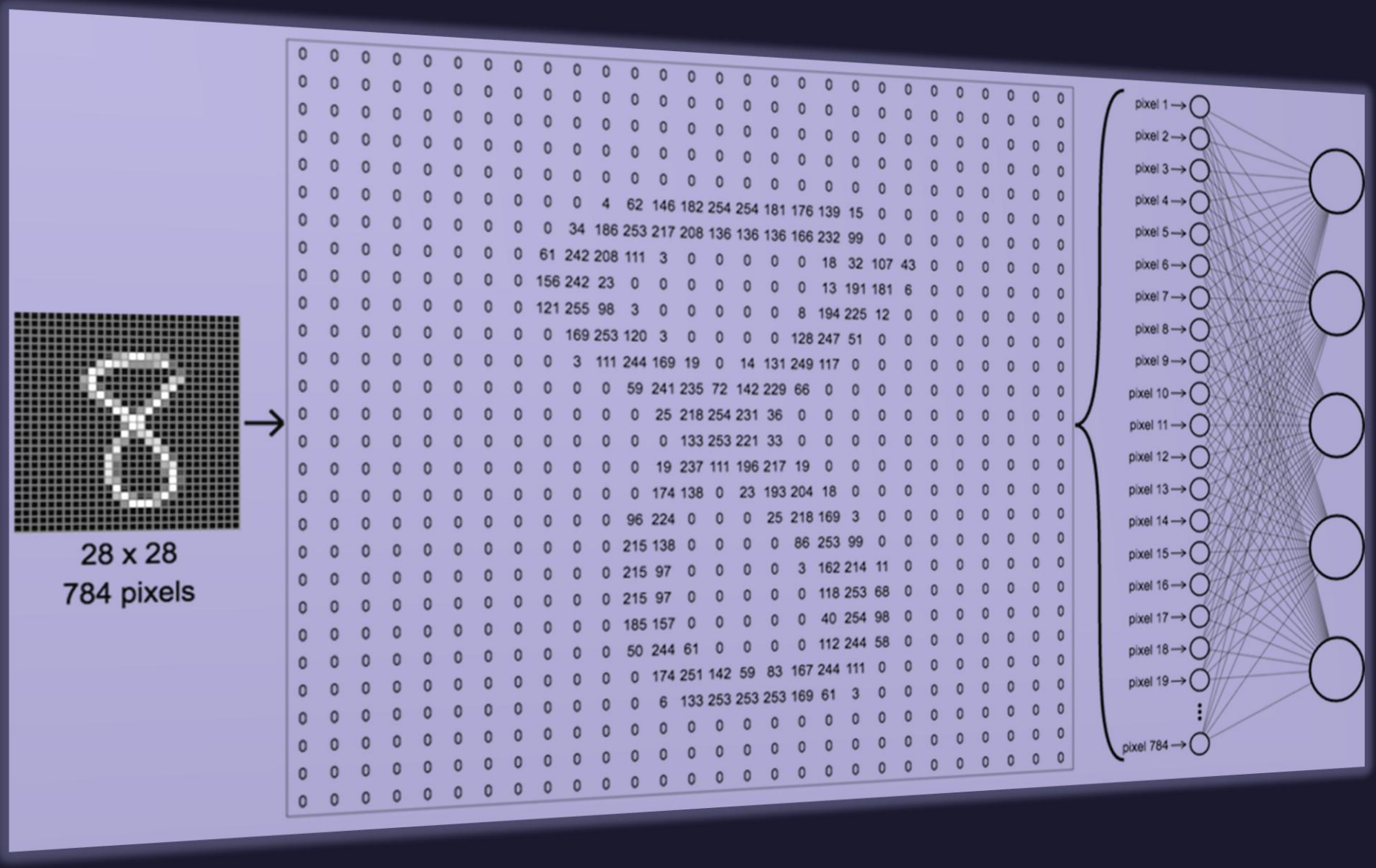
Mario Mamdouh

Introduction

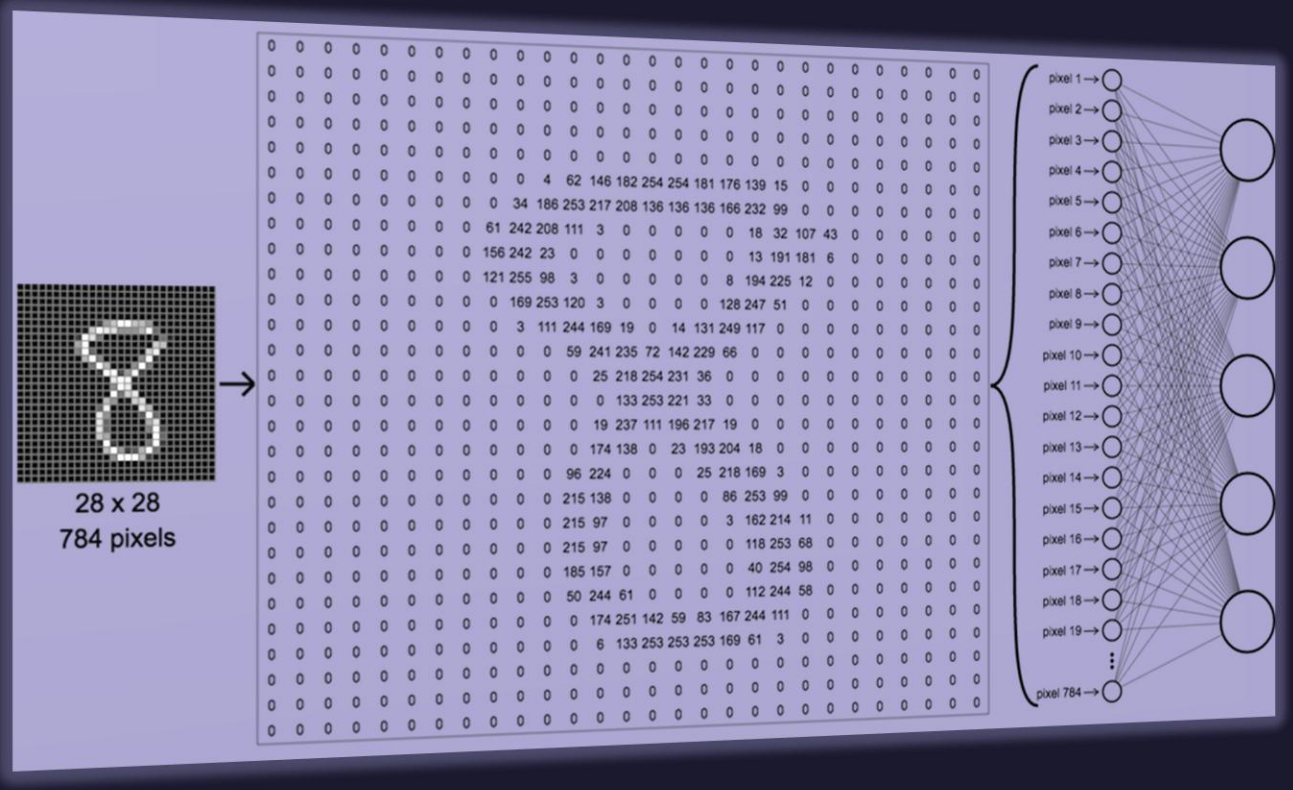
- A Convolutional Neural Network (CNN or ConvNet) is a type of deep neural network specifically designed for processing structured grid data, such as images and video.
- good with image, speech, or audio signal inputs



Recall ANN dealing with images



Recall ANN dealing with images

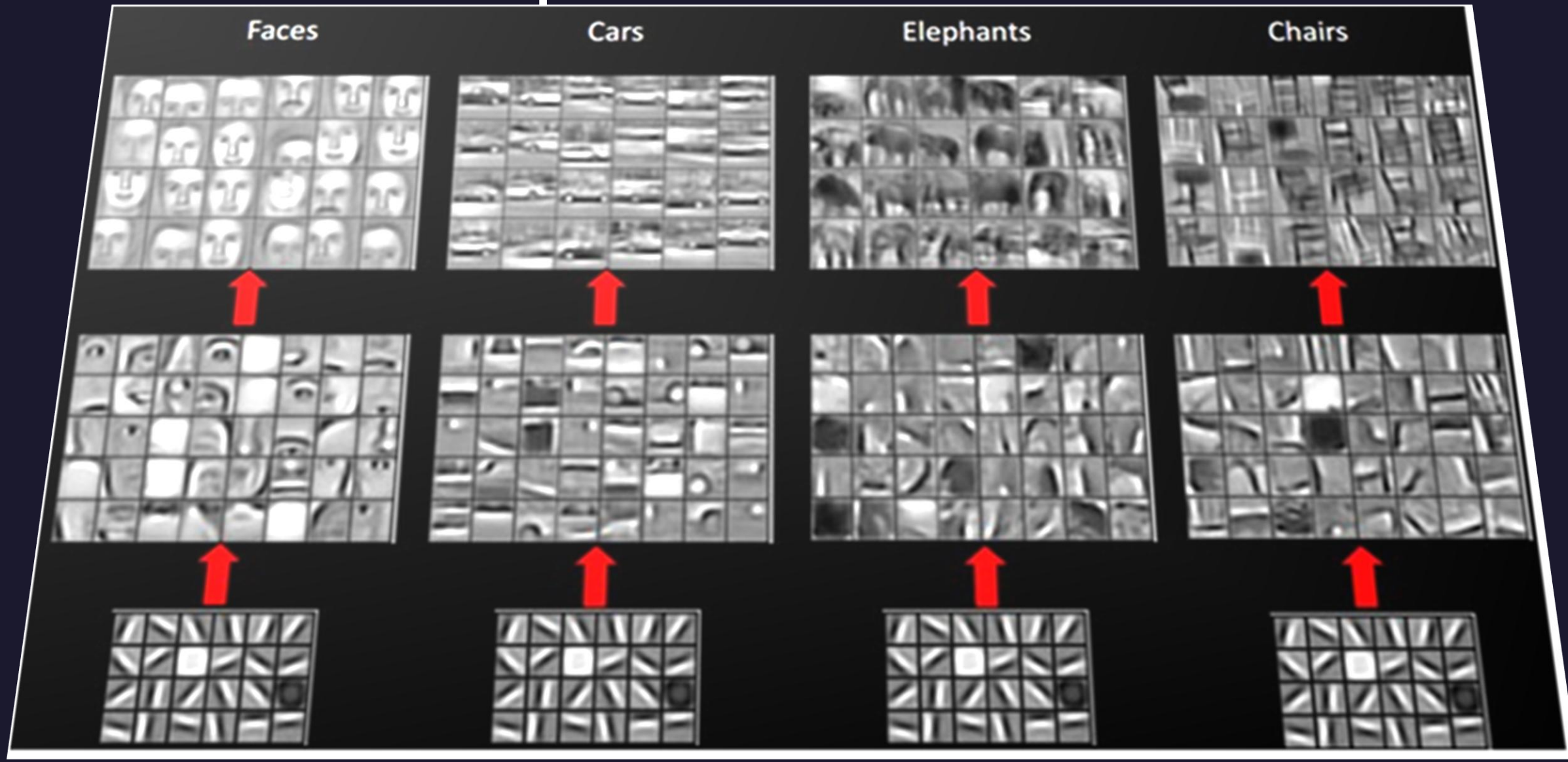


- 28×28 image would need input layer of 785 nodes imagine how large images would be
  - HD 1280 x 720 pixels -> 921600 as input nodes
- Losing the meaning of shapes as we take each pixel as separated feature not related with other pixels.
- If we slightly rotate the image, it would be totally a new input features (Translation invariance).
  - Rotation ,Shifting , Flipping, Shearing and Zooming
  - Linear Transformations
- ANNs, can be computationally expensive and memory-intensive due to the large number of parameters in fully connected layers.

CNN here to help

- CNNs can capture **spatial hierarchies** and local patterns in images more effectively.
- Spatial often refer to Width, Hight dimensions excluding ~~Depth~~
- **Translation Invariance they can recognize patterns regardless of their position in the input space.**
- **Hierarchical Feature Learning CNN architectures typically consist of multiple layers, allowing the network to learn hierarchical features. Lower layers capture basic features like edges and textures, while higher layers combine these features to represent more complex patterns and objects.**
- **Each Convolution layer can extract features from input space**
  - **If the input is Pixels the convolution layer can catch the edges .**
  - **The edges as input space followed by another convolution layer that catch higher level of features like shapes.**
  - **The shapes can be used as input for another convolution layer to catch higher level of features like objects.**

