Machine Learning

Louis Fippo Fitime

November 28, 2022

Convolutional Neural Networks

Introduction

- Convolutional neural networks (CNNs) emerged from the study of the brain's visual cortex, and they have been used in image recognition since the 1980s.
- A convolutional neural network (CNN) is specific kind of deep learning architecture that uses the convolution operation to extract relevant explanatory features for the input image.

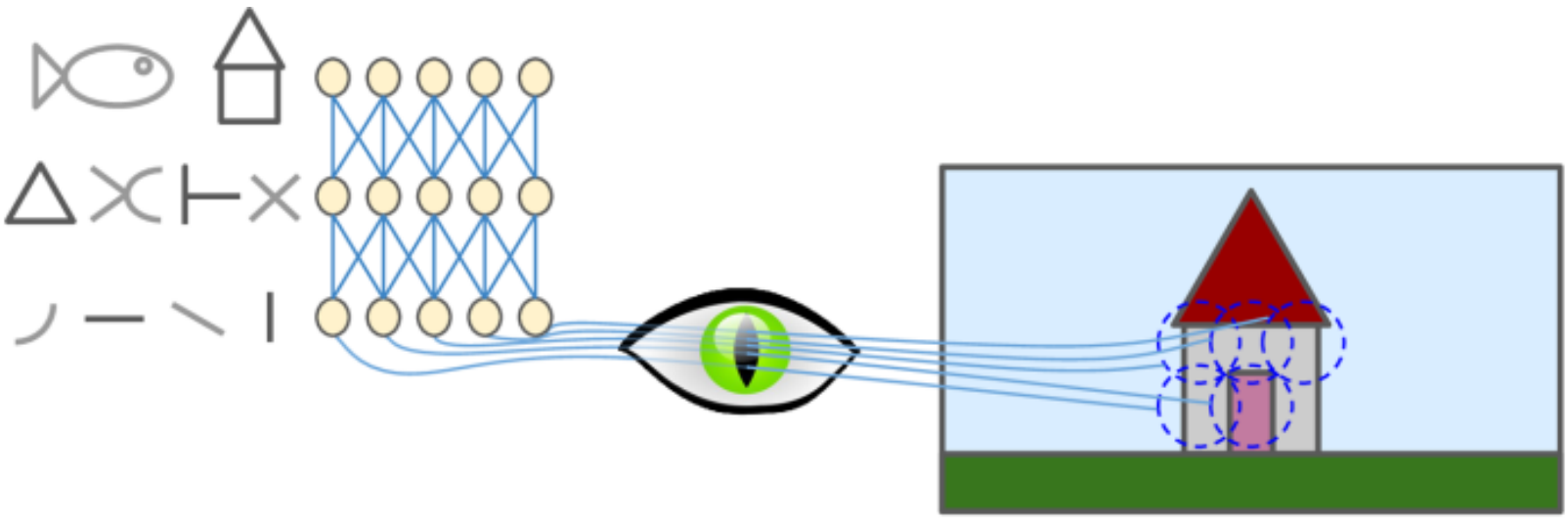


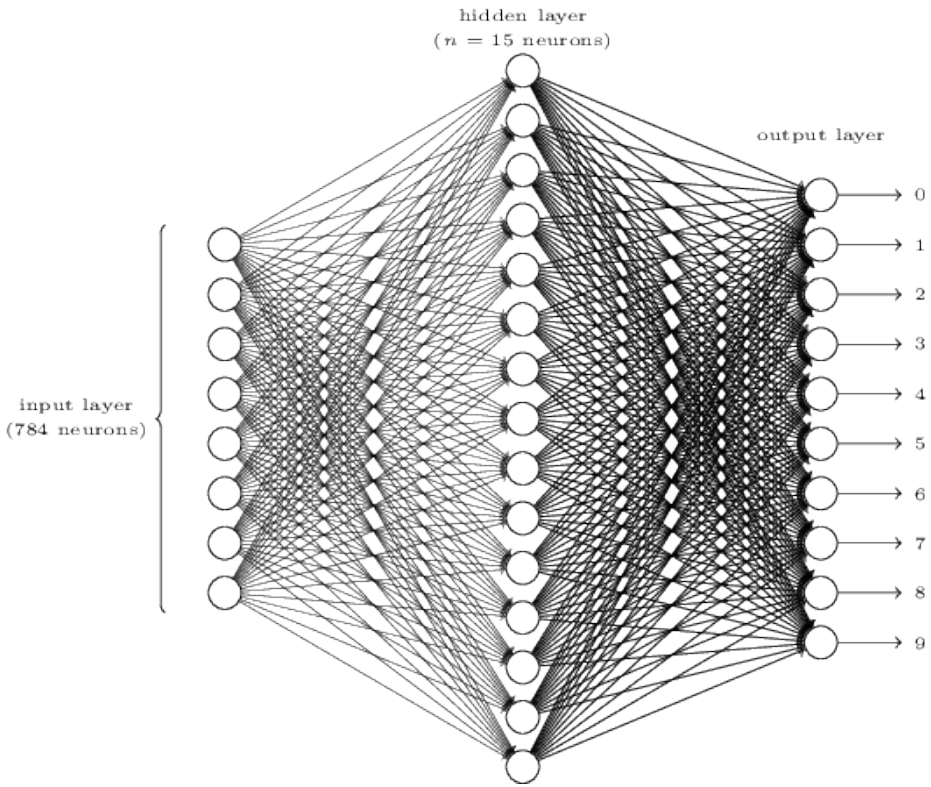
Figure 1: Biological neurons in the visual cortex respond

Convolutional Neural Networks

Intuition

A **convolutional neural network (CNN)** is essentially a localized and sparse feedfor-ward MLP that is designed to exploit spatial and/or temporal structure in the input data.

In a regular MLP all of the neurons in layer l are connected to all of the neurons in layer l + 1.



Convolutional Neural Networks

Intuition

A **convolutional neural network (CNN)** is essentially a localized and sparse feedfor-ward MLP that is designed to exploit spatial and/or temporal structure in the input data.

- CNN connects a contiguous or adjacent subset of neurons in layer l to a single neuron in the next layer $l + 1$.
- Different sliding windows comprising contiguous subsets of neurons in layer l connect to different neurons in layer $l + 1$.
- all of these sliding windows use parameter sharing, that is, the same set of weights, called a filter, is used for all sliding windows.
- Finally, different filters are used to automatically extract features from layer l for use by layer $l + 1$.

Convolutional Neural Networks

Illustration

A **convolutional neural network (CNN)** is essentially a localized and sparse feedfor-ward MLP that is designed to exploit spatial and/or temporal structure in the input data.

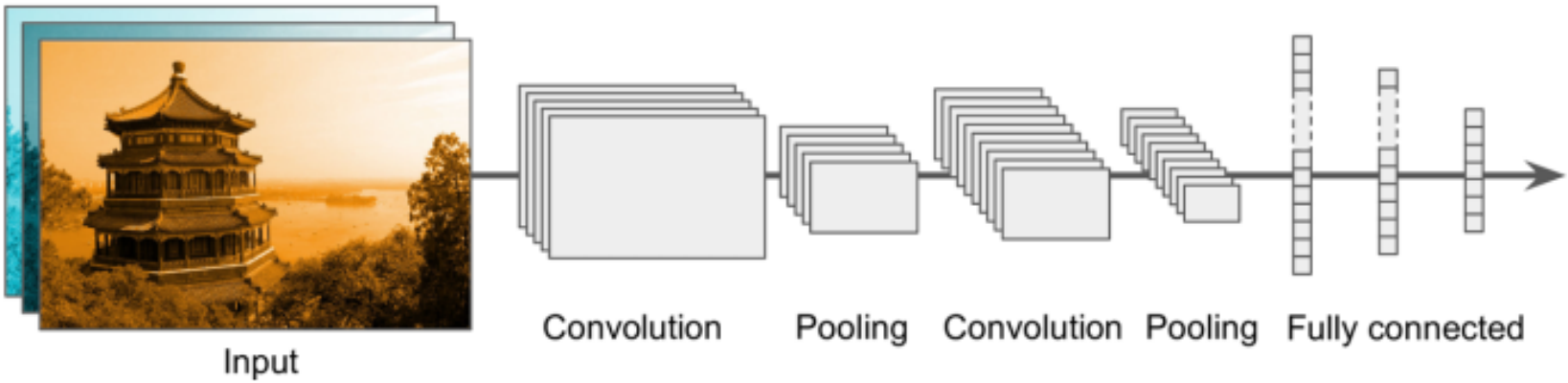


Figure 3: Illustration of CNN

Convolutional Neural Networks

Convolutions

- 1D Convolution
- 2D Convolution
- 3D Convolution

Convolutional Neural Networks

1D Convolution

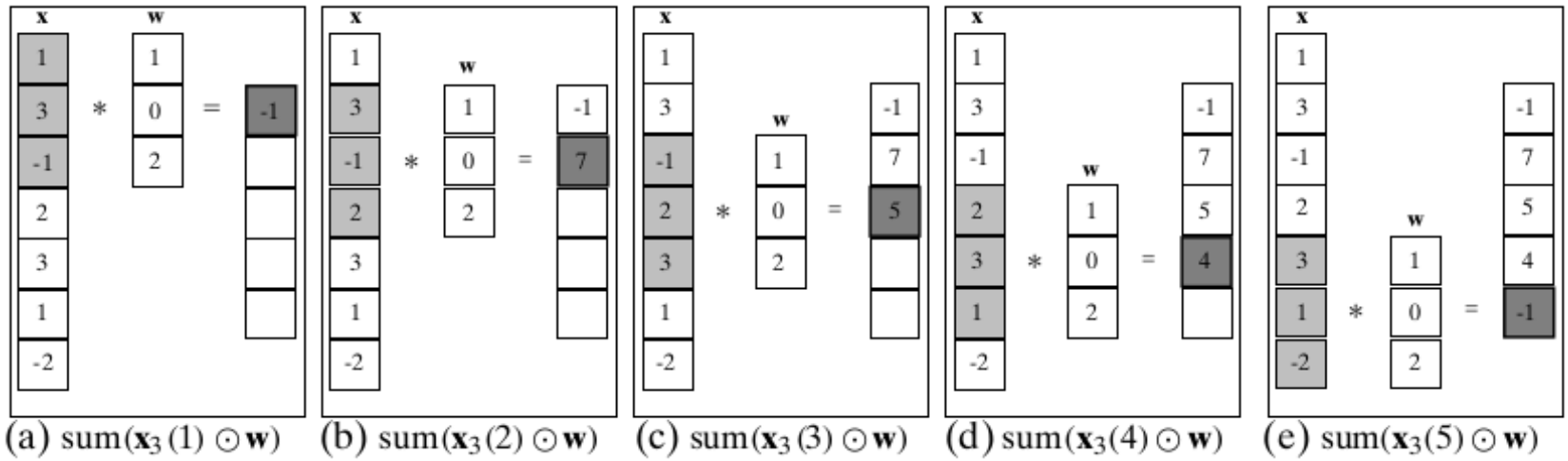


Figure 4: 1D Convolution

Convolutional Neural Networks

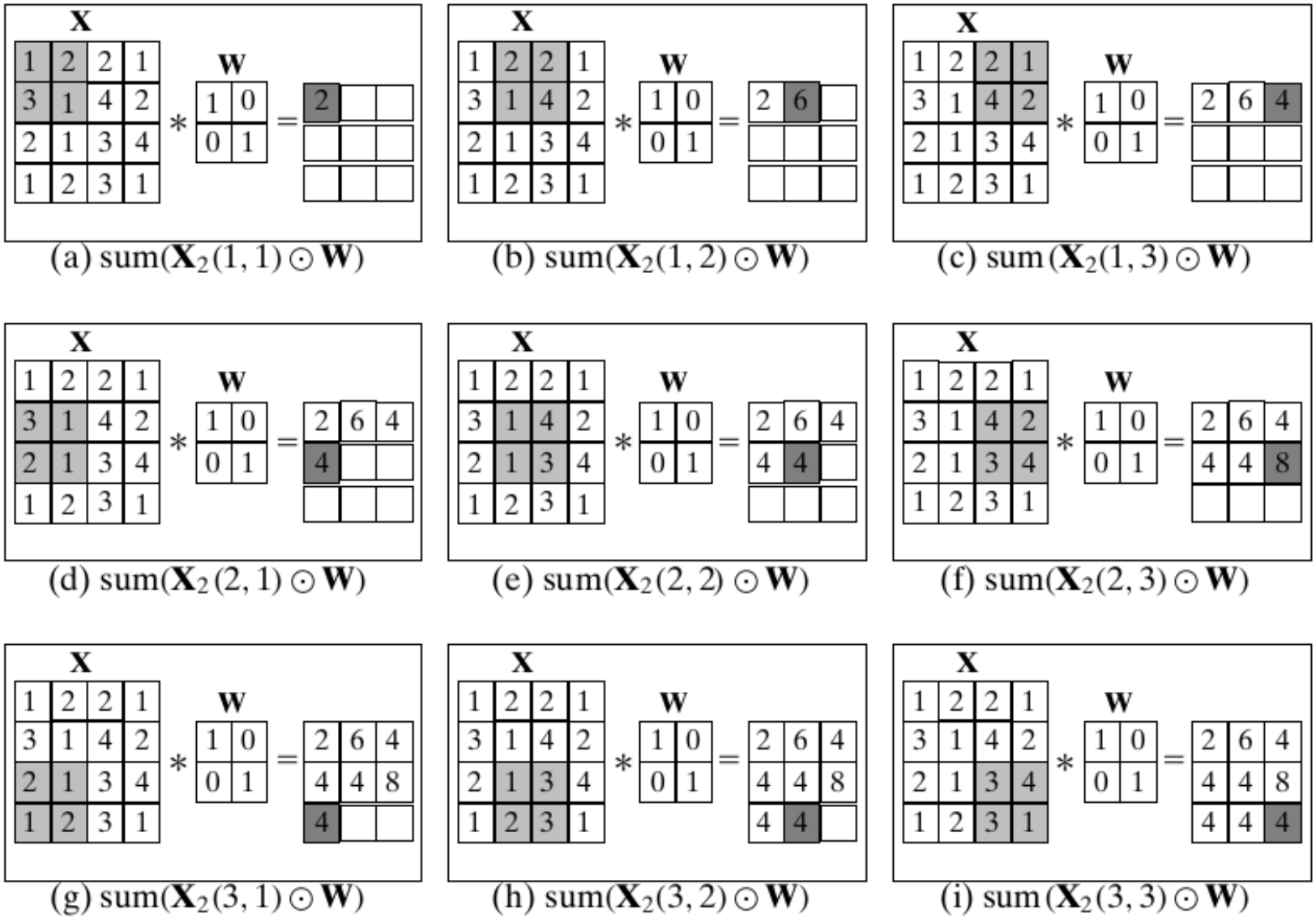
2D Convolution

$$\mathbf{X} * \mathbf{W} = \begin{pmatrix} \text{sum}\big(\mathbf{X}_k(1,1) \odot \mathbf{W}\big) & \cdots & \text{sum}\big(\mathbf{X}_k(1, n-k+1) \odot \mathbf{W}\big) \\ \text{sum}\big(\mathbf{X}_k(2,1) \odot \mathbf{W}\big) & \cdots & \text{sum}\big(\mathbf{X}_k(2, n-k+1) \odot \mathbf{W}\big) \\ \vdots & \cdots & \vdots \\ \text{sum}\big(\mathbf{X}_k(n-k+1,1) \odot \mathbf{W}\big) & \cdots & \text{sum}\big(\mathbf{X}_k(n-k+1, n-k+1) \odot \mathbf{W}\big) \end{pmatrix}$$

Figure 5: 2D Convolution

Convolutional Neural Networks

2D Convolution illustration



Convolutional Neural Networks

3D Convolution

$$\mathbf{X} * \mathbf{W} = \begin{pmatrix} \text{sum}\big(\mathbf{X}_k(1,1) \odot \mathbf{W}\big) & \cdots & \text{sum}\big(\mathbf{X}_k(1,n-k+1) \odot \mathbf{W}\big) \\ \text{sum}\big(\mathbf{X}_k(2,1) \odot \mathbf{W}\big) & \cdots & \text{sum}\big(\mathbf{X}_k(2,n-k+1) \odot \mathbf{W}\big) \\ \vdots & \cdots & \vdots \\ \text{sum}\big(\mathbf{X}_k(n-k+1,1) \odot \mathbf{W}\big) & \cdots & \text{sum}\big(\mathbf{X}_k(n-k+1,n-k+1) \odot \mathbf{W}\big) \end{pmatrix}$$