CNN here to help



ConvNet Layers

3 main types of Layers

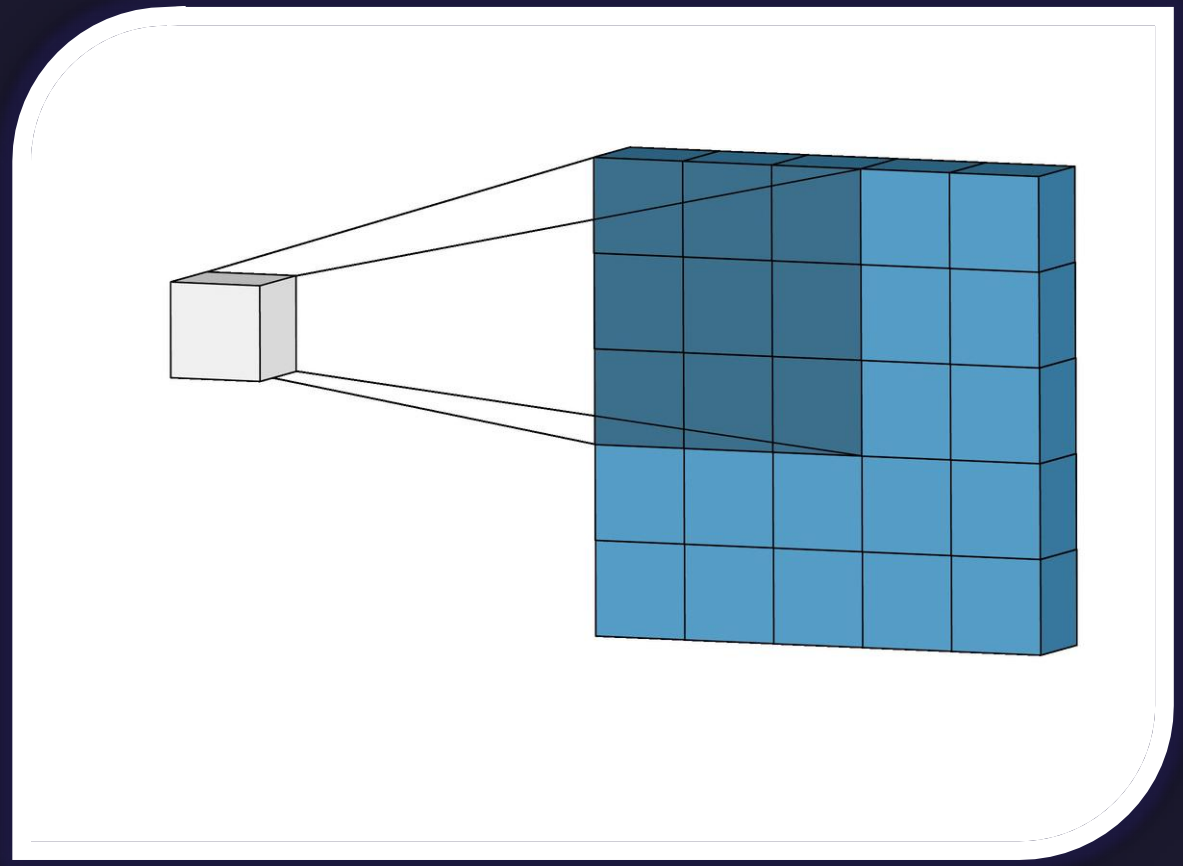
COVOLUTION LAYER

POOLING LAYER

FULY CONNECTED LAYER

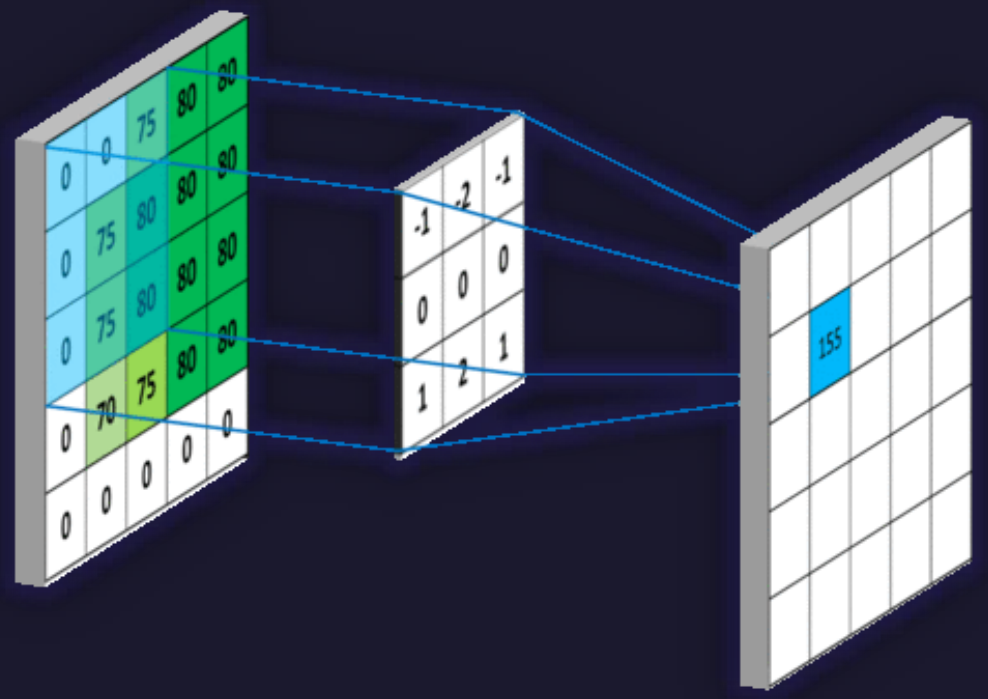
Convolution layer

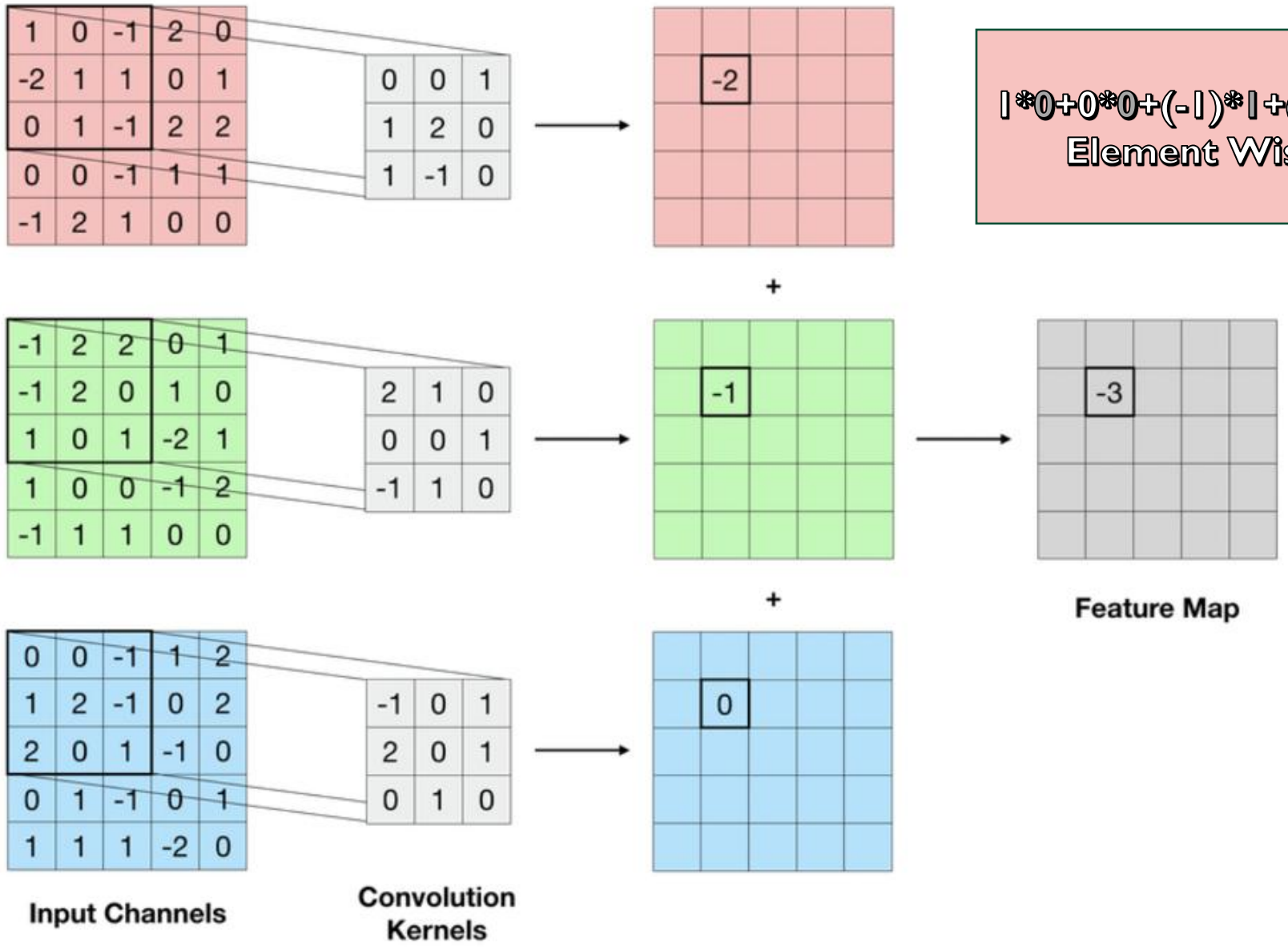
- The Conv layer is the core building block of a CNN that does most of the **computational heavy lifting**.
- Convolution layer is a **Parametric layer**
  - Set of learnable filters and their configurations



Convolution layer

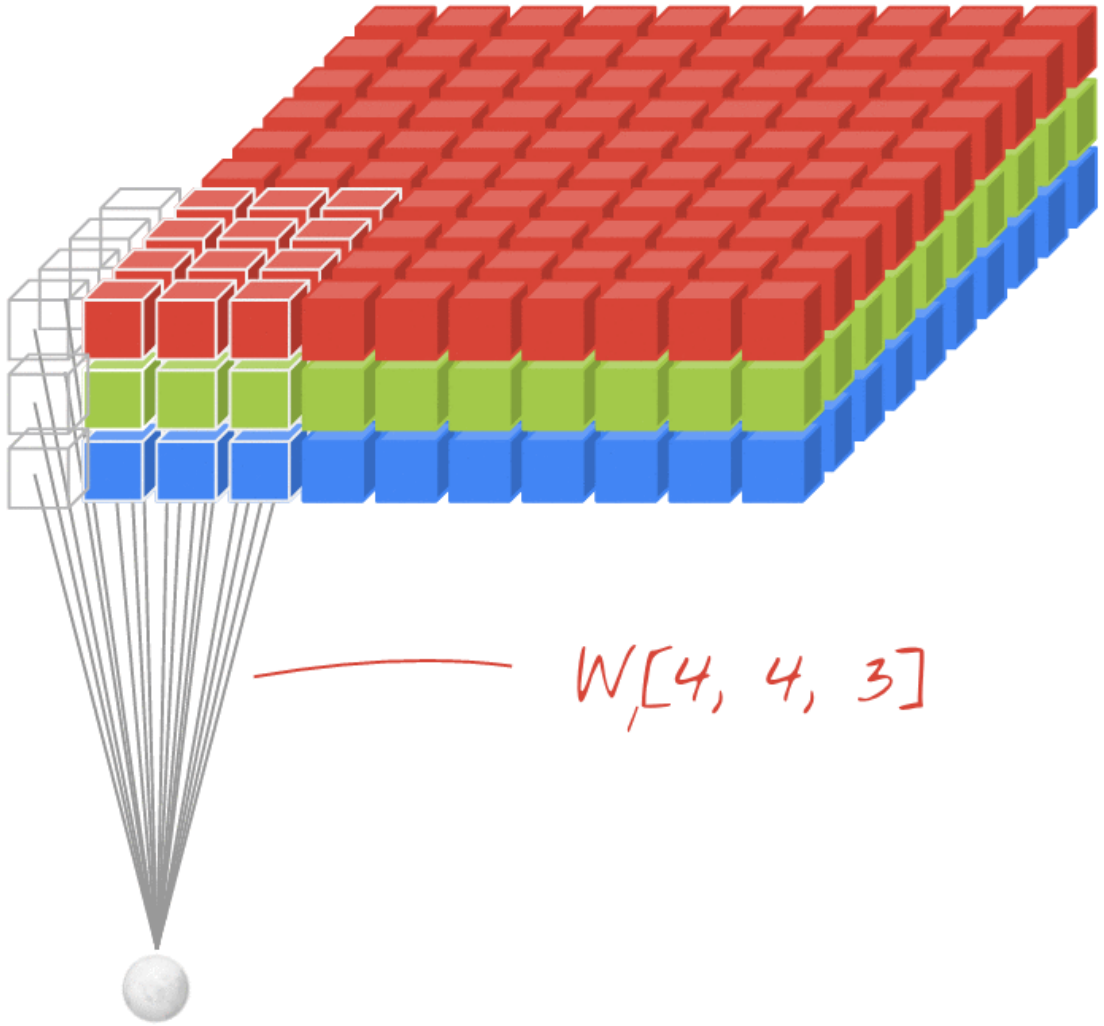
- The Conv layer is the core building block of a CNN that does most of the **computational heavy lifting**.
- Convolution layer is a Parametric layer
  - Set of learnable filters and their configurations
  - The Filter has weights
  - The Filter slide (Convolve) over the input



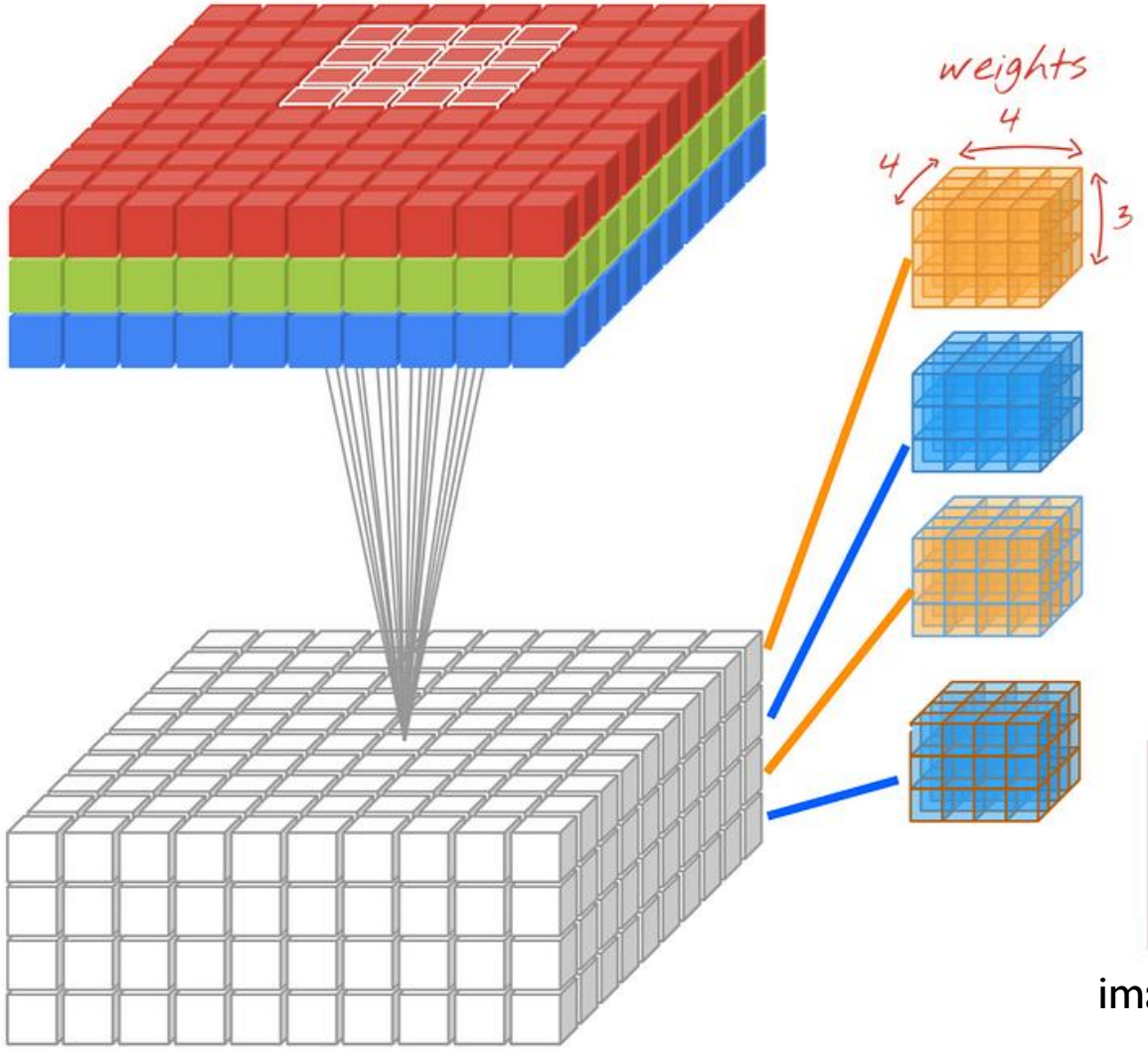


I*0+0*0+(-I)*I+(-2)*I+I*2+I*0+0*I+I*(-I)+(-I)*0=-2
Element Wise multiplication and summation

- **Note : each channel (depth) have its own filter but for spatial input (Width & Height) they share the same weights for the filter**
- **Note Feature map can be stacked that mean applying another set of filters over the 3 channels**
- **These filter would share parameters over Width and Hight (Spatially)**

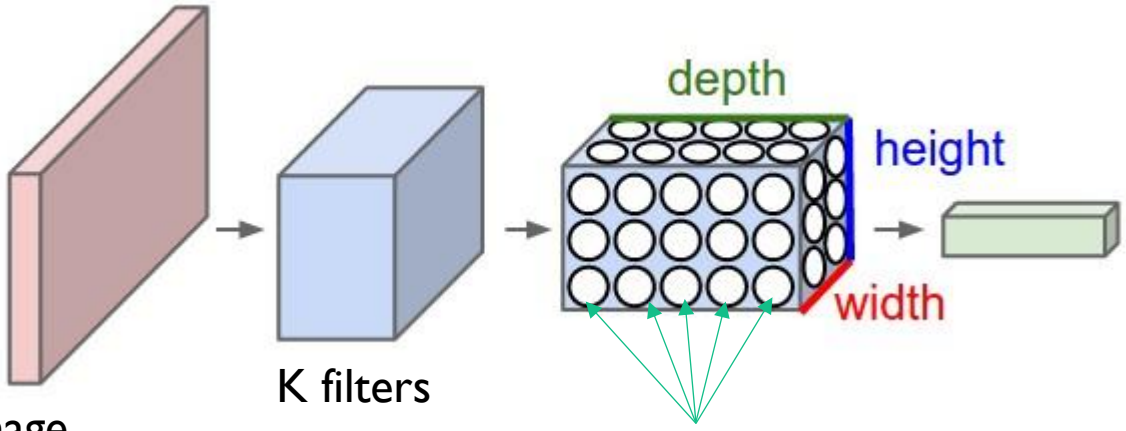


- **Note : each channel (depth) have its own filter but for spatial input (Width & Height) they share the same weights for the filter**
- **Note Feature map can be stacked that mean applying another set of filters over the 3 channels**
- **These filter would share parameters over Width and Hight (Spatially)**
- **W1 and W2 are different filter weights**



More filters, more output channels.