Figure 7: 3D Convolution

Convolutional Neural Networks

3D Convolution illustration

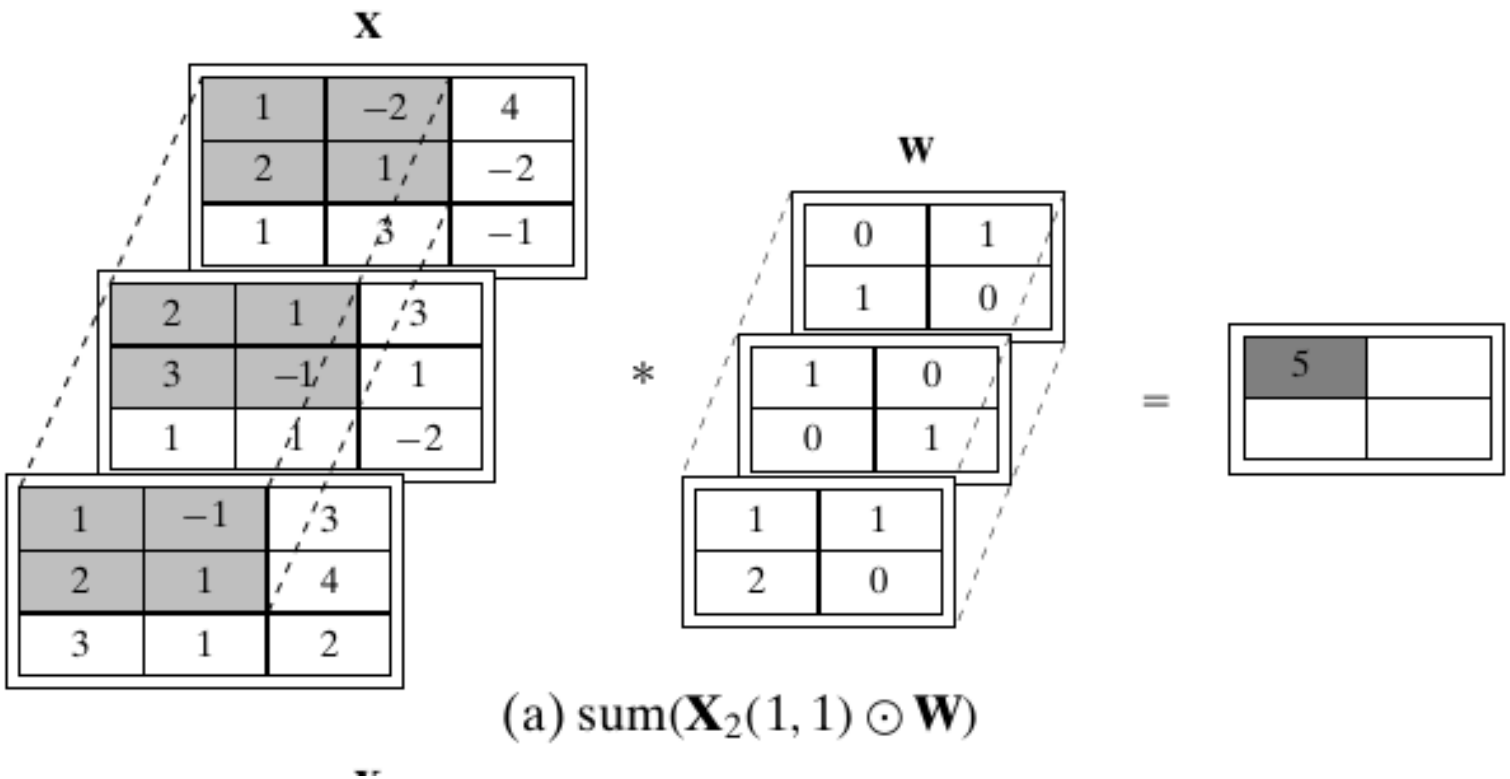


Figure 8: 3D Convolution illustration

Convolutional Neural Networks

3D Convolution illustration

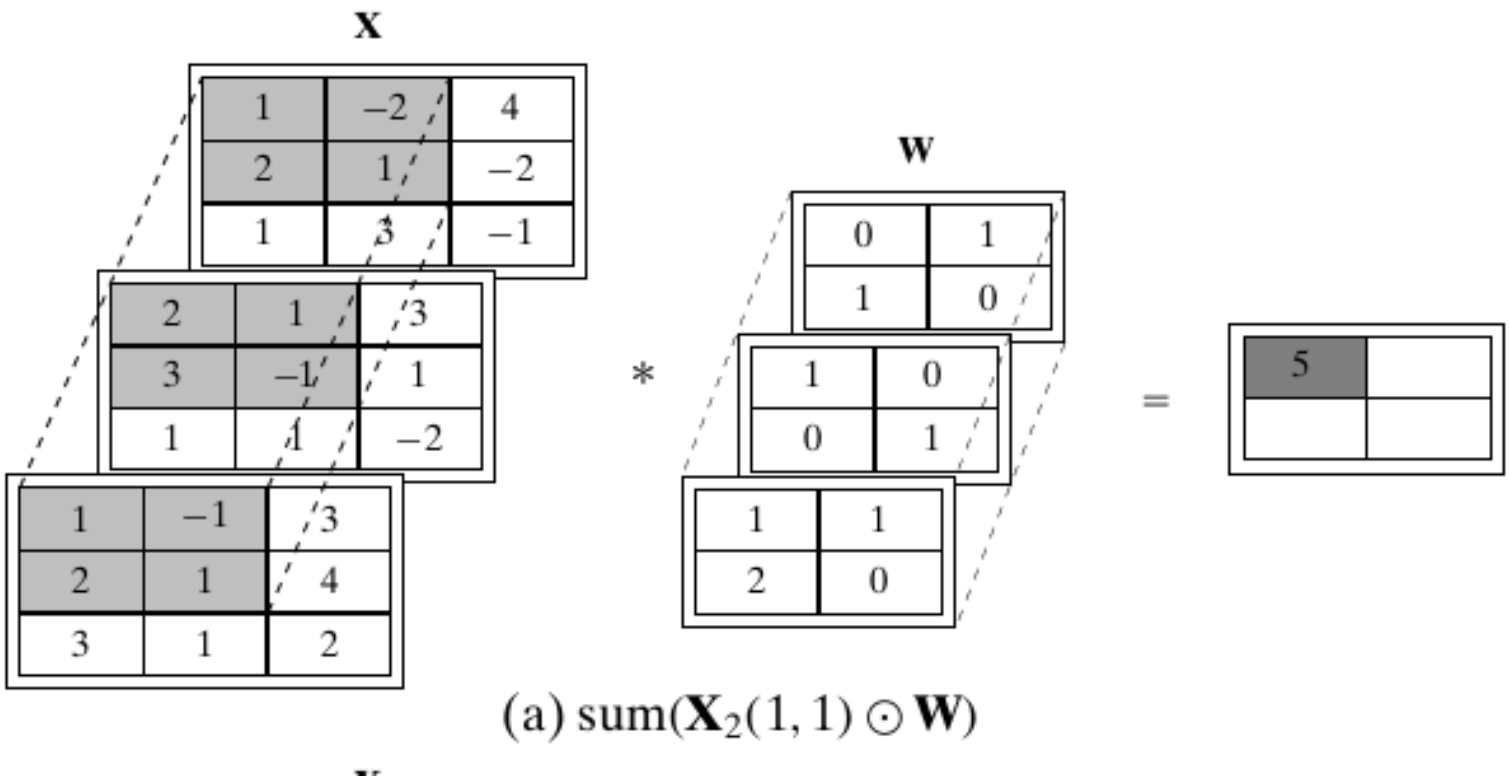
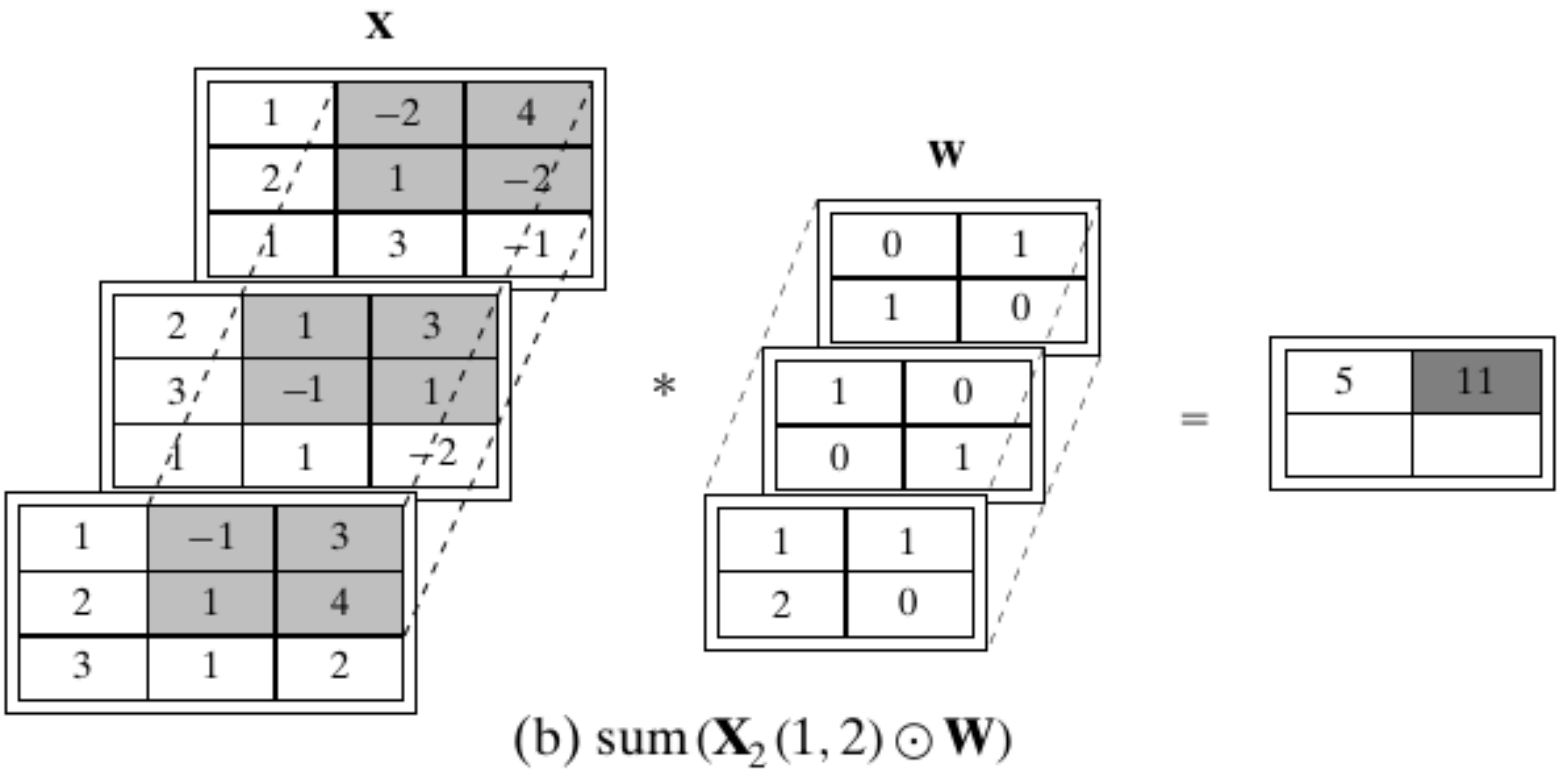


Figure 9: 3D Convolution illustration

Convolutional Neural Networks

3D Convolution illustration



(b) $\text{sum}(\mathbf{X}_2(1,2) \odot \mathbf{W})$

Figure 10: 3D Convolution illustration

Convolutional Neural Networks

3D Convolution illustration

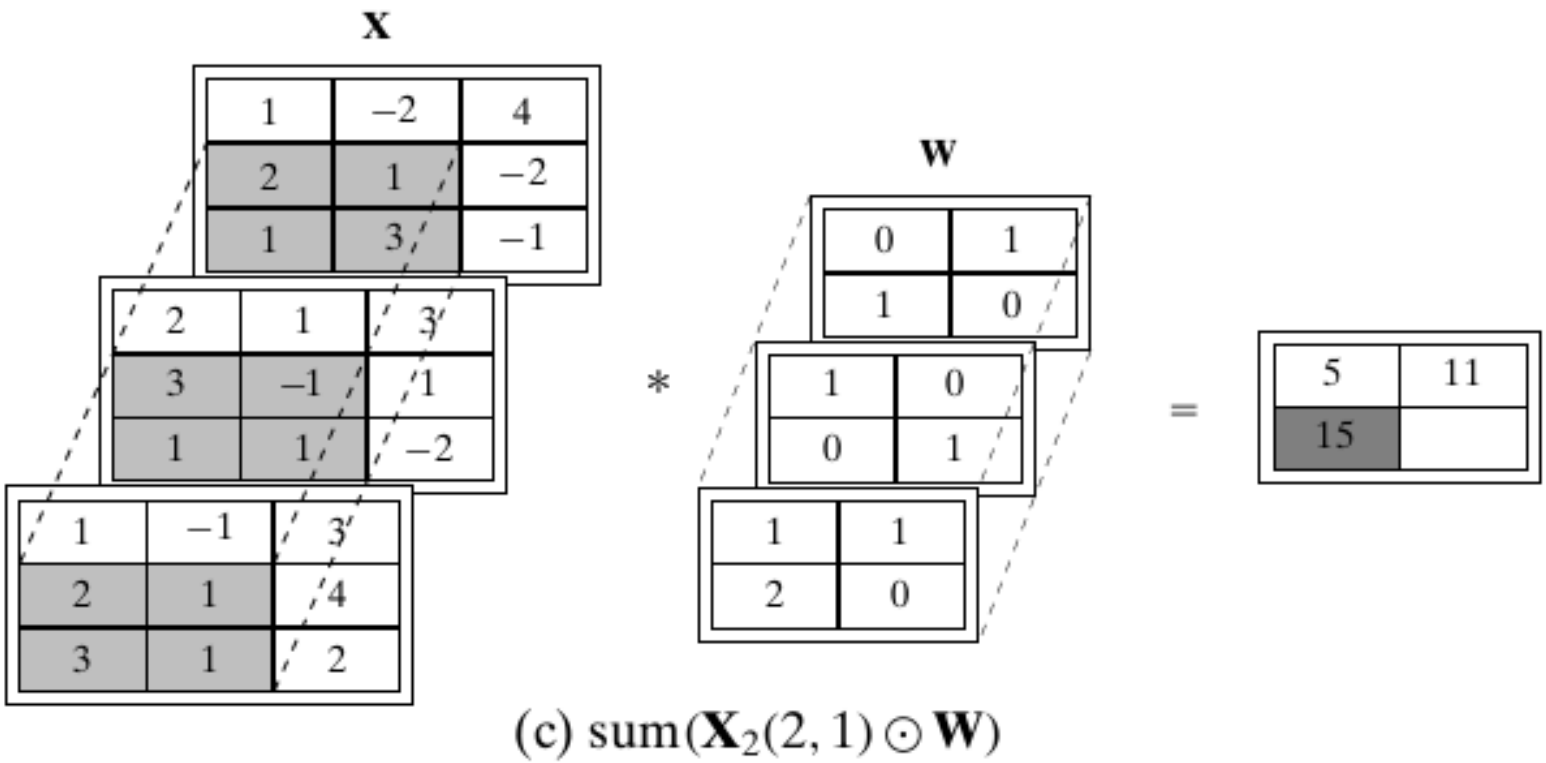


Figure 11: 3D Convolution illustration

Convolutional Neural Networks

3D Convolution illustration

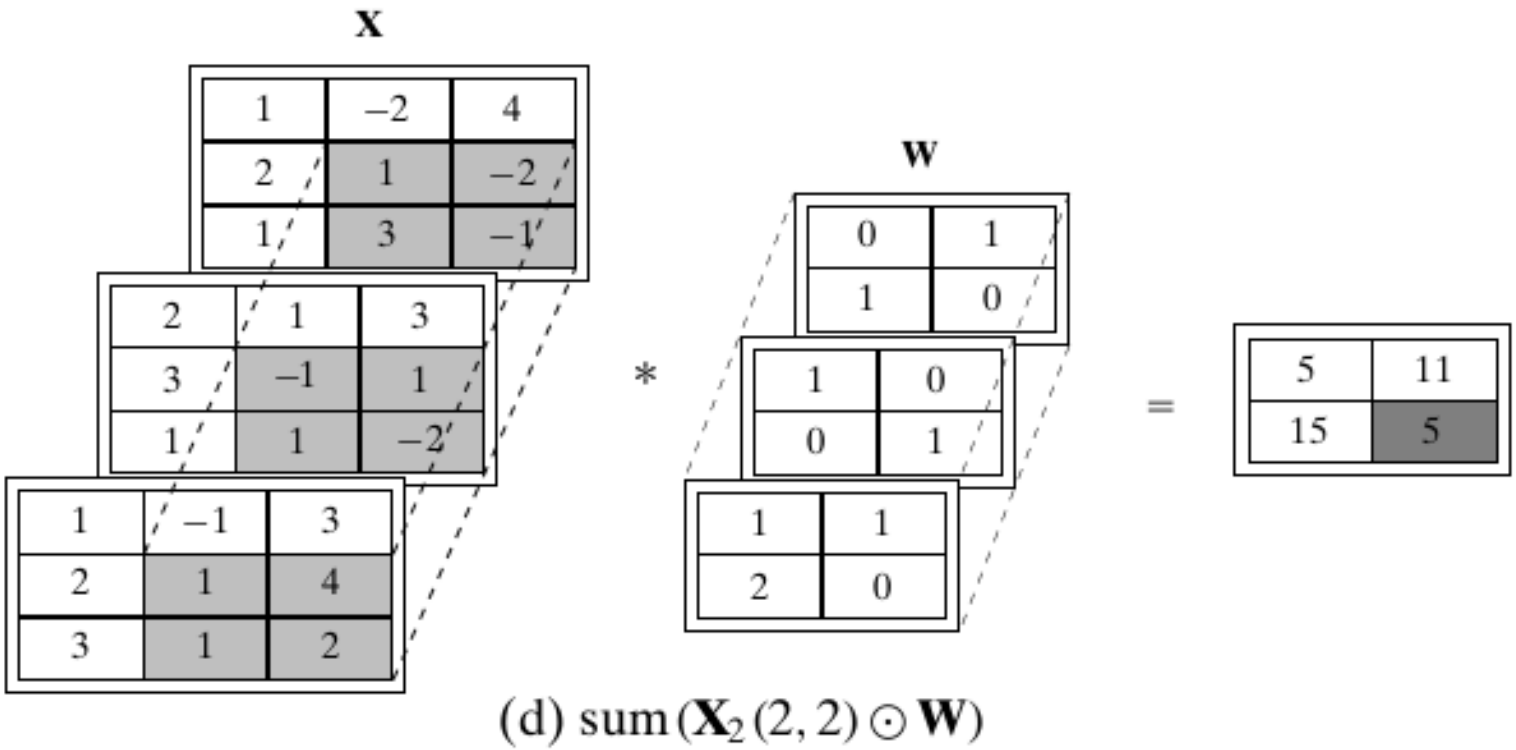


Figure 12: 3D Convolution illustration

Convolutional Neural Networks

Training a CNN

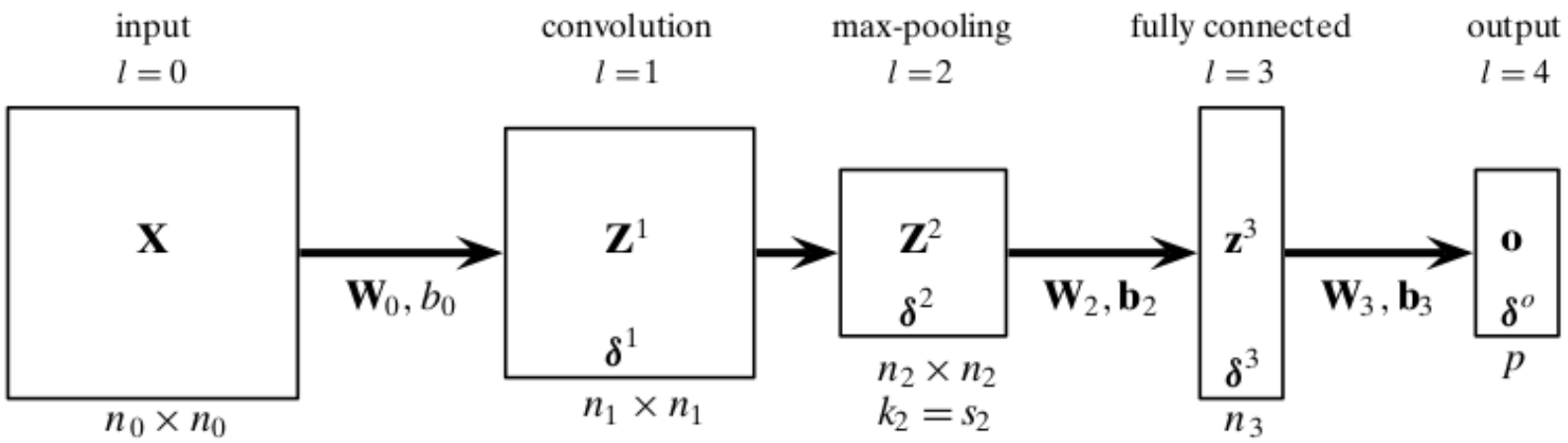


Figure 13: Illustration of CNN

Convolutional Neural Networks

CNN for MNIST

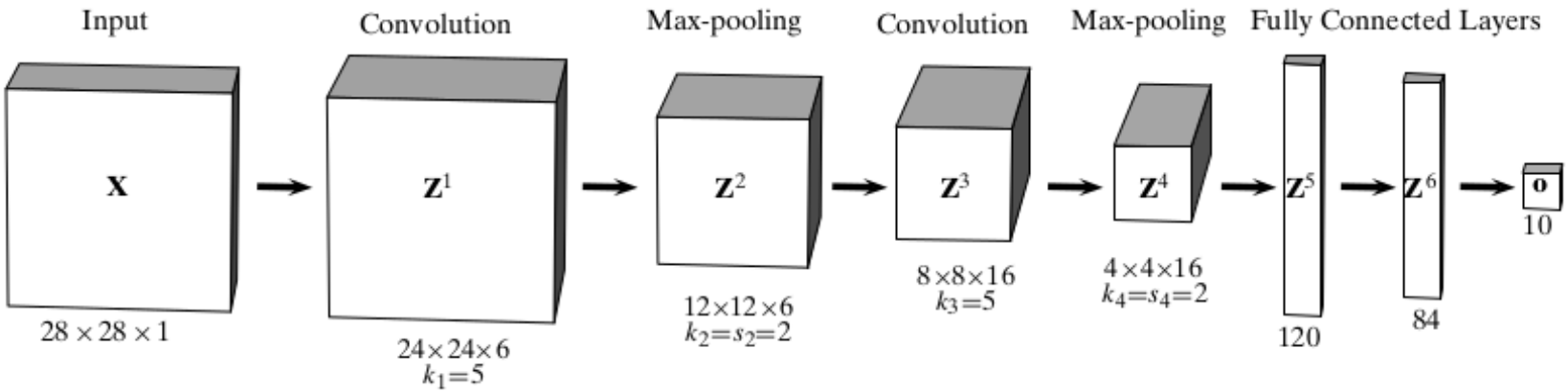


Figure 14: CNN for MNIST

Convolutional Neural Networks

CNN for MNIST with Keras

```
# This code implement a basic CNN architecture with Keras
    from functools import partial
DefaultConv2D = partial(keras.layers.Conv2D,
                        kernel_size=3, activation='relu', padding="SAME")
model = keras.models.Sequential([
    DefaultConv2D(filters=64, kernel_size=7, input_shape=[28, 28, 1]),
    keras.layers.MaxPooling2D(pool_size=2),
    DefaultConv2D(filters=128),
    DefaultConv2D(filters=128),
    keras.layers.MaxPooling2D(pool_size=2),
    DefaultConv2D(filters=256),
    DefaultConv2D(filters=256),
    keras.layers.MaxPooling2D(pool_size=2),
    keras.layers.Flatten(),
    keras.layers.Dense(units=128, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(units=64, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(units=10, activation='softmax'),
])
```