More filters more output depth , number of filters would be denoted as K



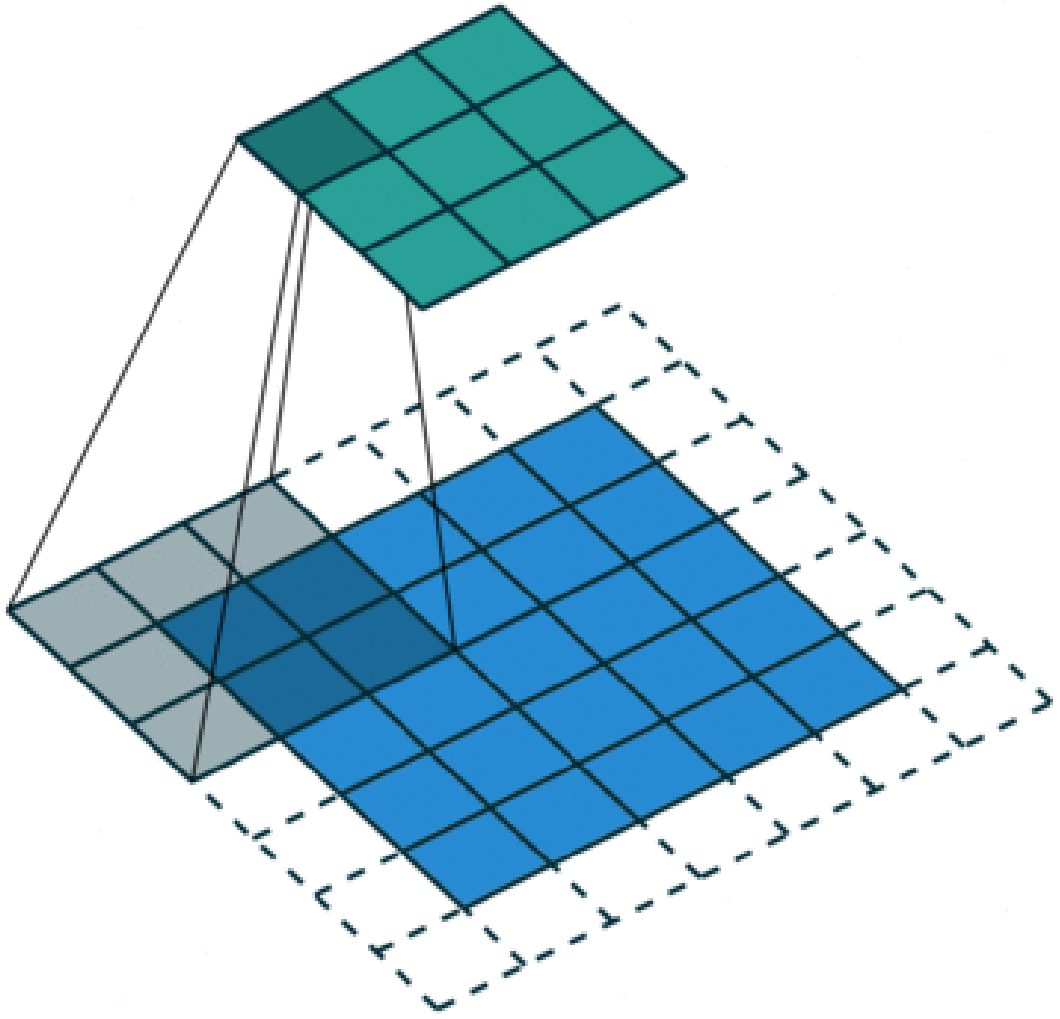
image

K filters

K filters result in k depth activation ma

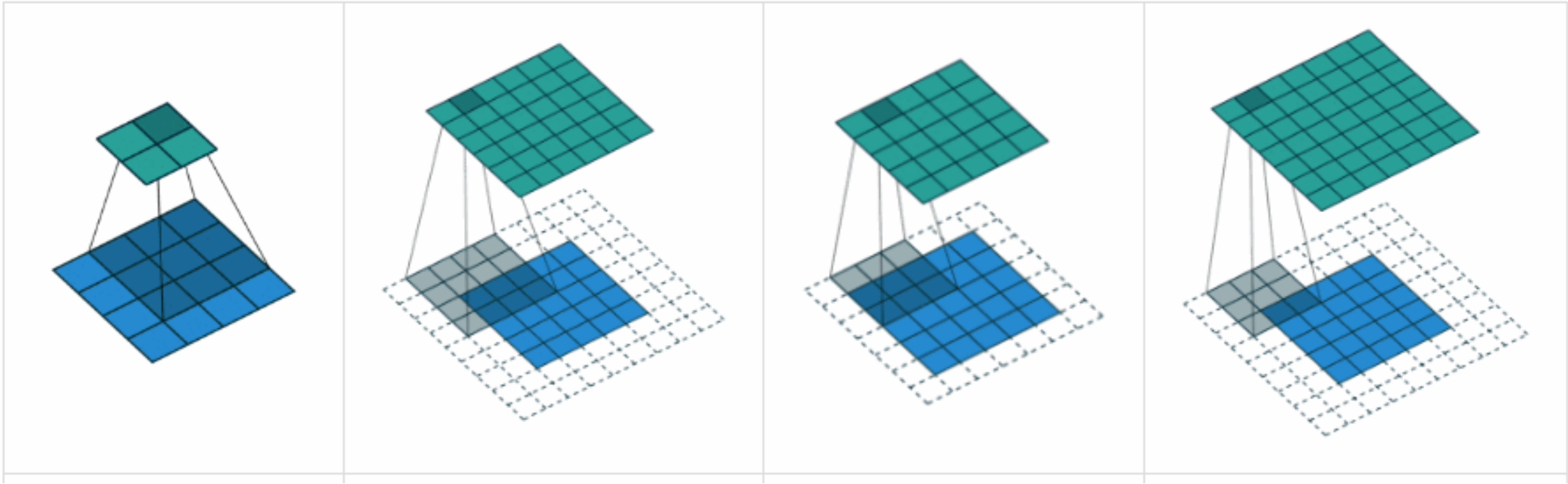
Convolution layer

- The Conv layer is the core building block of a CNN that does most of the **computational heavy lifting**.
- Convolution layer is a Parametric layer
  - Set of learnable filters and their configurations
  - The Filter has weights
  - The Filter slide (Convolve) over the input
  - We can apply a K filters that result in output of depth K also we can apply different set of filters over the depth color channels
  - We can use **Step (Stride)** as another parameter
  - Stride indicates how many pixels the kernel should be shifted over at a time
  - **Zero Padding** also may be used



Padding involves adding extra pixels around the input image, creating a "padding" of zeros or other constant values

More Padding give us larger or equal output Width and Hight



Convolution layer Spatial arrangement.

- Three hyperparameters control the size of the output volume: the depth, stride and zero-padding

1. Depth : corresponds to the number of filters we would like to use, each learning to look for something different in the input
    - set of neurons that are all looking at the same region of the input as a depth column(fibre)
2. Stride with which we slide the filter, the higher we jump the smaller output volumes we get.
3. Zero-padding:
    - Prevention of Shrinking Output Size
        - spatial dimensions of the feature maps decrease with each convolutional layer.
        - Padding prevent the extreme reduction of the size of features map
    - Preservation of Spatial Information
        - Like edges

Convolution layer Spatial arrangement.

- **Zero-padding:**
  - Preservation of Spatial Information
    - Like edges
    - Padding allows the pixels at the corners and edges of the input to be aligned with the pixels at the center of the filter
    - This alignment is important for learning features at different spatial scales.
  - Handling Various Input Sizes
- To compute the spatial(width and Hight) size of the output as a Function of
  - W input volume size (width) by symmetry width and height are the same most of the time
  - F receptive field(filter) size of the Conv Layer neurons
  - S step (stride)
  - P amount of zero padding

Convolution layer **Spatial arrangement.**

- To compute the spatial(width and Hight) size of the output as a Function of
  - W input volume size (width) by symmetry width and height are the same most of the time
  - F receptive field(filter) size of the Conv Layer neurons
  - S step (stride)
  - P amount of zero padding

$$\frac{w - f + 2P}{S} + 1$$

Convolution layer **Spatial arrangement.**

- Example
- Input [227×227×3]
- Filters [11×11×3] 3 is K number of filters
- Stride 4
- No Padding

$$\frac{w - f + 2P}{S} + 1$$

Convolution layer **Spatial arrangement.**

- Example
- Input [227×227×3]
- Filters [11×11×3] 3 is K number of filters
- Stride 4
- No Padding

$$\frac{227 - 11 + 2 * 0}{4} + 1 = 55$$

Convolution layer **Summary.**

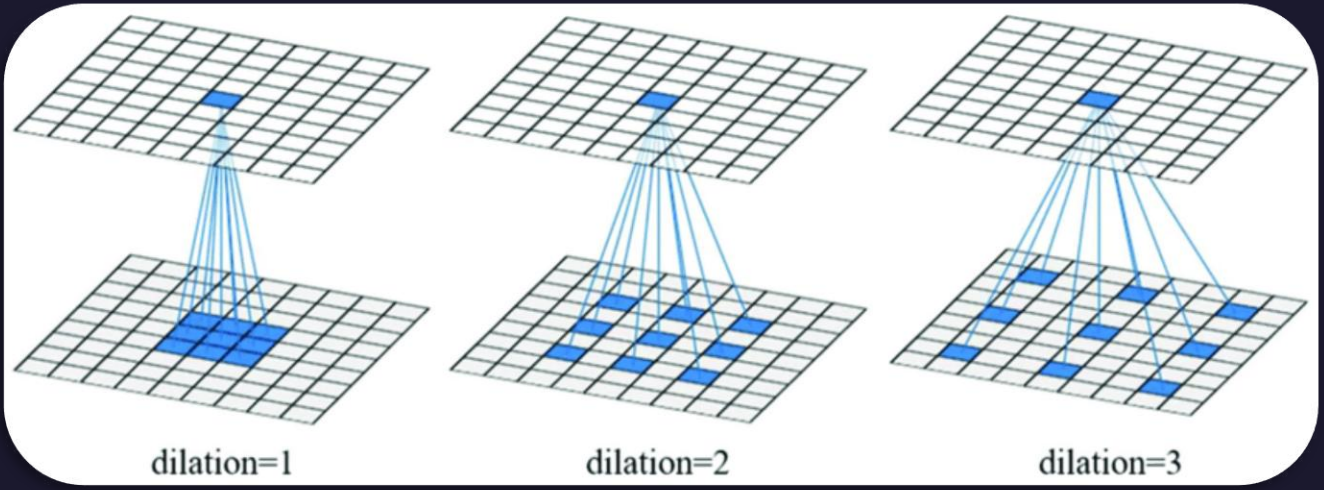
- Accept input of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters **K**
  - Filter Size **F**
  - Stride **S**
  - Padding **P**
- Produces a volume of size $W_2 \times H_2 \times D_2$ where :
  - $W_2 = \frac{W_1 - f + 2P}{S} + 1$
  - $H_2 = \frac{H_1 - f + 2P}{S} + 1$
  - $D_2 = D_1$

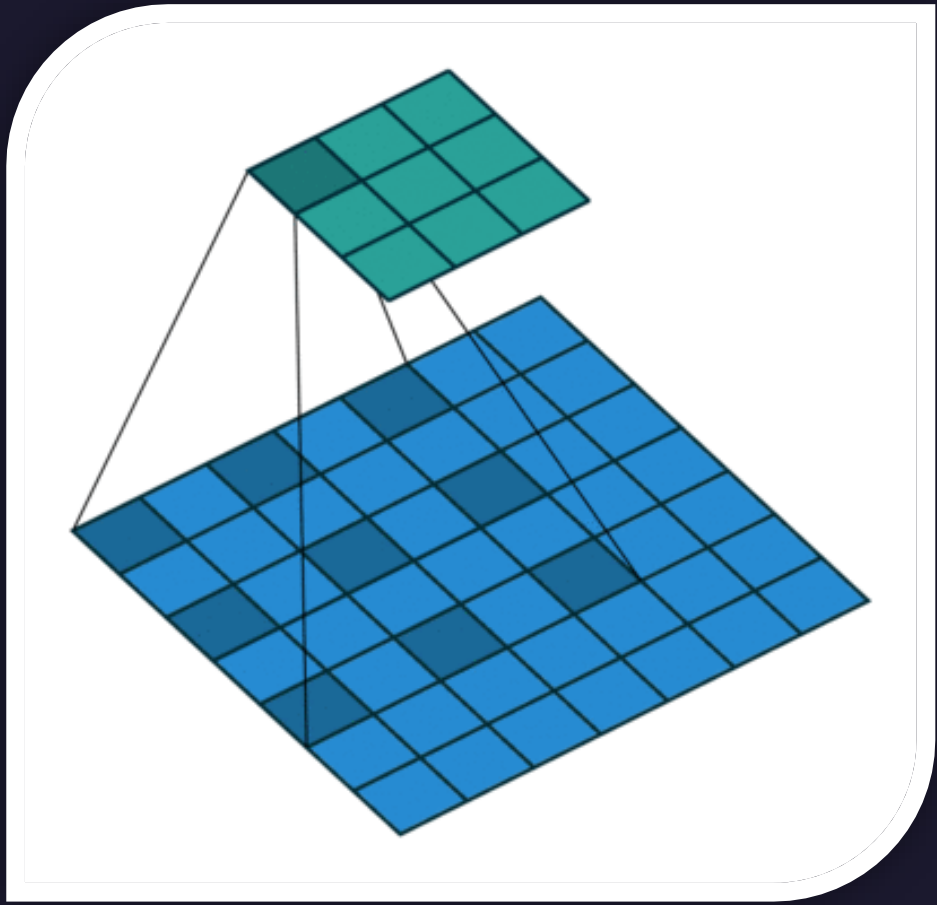
Convolution layer Summary.

- Produces a volume of size $W_2 \times H_2 \times D_2$ where :
  - $W_2 = \frac{W_1 - f + 2P}{S} + 1$
  - $H_2 = \frac{H_1 - f + 2P}{S} + 1$
  - $D_2 = D_1$
- With Parameter Sharing it introduce $F.F.D_1$ weights per filter for a total of $K \times (F.F.D_1)$
  - Think about a filter of 3×3 applied over 3 channels and there are K different filters
- Backpropagation is a Convolution with spatially flipped filters
- Dilated convolutions : one more hyperparameter to the CONV layer, it's possible to have filters that have spaces between each cell, called dilation.
- After the Conv Layer we apply activation function like ReLU max(0, x)

Convolution layer **Dilation.**

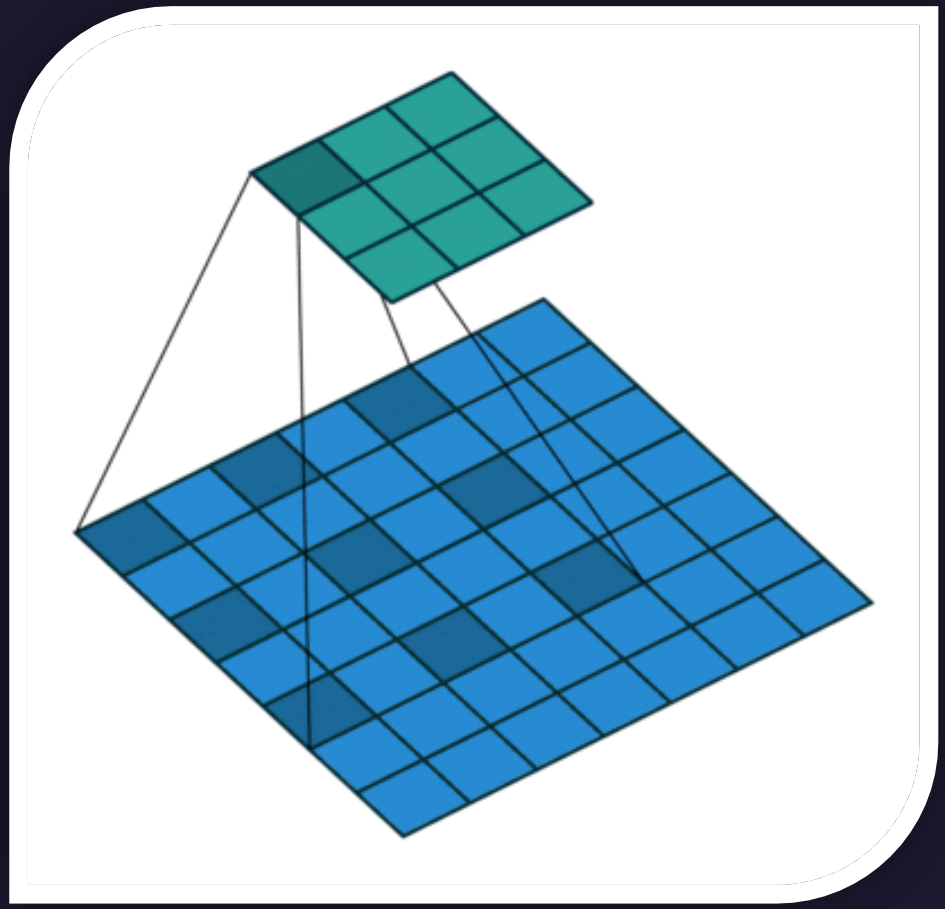
- **Dilated convolutions : one more hyperparameter to the CONV layer, it's possible to have filters that have spaces between each cell, called dilation.**





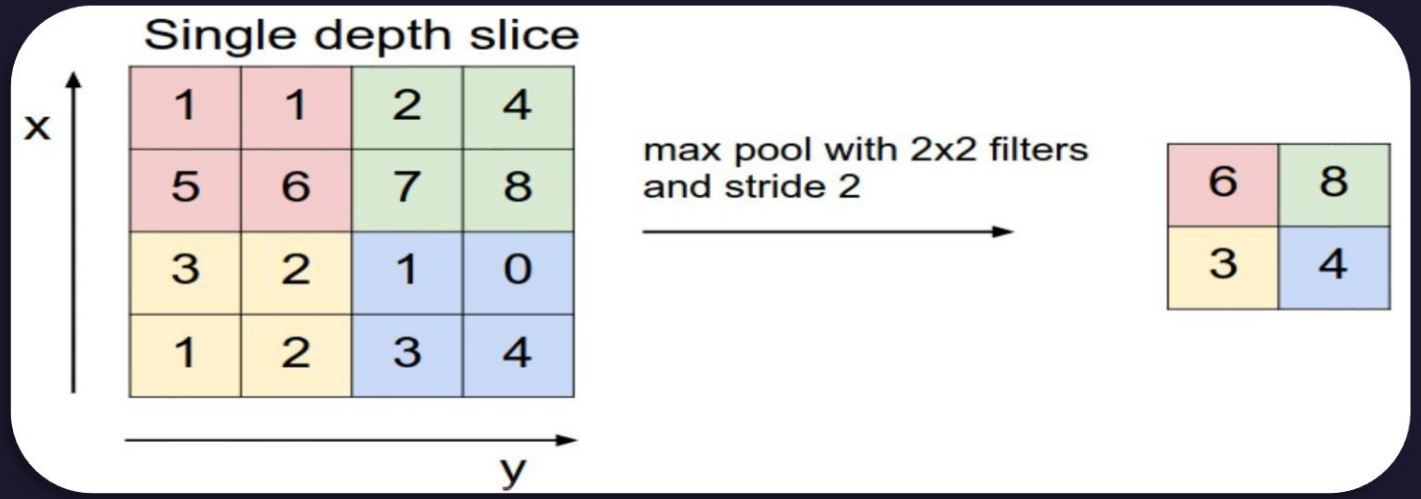
Convolution layer **Dilation.**

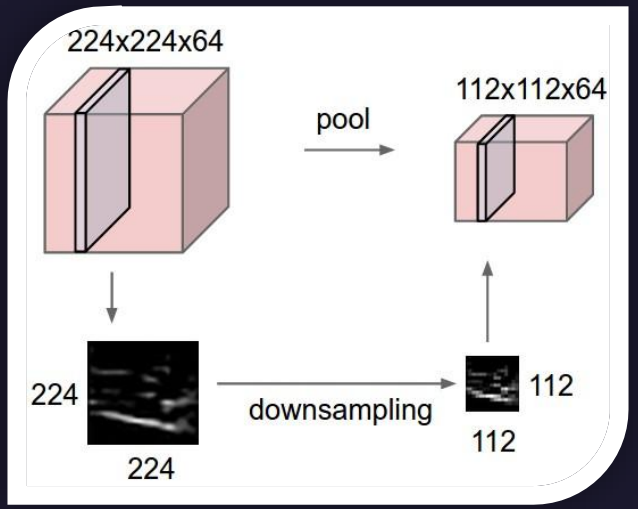
- Dilated convolutions : one more hyperparameter to the CONV layer, it's possible to have filters that have spaces between each cell, called dilation.
- Dilation allows the convolutional layer to cover a larger area of the input, capturing information from a broader spatial context. This is beneficial for tasks where understanding global patterns or context is crucial.
- Traditional methods for enlarging the receptive field, such as using larger filter sizes, can lead to a reduction in spatial resolution. Dilation provides a way to enlarge the receptive field without sacrificing resolution.
- Dilation can be used at different layers of a network to enable multi-scale feature learning



Pooling layer

- It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture. Its function is to progressively **reduce the spatial size** of the representation to **reduce the amount of parameters and computation in the network**, and hence to also **control overfitting**.
- The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation.
- The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations





Pooling layer summary

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - F Filter size
  - S Stride
- Produces a volume of size $W_2 \times H_2 \times D_2$ where
  - $W_2 = \frac{W_1 - f}{S} + 1$
  - $H_2 = \frac{H_1 - f}{S} + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- For Pooling layers, it is not common to pad the input using zero-padding.

Pooling layer summary

- Produces a volume of size $W_2 \times H_2 \times D_2$ where
  - $W_2 = \frac{W_1 - f}{S} + 1$
  - $H_2 = \frac{H_1 - f}{S} + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- For Pooling layers, it is not common to pad the input using zero-padding.
- A pooling layer with F=3,S=2 (also called overlapping pooling), and more commonly F=2,S=2 Pooling sizes with larger receptive fields are too destructive.
- General pooling :can also perform other functions, such as average pooling or even L2-norm pooling.

Pooling layer Getting rid of pooling

Some people preferer not to use Pooling Layers

To reduce the size of the representation they suggest using larger stride in CONV layer once in a while.

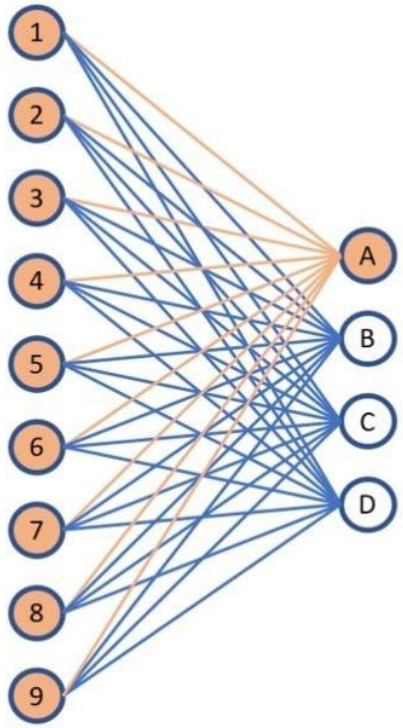
Discarding pooling layers has also been found to be important in training good generative models

This comes form a paper called "Striving for simplicity : *The ALL Convolution Net*"

Fully Connected Layer

- Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks
- the only difference between FC and CONV layers is that the neurons in the CONV layer are connected only to a local region in the input (not Dense), and that many of the neurons in a CONV volume share parameters.
- However, the neurons in both layers still compute dot products, so their functional form is identical. Therefore, it turns out that it's possible to convert between FC and CONV layers

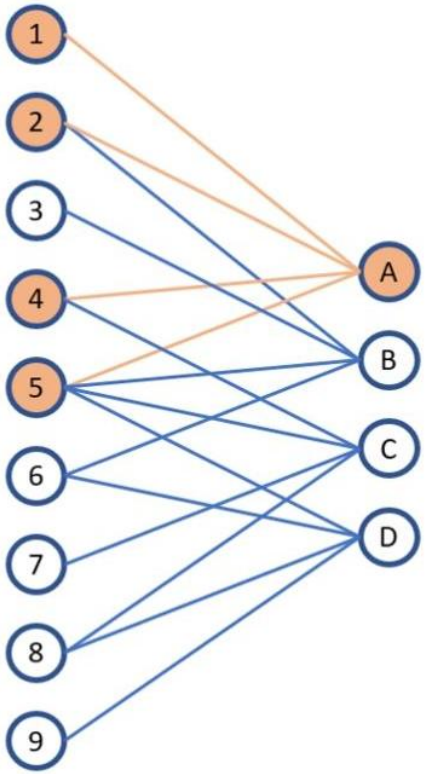
Fully Connected Layer



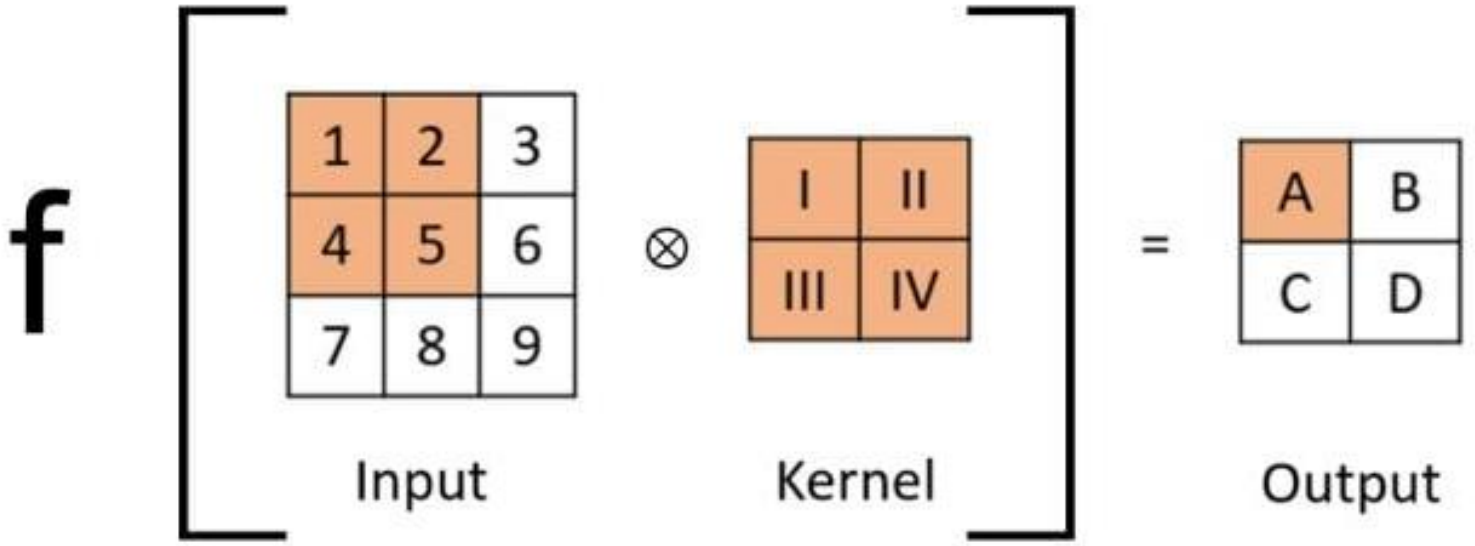
Each neuron in the second layer got a 9 links (weights)

$$f\left[\text{INPUT VECTOR}\ (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)\ \text{1x9} \cdot \text{WEIGHTS MATRIX}\ \text{9x4}\right] = \text{OUTPUT VECTOR}\ (A\ B\ C\ D)\ \text{1x4}$$

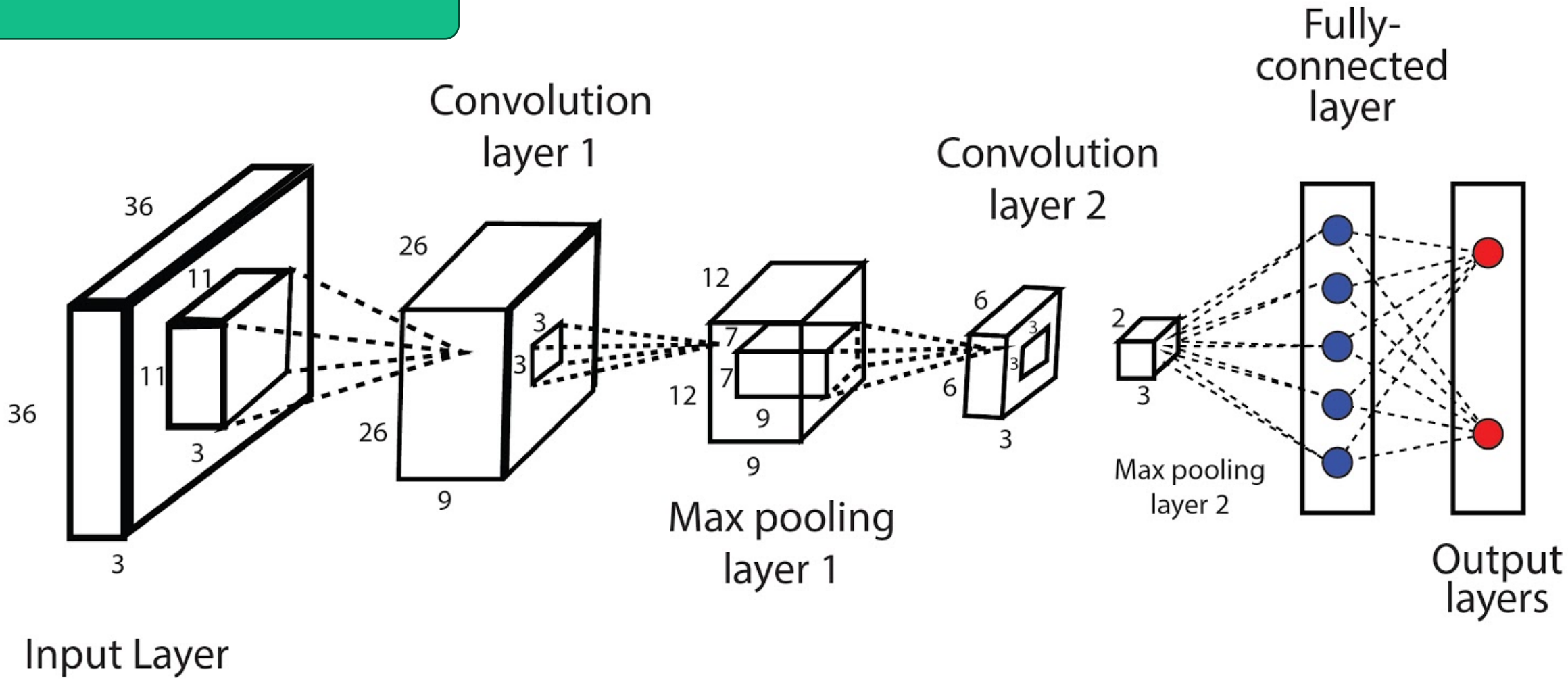
Fully Connected Layer



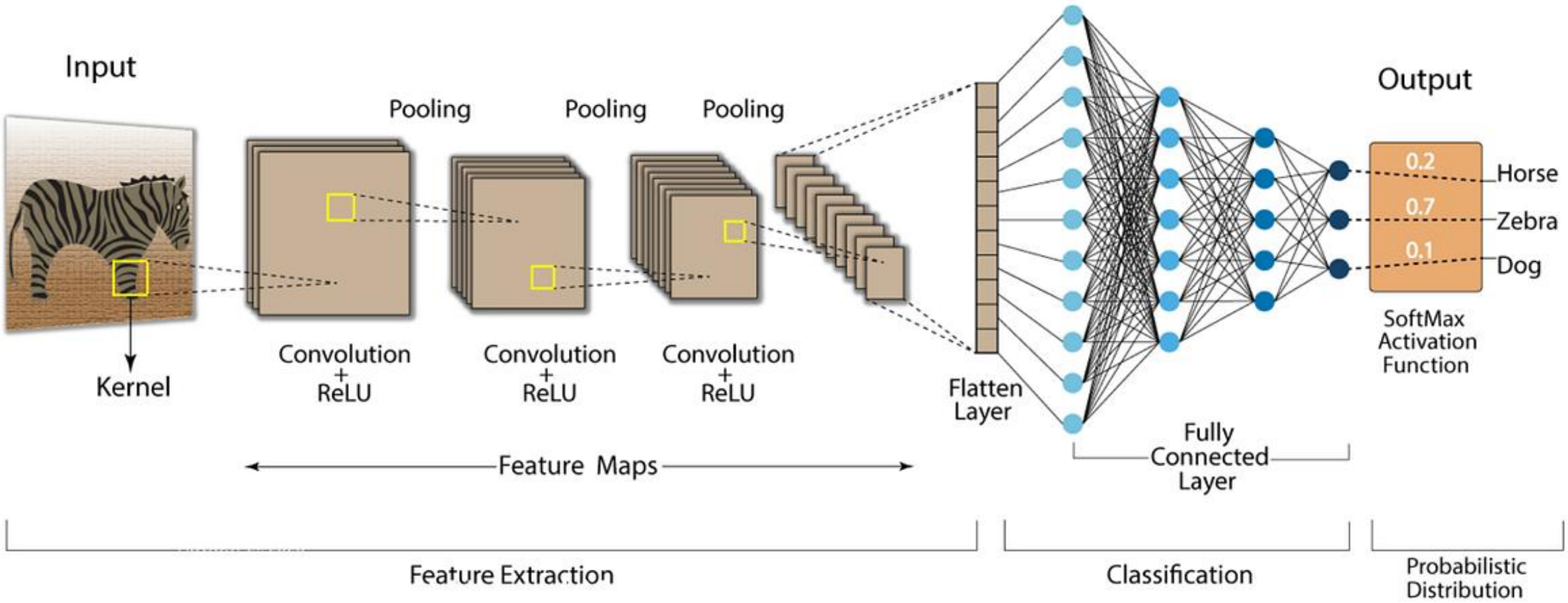
Note this is CONV representation each neuron relates to a local region not a Dense



$$\frac{36 - 11 + 2 * 0}{1} + 1 = 26$$



Convolution Neural Network (CNN)



Convention

- The most common form of a ConvNet architecture stacks a few CONV-RELU layers, follows them with POOL layers, and repeats this pattern until the image has been merged spatially to a small size.
- At some point, it is common to transition to fully-connected layers. The last fully-connected layer holds the output, such as the class scores. In other words, the most common ConvNet architecture follows the pattern:

INPUT ➡️ [[CONV ➡️ RELU] 🔁 N ➡️ POOL ❓] 🔁 M ➡️ [FC ➡️ RELU] 🔁 K ➡️ FC

- 🔁 (*) REPRESENT LOOP
- N, M AND K NUMBER OF ITERATION FOR EACH LOOP
- POOL ❓ indicates an optional pooling layer.