CNN Architectures

LeNet-5 form MNIST by Yann LeCun (1998)

Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully connected	–	10	–	–	RBF
F6	Fully connected	–	84	–	–	tanh
C5	Convolution	120	1×1	5×5	1	tanh
S4	Avg pooling	16	5×5	2×2	2	tanh
C3	Convolution	16	10×10	5×5	1	tanh
S2	Avg pooling	6	14×14	2×2	2	tanh
C1	Convolution	6	28×28	5×5	1	tanh
In	Input	1	32×32	–	–	–

Figure 15: LeNet-5

CNN Architectures

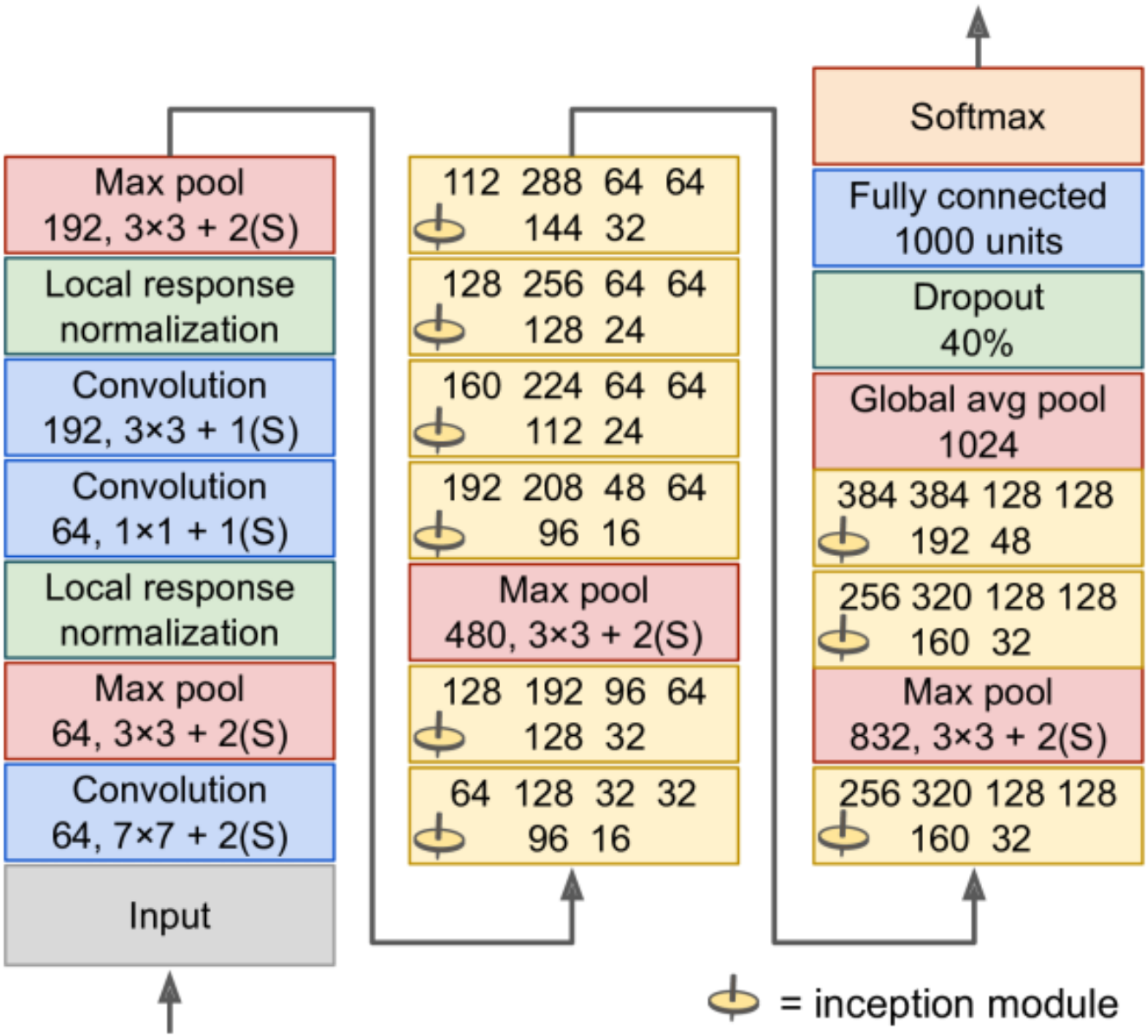
AlexNet

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully connected	–	1,000	–	–	–	Softmax
F9	Fully connected	–	4,096	–	–	–	ReLU
F8	Fully connected	–	4,096	–	–	–	ReLU
C7	Convolution	256	13 × 13	3 × 3	1	`same`	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	`same`	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	`same`	ReLU
S4	Max pooling	256	13 × 13	3 × 3	2	`valid`	–
C3	Convolution	256	27 × 27	5 × 5	1	`same`	ReLU
S2	Max pooling	96	27 × 27	3 × 3	2	`valid`	–
C1	Convolution	96	55 × 55	11 × 11	4	`valid`	ReLU
In	Input	3 (RGB)	227 × 227	–	–	–	–