Convention

INPUT ➡️ [[CONV ➡️ RELU] 🔁 N ➡️ POOL ❓] 🔁 M ➡️ [FC ➡️ RELU] 🔁 K ➡️ FC

INPUT ➡️ FC , N = K = M = 0 implement a linear classifier

- Here we see that there is a single CONV layer between every POOL layer.

INPUT ➡️ [CONV ➡️ RELUE ➡️ POOL] 🔁 2 ➡️ FC ➡️ RELU ➡️ FC

Convention

- Here we see two CONV layers stacked before every POOL layer. This is generally a good idea for larger and deeper networks, because multiple stacked CONV layers can develop more complex features of the input volume before the destructive pooling operation.

INPUT ➡️ [CONV ➡️ RELUE ➡️ CONV ➡️ RELU ➡️ POOL] 🔁 3 ➡️ [FC ➡️ RELU] 🔁 2 ➡️ FC

- In practice: use whatever works best on ImageNet; download a pretrained model and finetune it on your data. You should rarely ever have to train a ConvNet from scratch or design one from scratch

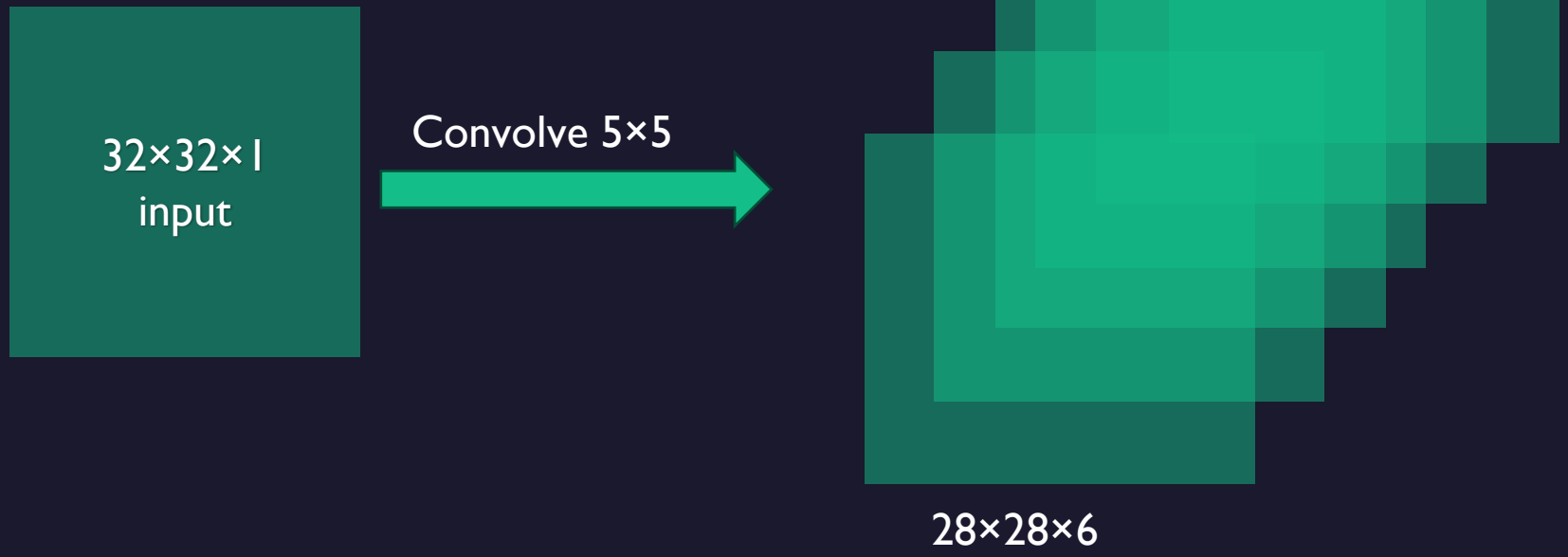
LeNET

- LeNet-5 consists of seven layers, including three convolutional layers and two fully connected layers.
- the architecture of Lenet-5. The network has 5 layers with learnable parameters and hence named Lenet-5.
  - Learnable layers layers that contain parameters (weights and biases) that are adjusted during the training process
  - FC, CONV, Recurrent layers , Embedding layers , Normalization layers and attention layers

INPUT ➡️ [CONV(6, 5x5, tanh) ➡️ AVG_POOL(2x2)]
➡️ [CONV(16, 5x5, tanh) ➡️ AVG_POOL(2x2)]
➡️ [CONV(120, 5x5, tanh)] ➡️ FC(84, tanh) ➡️ FC(10, softmax)

LeNET

INPUT ➡️ [CONV(6, 5x5, tanh)…



$$\frac{w - f + 2P}{S} + 1$$

$$\frac{32 - 5 + 2 * 0}{1} + 1 = 28$$

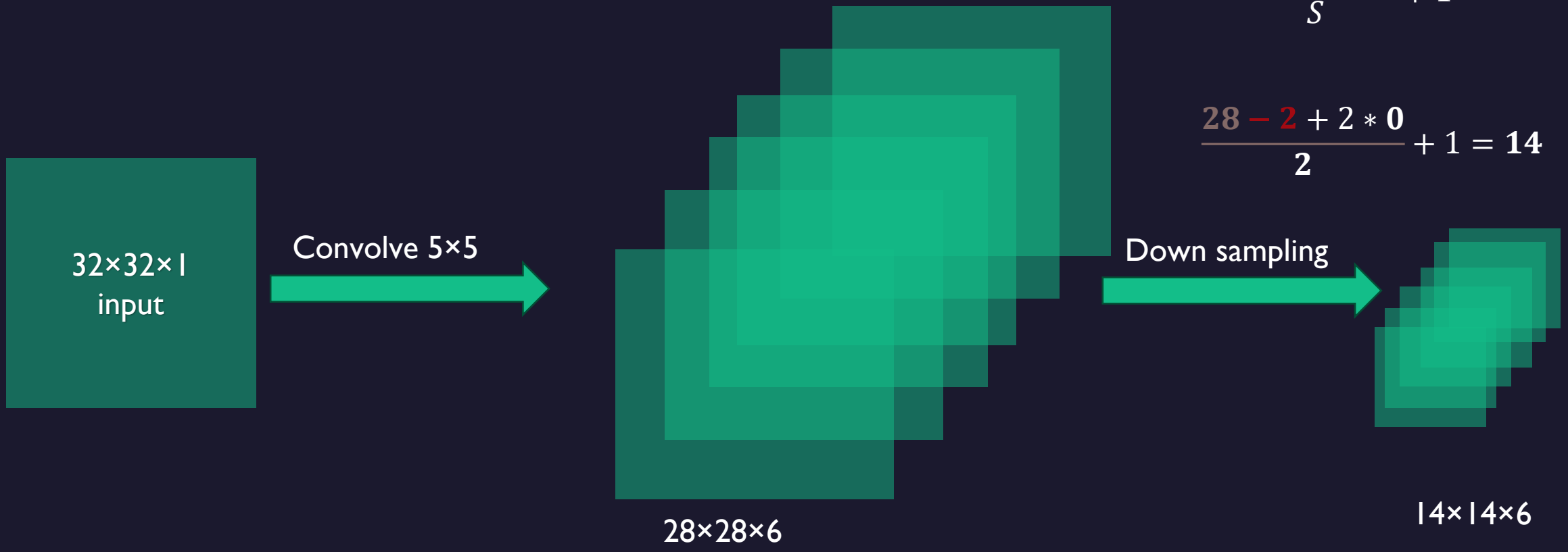
LeNET

INPUT ➡️ [CONV(6, 5x5, tanh) ➡️ AVG_POOL(2x2)]

Helwan
t Branch

$$\frac{w - f + 2P}{S} + 1$$

$$\frac{28 - 2 + 2 * 0}{2} + 1 = 14$$

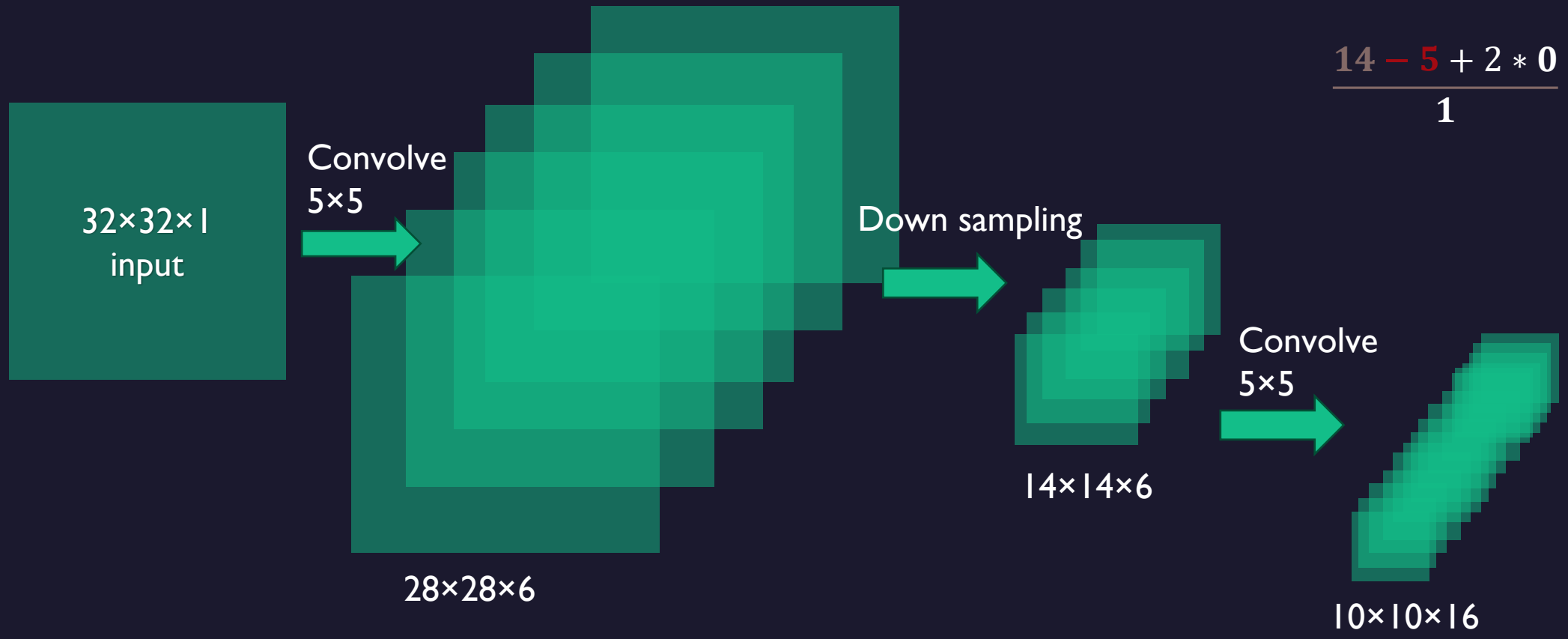


LeNET

INPUT ➡️ [CONV(6, 5x5, tanh) ➡️ AVG_POOL(2x2)]
➡️ [CONV(16, 5x5, tanh)

$$\frac{w - f + 2P}{S} + 1$$

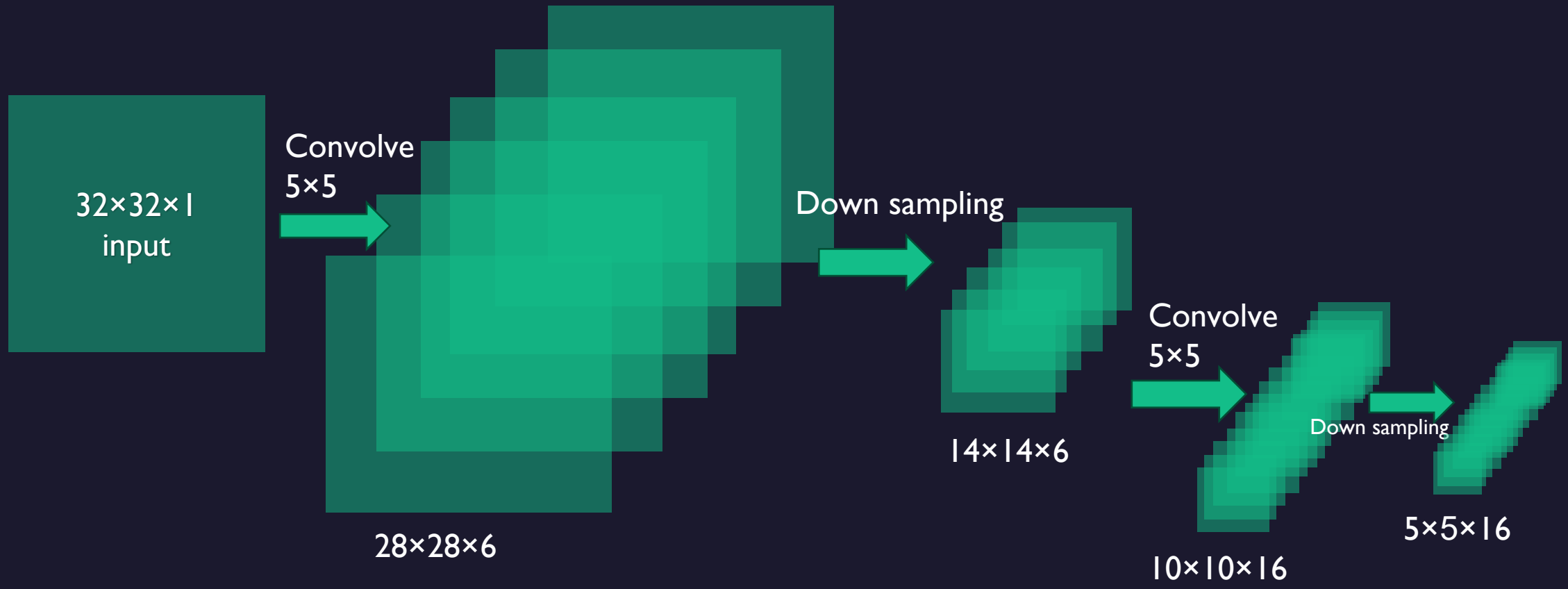
$$\frac{14 - 5 + 2 * 0}{1} + 1 = 10$$



LeNET

INPUT ➡️ [CONV(6, 5x5, tanh) ➡️ AVG_POOL(2x2)]
➡️ [CONV(16, 5x5, tanh) ➡️ AVG_POOL(2x2)]