Figure 16: AlexNet

CNN Architectures

GoogleNet



CNN Architectures

GoogleNet: inception module

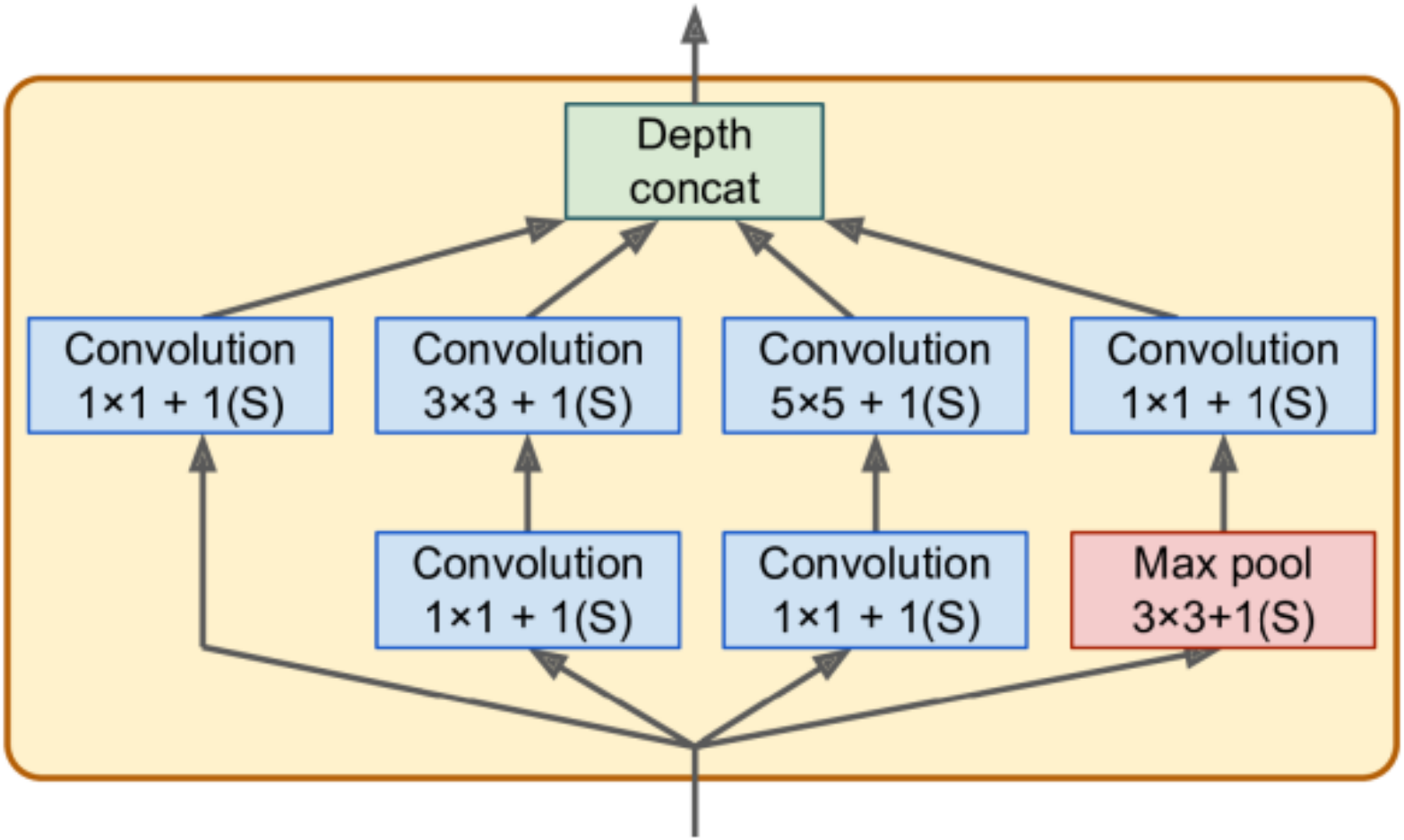


Figure 18: inception

CNN and applications

Classification and Localization

- Localizing an object in a picture can be expressed as a regression task
- To predict a bounding box around the object, a common approach is to predict the horizontal and vertical coordinates of the object's center
- We need to add a second dense output layer with four units

CNN and applications

Classification and Localization

```
base_model = keras.applications.xception.Xception(weights="imagenet",
include_top=False)
avg = keras.layers.GlobalAveragePooling2D()(base_model.output)
class_output = keras.layers.Dense(n_classes, activation="softmax")(avg)
loc_output = keras.layers.Dense(4)(avg)
model = keras.Model(inputs=base_model.input,
outputs=[class_output, loc_output])
model.compile(loss=["sparse_categorical_crossentropy", "mse"],
loss_weights=[0.8, 0.2], # depends on what you care most about
optimizer=optimizer, metrics=["accuracy"])
```

CNN and applications

Classification and Localization

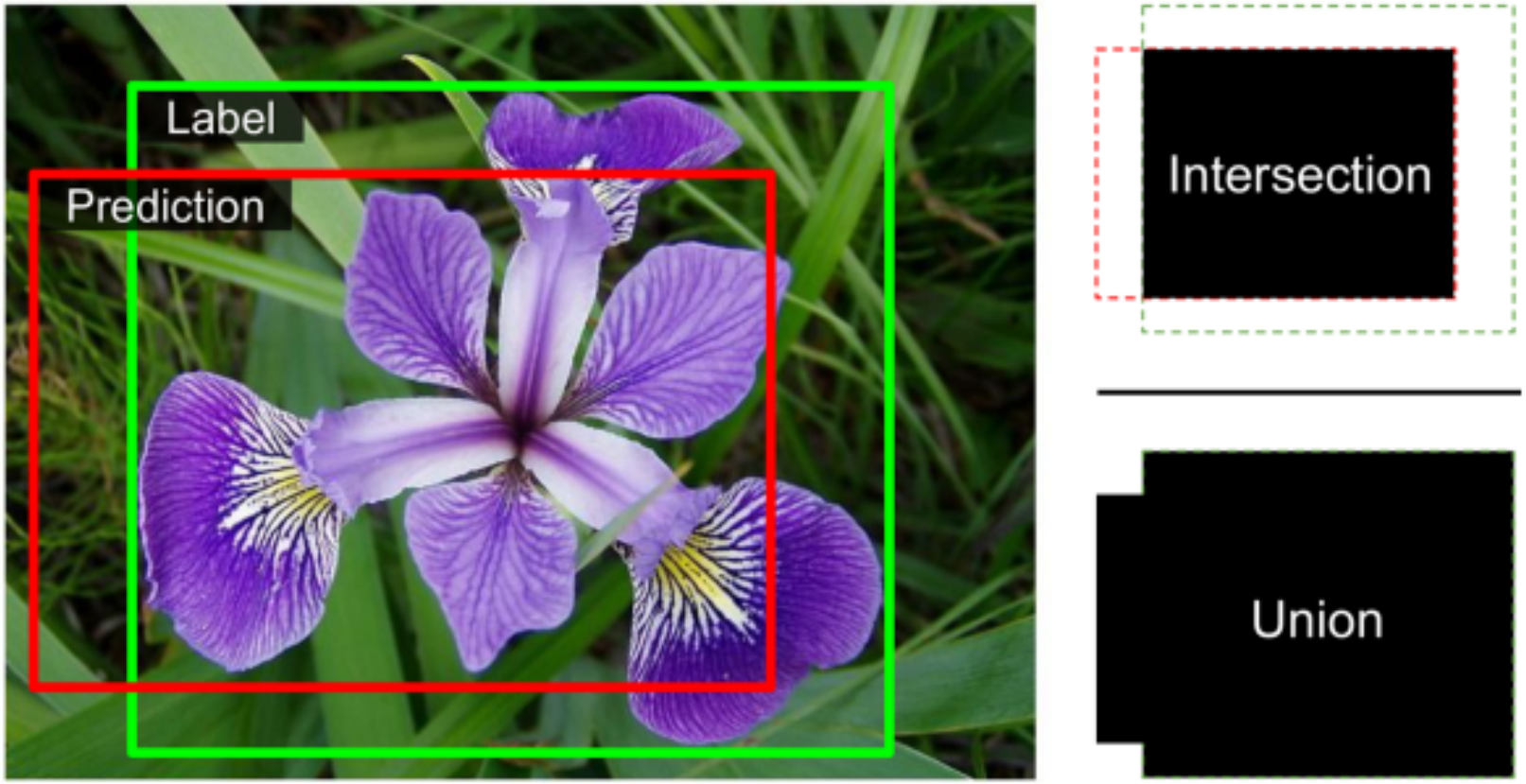


Figure 19: localisation

CNN and applications

Image segmentation

Image segmentation is one of the key applications in the **Computer Vision** domain.

Image segmentation is the task of clustering parts of an image together that belong to the same object class. This process is also called pixel-level classification. In other words, it involves partitioning images (or video frames) into multiple segments or objects.

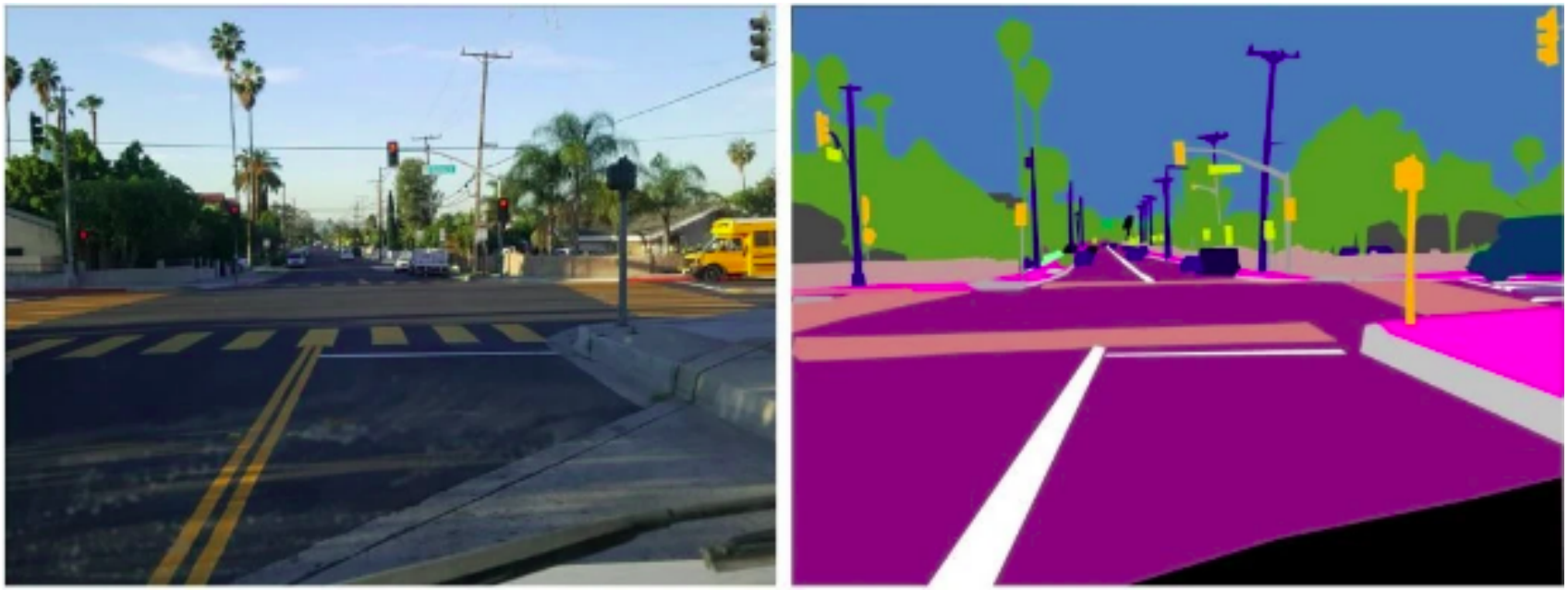


Figure 20: Image Segmentation

CNN and application

Image segmentation

Segmentation is useful and can be used in real-world applications such as medical imaging, clothes segmentation, flooding maps, self-driving cars, etc.

- Medical applications such as tumor boundary extraction or measurement of tissue volumes
- Autonomous vehicles and Advanced Driver Assistance Systems (ADAS)
- Agriculture: precision agriculture

CNN and applications

Image segmentation

There are two types of image segmentation:

- **Semantic segmentation:** classify each pixel with a label.
- **Instance segmentation:** classify each pixel and differentiate each object instance.

CNN and applications

Image segmentation: U-Net

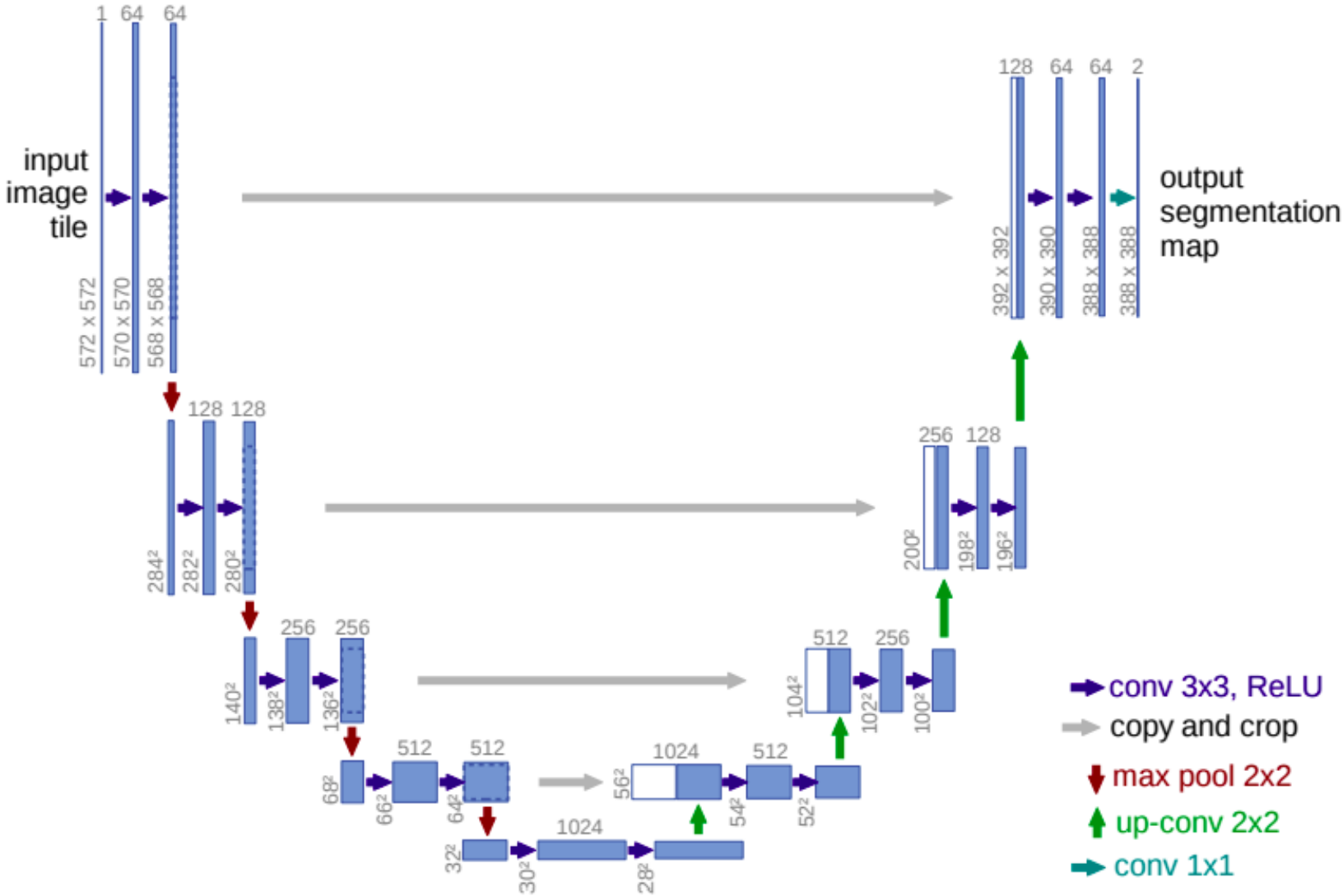


Figure 21: Image Segmentation: U-Net

CNN and application

Image segmentation: U-net in Keras

```
import tensorflow as tf


IMG_WIDTH = 128
IMG_HEIGHT = 128
IMG_CHANNELS = 3
```

CNN and application

Image segmentation: U-net in Keras

```
#Build the model
inputs = tf.keras.layers.Input((IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS))
s = tf.keras.layers.Lambda(lambda x: x / 255)(inputs)

#Contraction path
c1 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu',
kernel_initializer='he_normal',padding='same')(s)
c1 = tf.keras.layers.Dropout(0.1)(c1)
c1 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c1)
p1 = tf.keras.layers.MaxPooling2D((2, 2))(c1)

c2 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(p1)
c2 = tf.keras.layers.Dropout(0.1)(c2)
c2 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c2)
p2 = tf.keras.layers.MaxPooling2D((2, 2))(c2)
```

CNN and application

Image segmentation: U-net in Keras

```
c3 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(p2)
c3 = tf.keras.layers.Dropout(0.2)(c3)
c3 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c3)
p3 = tf.keras.layers.MaxPooling2D((2, 2))(c3)

c4 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(p3)
c4 = tf.keras.layers.Dropout(0.2)(c4)
c4 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c4)
p4 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(c4)

c5 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(p4)
c5 = tf.keras.layers.Dropout(0.3)(c5)
c5 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c5)
```

CNN and application

Image segmentation: U-net in Keras

```
#Expansive path
u6 = tf.keras.layers.Conv2DTranspose(128, (2, 2), strides=(2, 2),
padding='same')(c5)
u6 = tf.keras.layers.concatenate([u6, c4])
c6 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(u6)
c6 = tf.keras.layers.Dropout(0.2)(c6)
c6 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c6)

u7 = tf.keras.layers.Conv2DTranspose(64, (2, 2), strides=(2, 2),
padding='same')(c6)
u7 = tf.keras.layers.concatenate([u7, c3])
c7 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(u7)
c7 = tf.keras.layers.Dropout(0.2)(c7)
c7 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c7)
```

CNN and application

Image segmentation: U-net in Keras

```
u8 = tf.keras.layers.Conv2DTranspose(32, (2, 2), strides=(2, 2),
padding='same')(c7)
u8 = tf.keras.layers.concatenate([u8, c2])
c8 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(u8)
c8 = tf.keras.layers.Dropout(0.1)(c8)
c8 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_n

u9 = tf.keras.layers.Conv2DTranspose(16, (2, 2), strides=(2, 2),
padding='same')(c8)
u9 = tf.keras.layers.concatenate([u9, c1], axis=3)
c9 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(u9)
c9 = tf.keras.layers.Dropout(0.1)(c9)
c9 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c9)
```

CNN and application

Image segmentation: U-net in Keras

```
outputs = tf.keras.layers.Conv2D(1, (1, 1), activation='sigmoid')(c9)

model = tf.keras.Model(inputs=[inputs], outputs=[outputs])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```