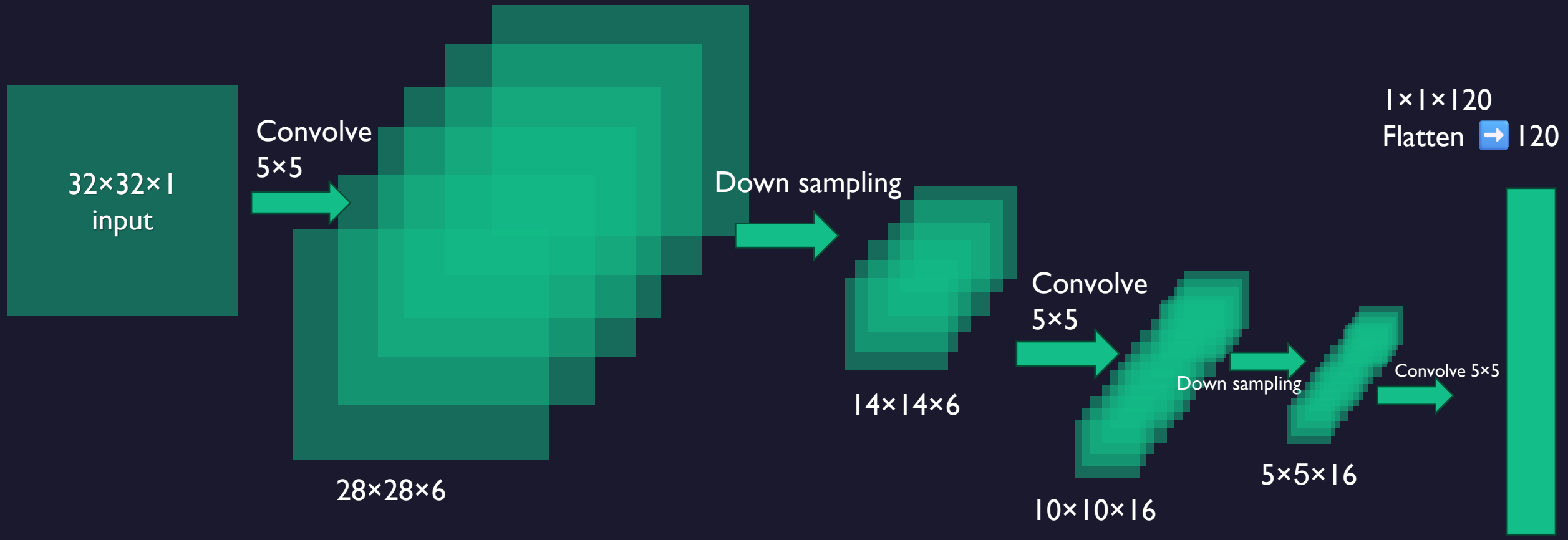
$$\frac{w - f + 2P}{S} + 1 \qquad \frac{10 - 2 + 2 * 0}{2} + 1 = 5$$



LeNET

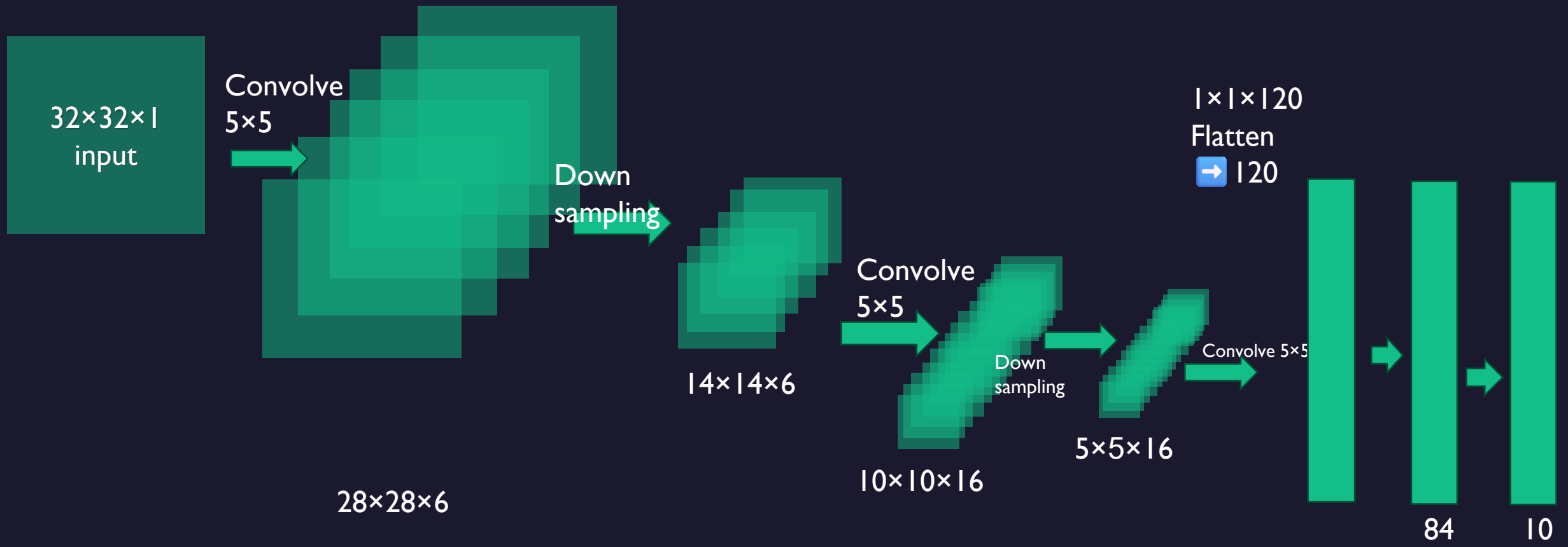
INPUT ➡ [CONV(6, 5x5, tanh) ➡ AVG_POOL(2x2)] ➡ [CONV(16, 5x5, tanh) ➡ AVG_POOL(2x2)] ➡ [CONV(120, 5x5, tanh)] ➡ FC(84, tanh) ➡ FC(10, softmax)

$$\frac{w - f + 2P}{S} + 1 \qquad \frac{5 - 5 + 2 * 0}{1} + 1 = 1$$



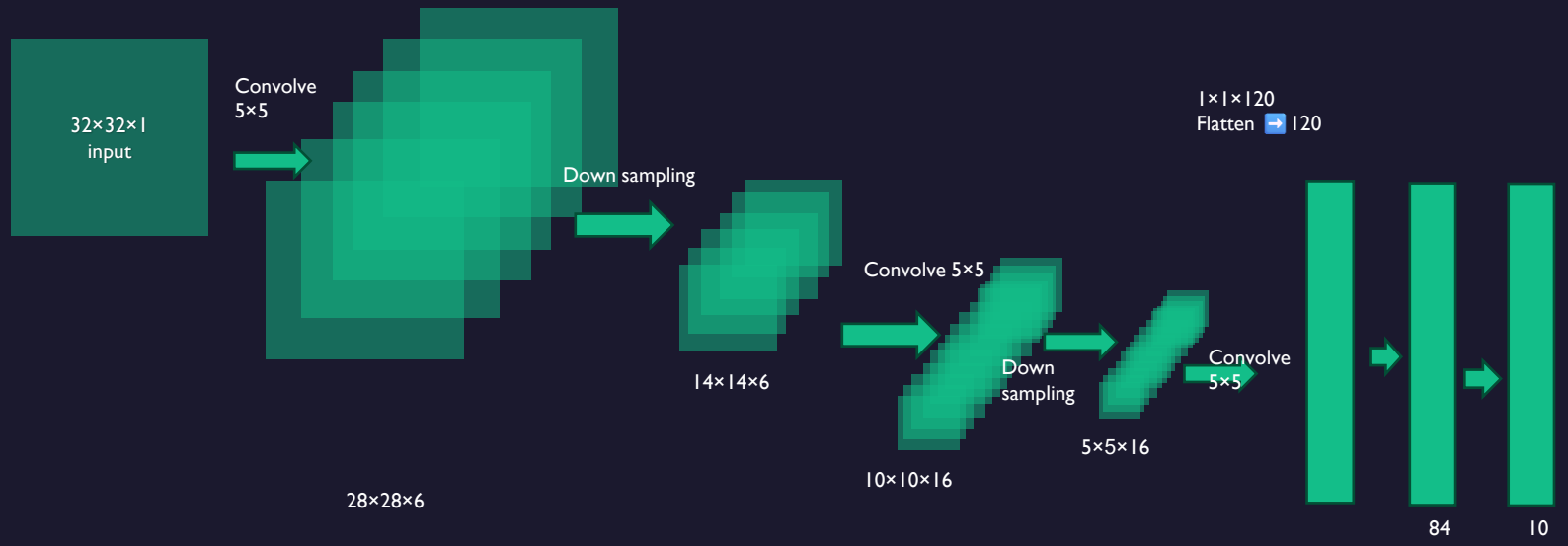
LeNET

INPUT ➡ [CONV(6, 5x5, tanh) ➡ AVG_POOL(2x2)]
➡ [CONV(16, 5x5, tanh) ➡ AVG_POOL(2x2)]
➡ [CONV(120, 5x5, tanh)] ➡ FC(84, tanh) ➡ FC(10, softmax)



LeNET What is number of parameters ?

- With Parameter Sharing it introduce $F.F.D_1$ weights per filter for a total of K ×($F.F.D_1$)



INPUT ➡ [CONV(6, 5x5, tanh) ➡ AVG_POOL(2x2)]
➡ [CONV(16, 5x5, tanh) ➡ AVG_POOL(2x2)]
➡ [CONV(120, 5x5, tanh)] ➡ FC(84, tanh) ➡ FC(10, softmax)

LeNET What is number of parameters ?

- **With Parameter Sharing it introduce $F.F.D_1$ weights per filter for a total of K ×($F.F.D_1$)**

INPUT ➡️ [CONV(6, 5x5, tanh) ➡️ AVG_POOL(2x2)]
➡️ [CONV(16, 5x5, tanh) ➡️ AVG_POOL(2x2)]
➡️ [CONV(120, 5x5, tanh)] ➡️ FC(84, tanh) ➡️ FC(10, softmax)

1.CONV(6, 5x5, tanh):

1. Parameters per filter: 5×5×1 (weights) + 1 (bias) = 26
2. Total parameters for the layer: 26×6 = 156

2.CONV(16, 5x5, tanh):

1. Parameters per filter: 5×5×6 (weights) + 1 (bias) = 151
2. Total parameters for the layer: 151×16 = 2416

3.CONV(120, 5x5, tanh):

1. Parameters per filter: 5×5×16 (weights) + 1 (bias) = 401
2. Total parameters for the layer: 401×120 = 48120

LeNET What is number of parameters ?

- **With Parameter Sharing it introduce $F.F.D_1$ weights per filter for a total of K ×($F.F.D_1$)**

INPUT ➡️ [CONV(6, 5x5, tanh) ➡️ AVG_POOL(2x2)]
➡️ [CONV(16, 5x5, tanh) ➡️ AVG_POOL(2x2)]
➡️ [CONV(120, 5x5, tanh)] ➡️ FC(84, tanh) ➡️ FC(10, softmax)

4. FC(84, tanh):

1. Parameters per neuron: 1×1×120 (assuming the previous layer has size 1×1×120) + 1 (bias) = 121
2. Total parameters for the layer: 121×84 = **10164 [previous #neurons * current # neurons in FC]**

5. FC(10, softmax):

1. Parameters per neuron: 84×1 + 1 (bias) = 85
2. Total parameters for the layer: 85×10 = 850

Now, summing up the parameters for all layers: 156+2416+48120+10164+850=**61706** parameters in total for the LeNet-5 architecture

LeNET implementation

```
import tensorflow as tf
from tensorflow.keras import layers, models

model = models.Sequential()
# Layer 1: Convolutional Layer
model.add(layers.Conv2D(6, (5, 5), activation='tanh', input_shape=(32, 32, 1)))
# downsampling
model.add(layers.AveragePooling2D((2, 2)))
# Layer 2: Convolutional Layer
model.add(layers.Conv2D(16, (5, 5), activation='tanh'))
# downsampling
model.add(layers.AveragePooling2D((2, 2)))
# Layer 3: Convolutional Layer
model.add(layers.Conv2D(120, (5, 5), activation='tanh'))
```

LeNET implementation

```
# Flatten the output for fully connected layers
model.add(layers.Flatten())
# Layer 4: Fully Connected Layer
model.add(layers.Dense(84, activation='tanh'))
# Output Layer
model.add(layers.Dense(10, activation='softmax'))

# Display the model summary
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 28, 28, 6)         156

 average_pooling2d_2 (Avera  (None, 14, 14, 6)         0
 gePooling2D)

 conv2d_4 (Conv2D)           (None, 10, 10, 16)        2416

 average_pooling2d_3 (Avera  (None, 5, 5, 16)          0
 gePooling2D)

 conv2d_5 (Conv2D)           (None, 1, 1, 120)         48120

 flatten_1 (Flatten)         (None, 120)               0

 dense_2 (Dense)             (None, 84)                10164

 dense_3 (Dense)             (None, 10)                850

=================================================================
Total params: 61706 (241.04 KB)
Trainable params: 61706 (241.04 KB)
Non-trainable params: 0 (0.00 Byte)
```

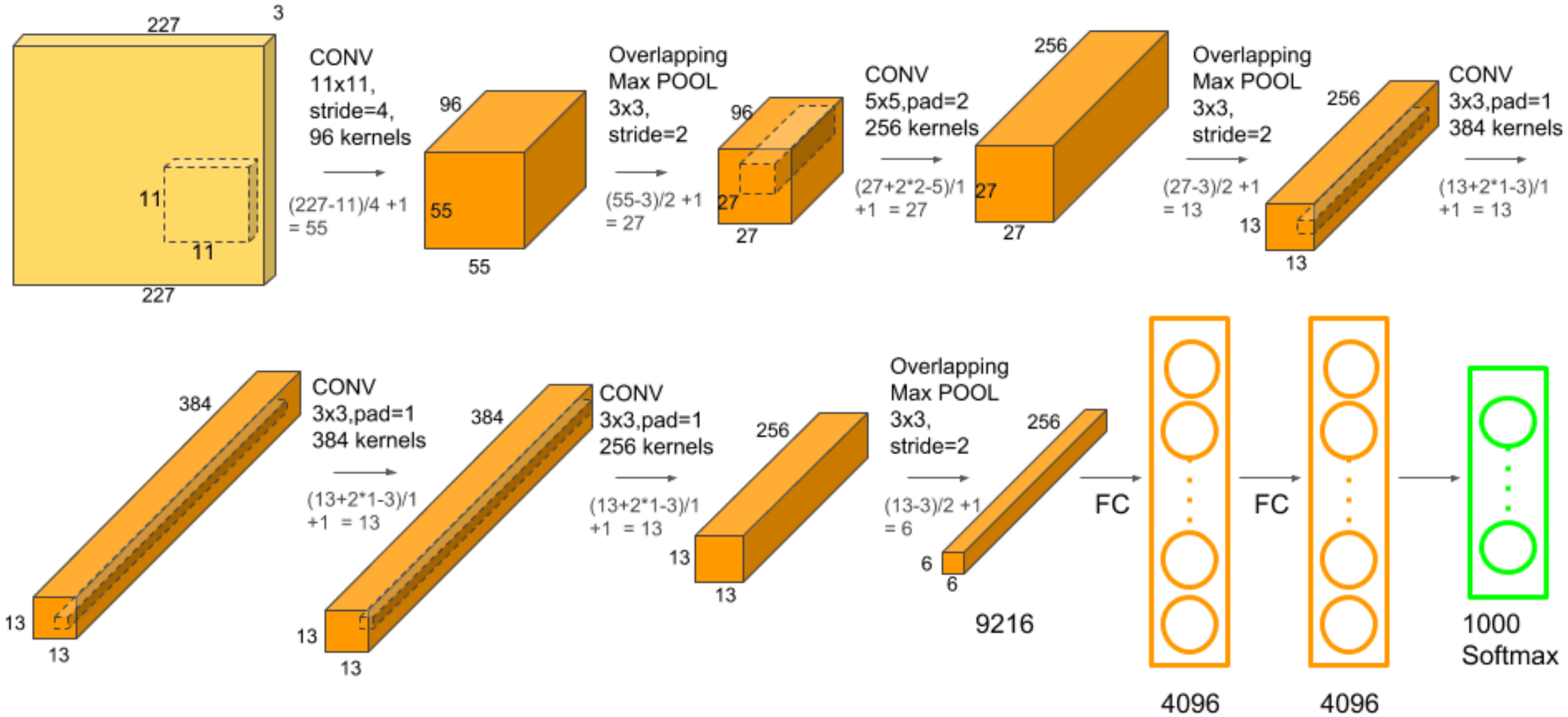
AlexNET

- The Alexnet has eight layers with learnable parameters. The model consists of five layers with a combination of max pooling followed by 3 fully connected layers and they use Relu activation in each of these layers except the output layer.

- They found out that using the relu as an activation function accelerated the speed of the training process by almost six times.

- They also used the dropout layers, that prevented their model from overfitting. Further, the model is trained on the Imagenet dataset. The Imagenet dataset has almost 14 million images across a thousand classes.

- Alexnet won the Imagenet large-scale visual recognition challenge in 2012.


227
3
227
11
11
CONV
11x11,
stride=4,
96 kernels
(227-11)/4 +1
= 55
96
55
55
Overlapping
Max POOL
3x3,
stride=2
(55-3)/2 +1
= 27
96
27
27
CONV
5x5,pad=2
256 kernels
(27+2*2-5)/1
+1 = 27
256
27
27
Overlapping
Max POOL
3x3,
stride=2
(27-3)/2 +1
= 13
256
13
13
CONV
3x3,pad=1
384 kernels
(13+2*1-3)/1
+1 = 13
384
13
13
CONV
3x3,pad=1
384 kernels
(13+2*1-3)/1
+1 = 13
384
13
13
CONV
3x3,pad=1
256 kernels
(13+2*1-3)/1
+1 = 13
256
13
13
Overlapping
Max POOL
3x3,
stride=2
(13-3)/2 +1
= 6
256
6
6
9216
FC
4096
FC
4096
1000
Softmax

AlexNET

[CONV ➡ RN ➡ POOL] 🔁 2
➡ [CONV 🔁 3 ➡ POOL]
➡ [FC ➡ DO] 🔁 2
➡ FC(SOFTMAX)

INPUT ➡ [CONV(96, 11x11, RELU) ➡ POOL(3x3)]
➡ [CONV(256, 5x5, RELU) ➡ POOL(3x3)]
➡ [CONV(384, 3x3, RELU)]
➡ [CONV(384, 3x3, RELU)]
➡ [CONV(256, 3x3, RELU) ➡ POOL(3x3)]
➡ FC(4096, RELU) ➡ FC(4096, RELU) ➡ FC(1000, softmax)

AlexNET

[CONV ➡️ RN ➡️ POOL] 🔁 2
➡️ [CONV 🔁 3 ➡️ POOL]
➡️ [FC ➡️ DO] 🔁 2
➡️ FC(SOFTMAX)

RN = local response normalization

DO = dropout

- It has 8 layers with learnable parameters.
- The input to the Model is RGB images.
- It has 5 convolution layers with a combination of max-pooling layers.
- Then it has 3 fully connected layers.
- The activation function used in all layers is Relu.
- It used two Dropout layers.
- The activation function used in the output layer is Softmax.
- The total number of parameters in this architecture is 62.3 million
- The first to use GPUs

AlexNET

- Alexnet is a deep architecture, the authors introduced padding to prevent the size of the feature maps from reducing drastically.
- In the original paper the size written as 224 and adding the padding would be 227 don't get confused
- Dropout involves randomly "dropping out" (i.e., setting to zero) a certain percentage of neurons during each training iteration.
- In AlexNet, dropout is employed in the fully connected layers. For example, in the last three fully connected layers with 4096 neurons each, dropout is used with a dropout rate of 0.5. This means that during each training iteration, approximately half of the neurons in these layers are randomly dropped out, preventing the model from relying too heavily on specific neurons and features.

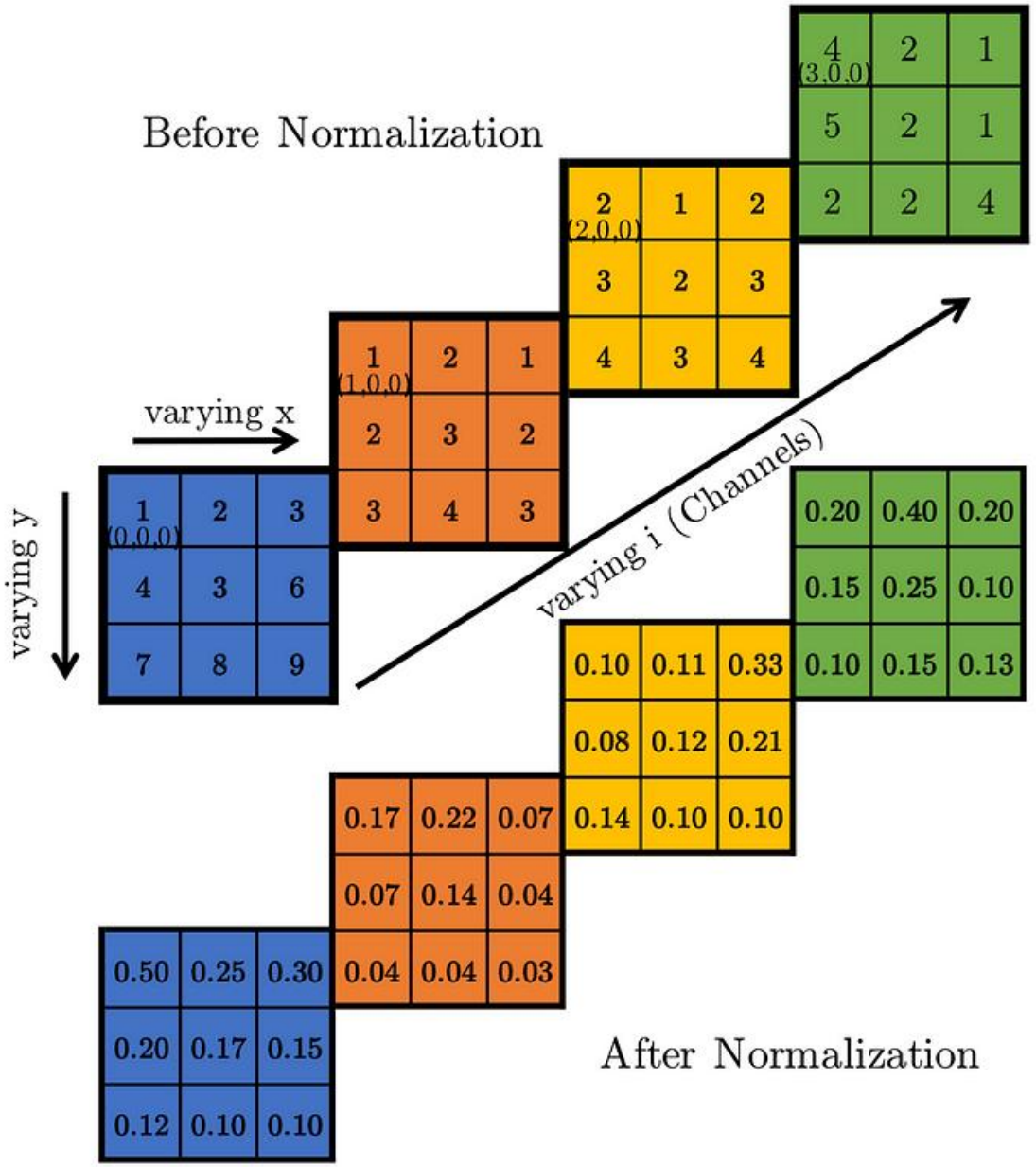
AlexNET

- **Local Response Normalization (LRN):** *a technique used to normalize the activity of neurons within a local neighborhood. The idea is to amplify the activations of neurons that are relatively more active compared to their neighbors. LRN is applied to the output of some convolutional layers.*

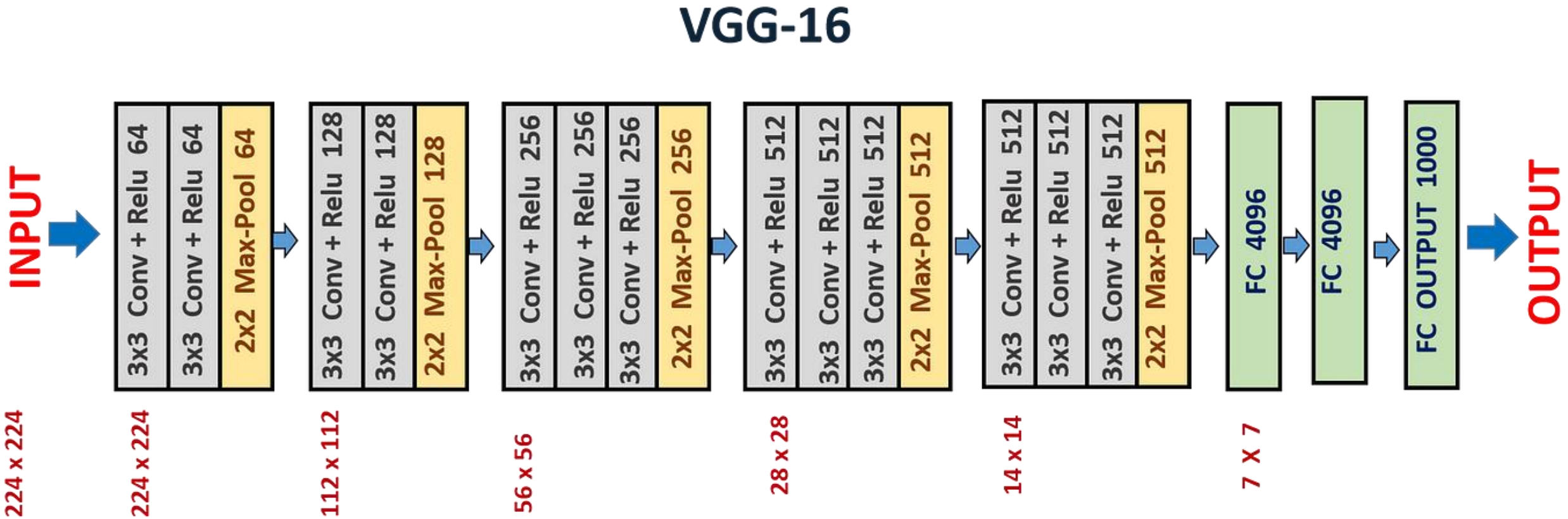
- *In AlexNet, LRN is applied after the first and second convolutional layers. The specific form of LRN used in AlexNet is a type of divisive normalization, where each activation is divided by the sum of the squares of the activations of nearby neurons (within a certain window). This normalization helps enhance the contrast between activated neurons, promoting competition and diversity among feature detectors.*

- *Brightness normalization*

Normalization



VGG-16 *16 weighted layers*

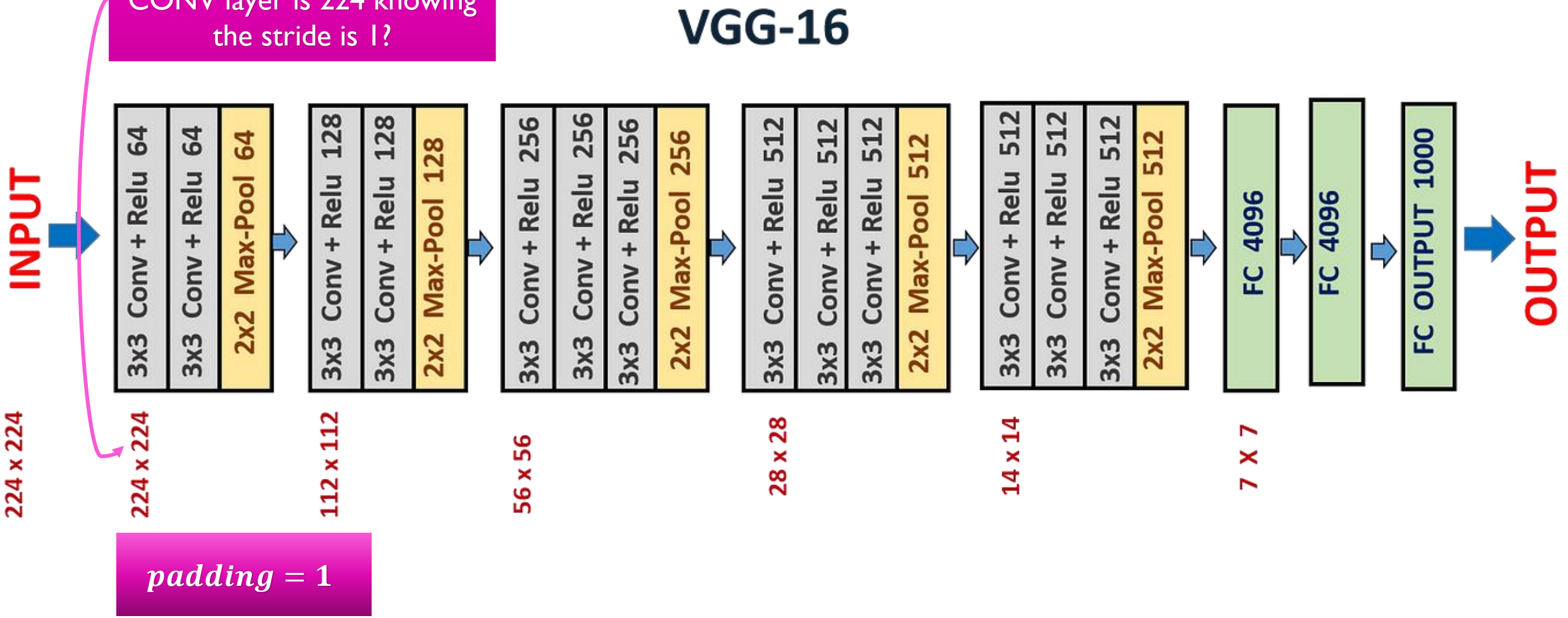


VGG-16 *16 weighted layers*

$$\frac{w - f + 2P}{S} + 1$$

$$\frac{224 - 3 + 2P}{1} + 1 = 224$$

How the output of the first CONV layer is 224 knowing the stride is 1?

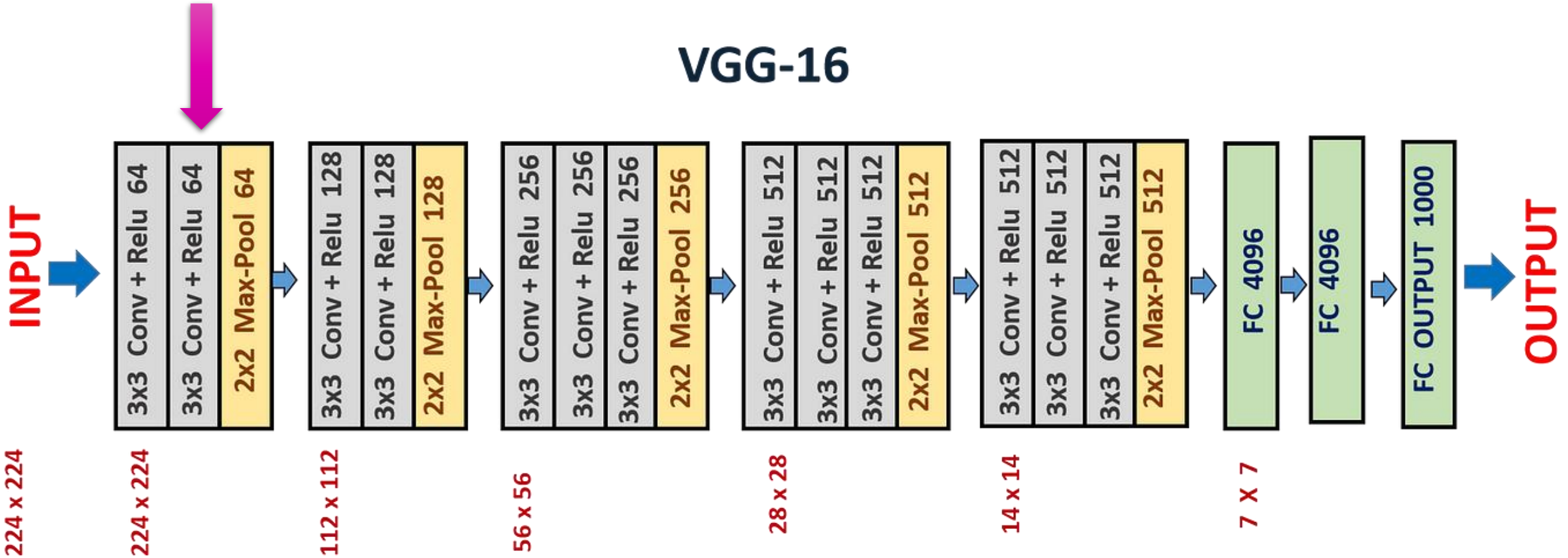


$$padding = 1$$

VGG-16 *16 weighted layers*

$$\frac{w - f + 2P}{S} + 1$$

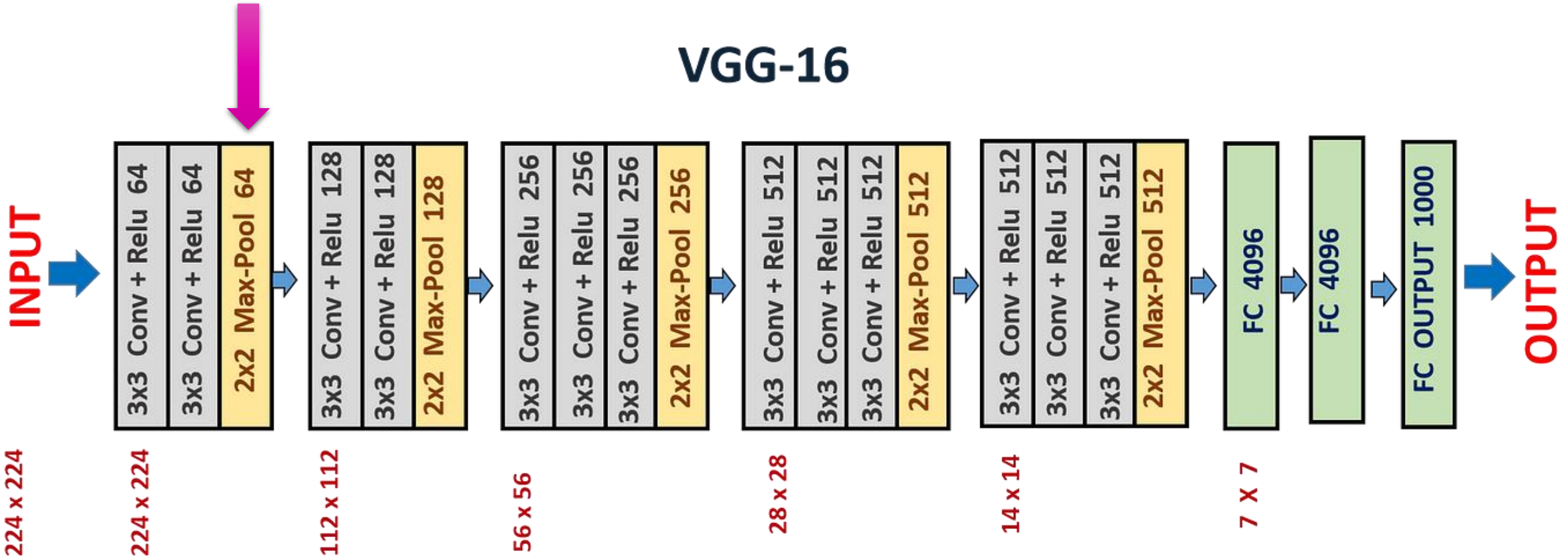
$$\frac{224 - 3 + 2 * 1}{1} + 1 = 224$$



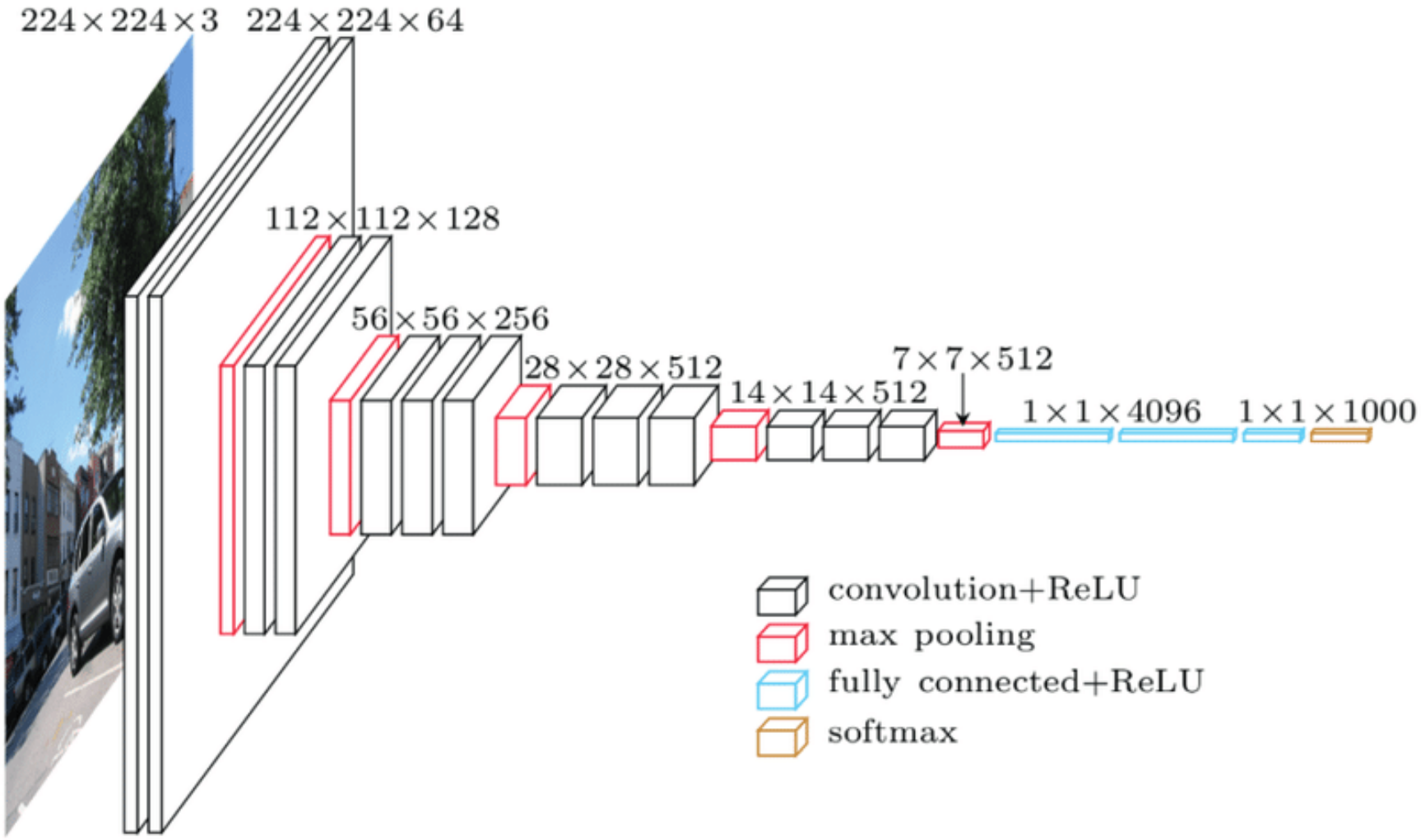
VGG-16 *16 weighted layers*

$$\frac{w - f + 2P}{S} + 1$$

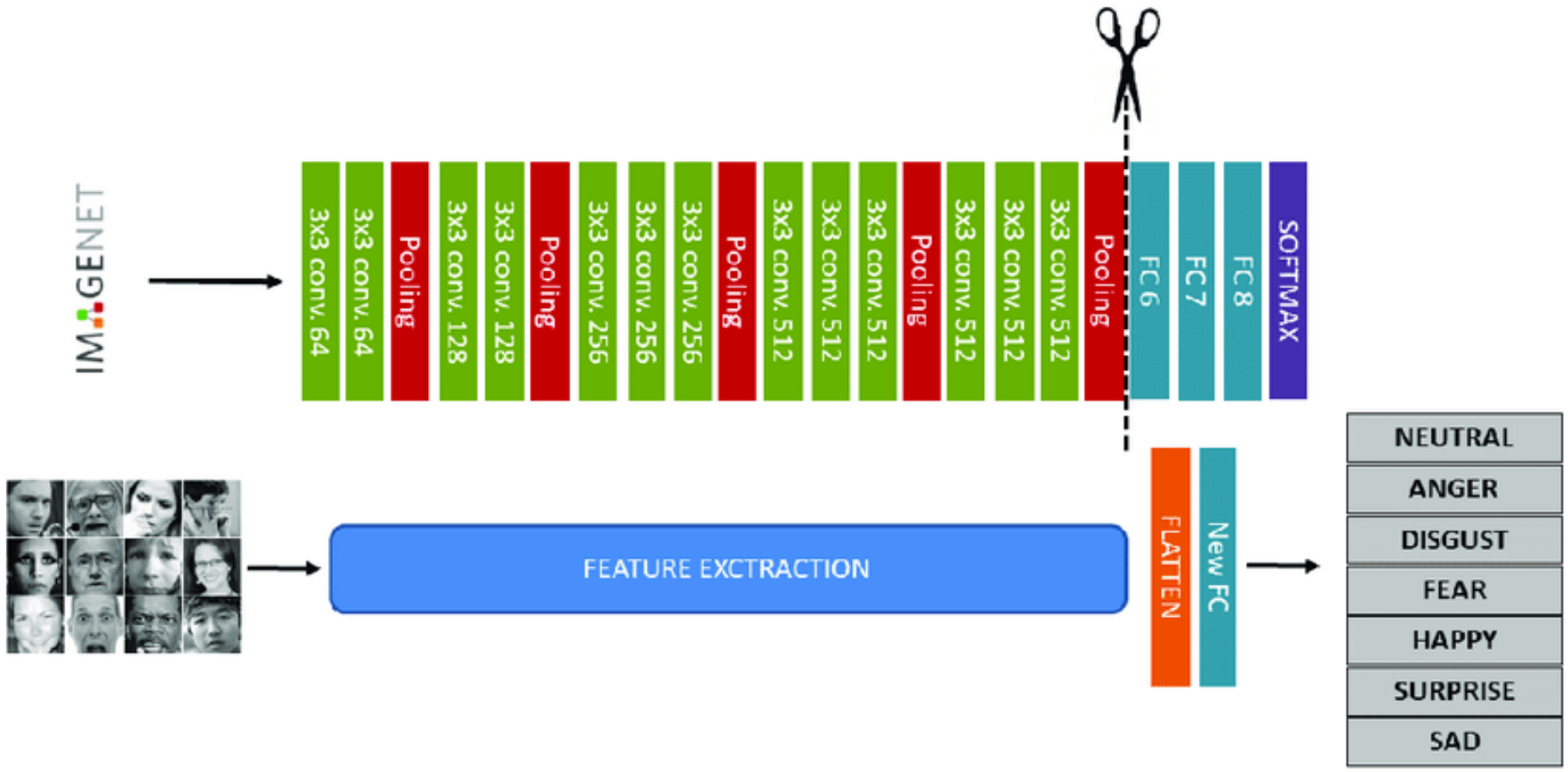
$$\frac{224 - 2 + 2 * 0}{2} + 1 = 112$$



VGG-16


224 × 224 × 3
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution+ReLU
max pooling
fully connected+ReLU
softmax

VGG-16 Transfer learning



VGG-16 as a pretrained model


```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
# can be used to apply data augmentation (transformations to data)
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load pre-trained VGG16 model with weights trained on ImageNet
# include_top set to False drop the last softmax layer
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

VGG-16 as a pretrained model


```
# Freeze the convolutional layers
for layer in base_model.layers:
    layer.trainable = False

# Create a new model by adding custom layers on top of the pre-trained base
model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dense(4096, activation='relu'))
model.add(Dense(1, activation='sigmoid'))  # Binary classification, adjust for your task

# Compile the model
model.compile(optimizer=Adam(lr=0.001), loss='binary_crossentropy', metrics=['accuracy'])

# Display the model summary
model.summary()
```

VGG-16 using

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14714688
flatten_2 (Flatten)	(None, 25088)	0
dense_4 (Dense)	(None, 4096)	102764544
dense_5 (Dense)	(None, 4096)	16781312
dense_6 (Dense)	(None, 1)	4097

Total params: 134264641 (512.18 MB)
Trainable params: 119549953 (456.05 MB)
Non-trainable params: 14714688 (56.13 MB)

VGG-16 and VGG-19 Common things

- VGG in general use CONV layers with 3×3 Kernels
- In some configurations it use 1×1 CONV which keep the spatial shape the same but can add more depth or reduce the depth of the channels
- 1×1 CONV to reduce the dimensions (depth) and max pooling layers with 2×2 and stride of 2 to reduce the spatial (width + height)
- Use padding of 1
- K the number of filters following this pattern {64, 128, 256, 512, 512} in the two variations with the main difference is the Loop in vgg-19 it use 4 sets of 256, 512, 512 while in vgg-16 only 3 sets of the same k numbers
- Uses ReLU as activation function

VGG-16 vs VGG-19

CONV(64) 🔁 2 ➡ POOL ➡
CONV(128) 🔁 2 ➡ POOL ➡
CONV(256) 🔁 3 ➡ POOL ➡
CONV(512) 🔁 3 ➡ POOL ➡
CONV(512) 🔁 3 ➡ POOL ➡
FC ➡ FC ➡ FC

VGG-16

2+2+3*3+3 = 16

CONV(64) 🔁 2 ➡ POOL ➡
CONV(128) 🔁 2 ➡ POOL ➡
CONV(256) 🔁 4 ➡ POOL ➡
CONV(512) 🔁 4 ➡ POOL ➡
CONV(512) 🔁 4 ➡ POOL ➡
FC ➡ FC ➡ FC

VGG-19

2+2+4*3+3 = 19

See 👀

- https://poloclub.github.io/cnn-explainer/ [Visualization]
- https://www.youtube.com/watch?v=eMXuk97NeSI&list=PLZDCDMGmeIH-pHt-Ij0nImVrOmj8DYKbB&pp=iAQB [YouTube playlist with nice 3D visualizations]
- https://www.youtube.com/playlist?list=PLkDaE6sCZn6GI29AoE31iwdVwSG-KnDzF [You can't mess this one really would be helpful]
- https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks [Stanford]
- https://www.youtube.com/playlist?list=PLkDaE6sCZn6GI29AoE31iwdVwSG-KnDzF [You can't mess this one really would be helpful]

References

- https://cs231n.github.io/convolutional-networks/#conv
- https://poloclub.github.io/cnn-explainer/
- https://builtin.com/machine-learning/fully-connected-layer
- https://www.ibm.com/topics/convolutional-neural-networks
- https://en.wikipedia.org/wiki/Convolutional_neural_network
- https://datahacker.rs/028-visualization-and-understanding-of-convolutional-neural-networks-in-pytorch/
- https://towardsdatascience.com/difference-between-local-response-normalization-and-batch-normalization-272308c034ac
- https://en.wikipedia.org/wiki/AlexNet
- https://keras.io/api/applications/ [Keras pretrained models]
- https://keras.io/examples/ [Keras projects examples]
- https://keras.io/api/applications/vgg/#vgg16-function
- https://www.analyticsvidhya.com/blog/2021/03/introduction-to-the-architecture-of-alexnet/
- https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/
- https://arxiv.org/abs/1409.1556 [VGG paper 📄]


Thanks