BIG DATA SEMINAR

TOPIC: ML-1

Group CQ2021-01

GROUP CQ2021-01

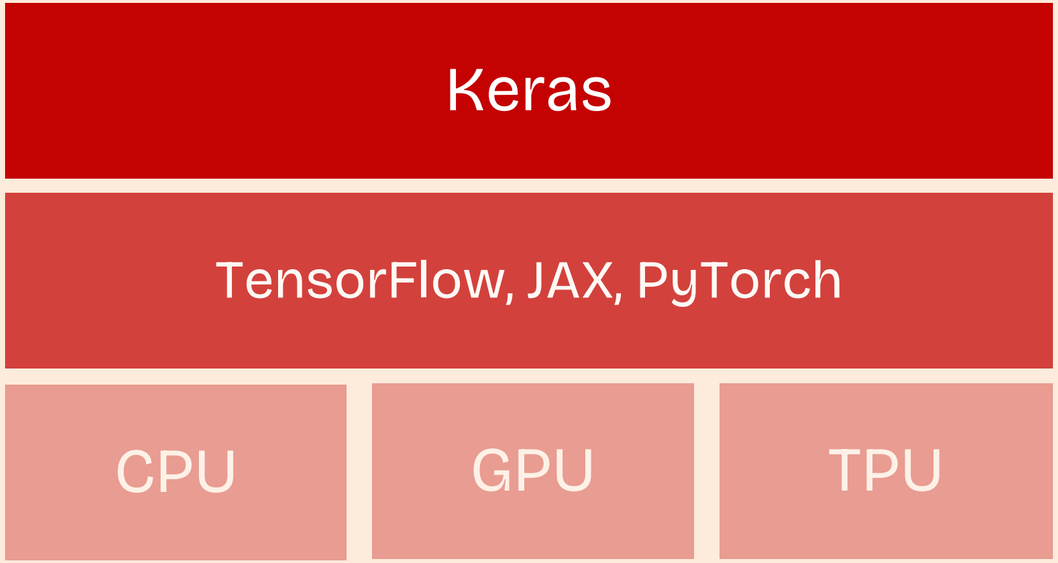
Huỳnh Cao Khôi
21120275

Phạm Lê Tú Nhi
21120308

Chu Hải Linh
21120496

Lê Thị Minh Phương
21120533

What is Keras?



Keras is a **high-level** and **cross-backend** deep learning framework.

But Keras definition is **NOT**
the objective of this presentation.

Our final objective is to show you

HOW TO USE KERAS

What is Keras used for?

Keras is high-level, cross-backend

deep learning

API running on top of lower APIs.

But we might have little idea what **deep learning** is...

... which makes it hard to understand what Keras is doing.

So we will approach Keras with a
“Hello World” problem in Deep Learning

MNIST

handwritten digit classification

CONTENTS

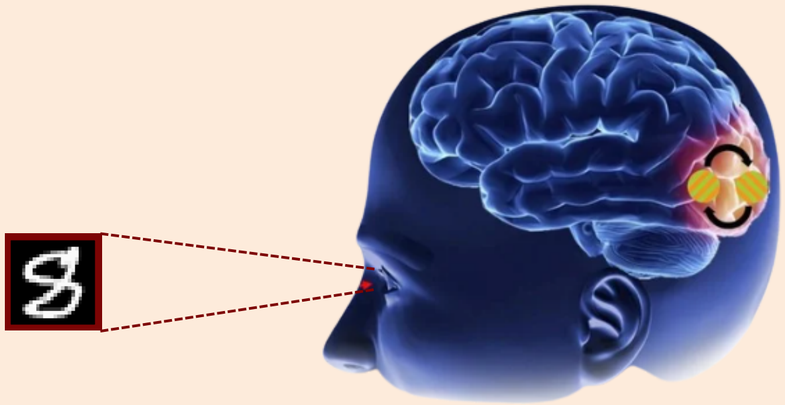
01 Problem

Handwritten digit

MNIST dataset

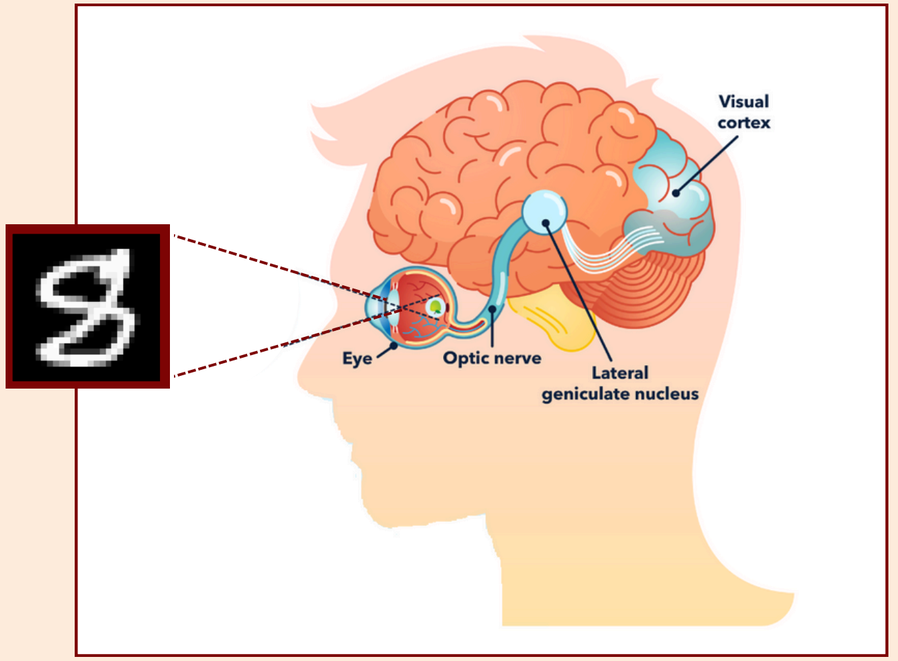
Can you classify the digit depicted in the image?

We may classify the image in the blink of an eye...

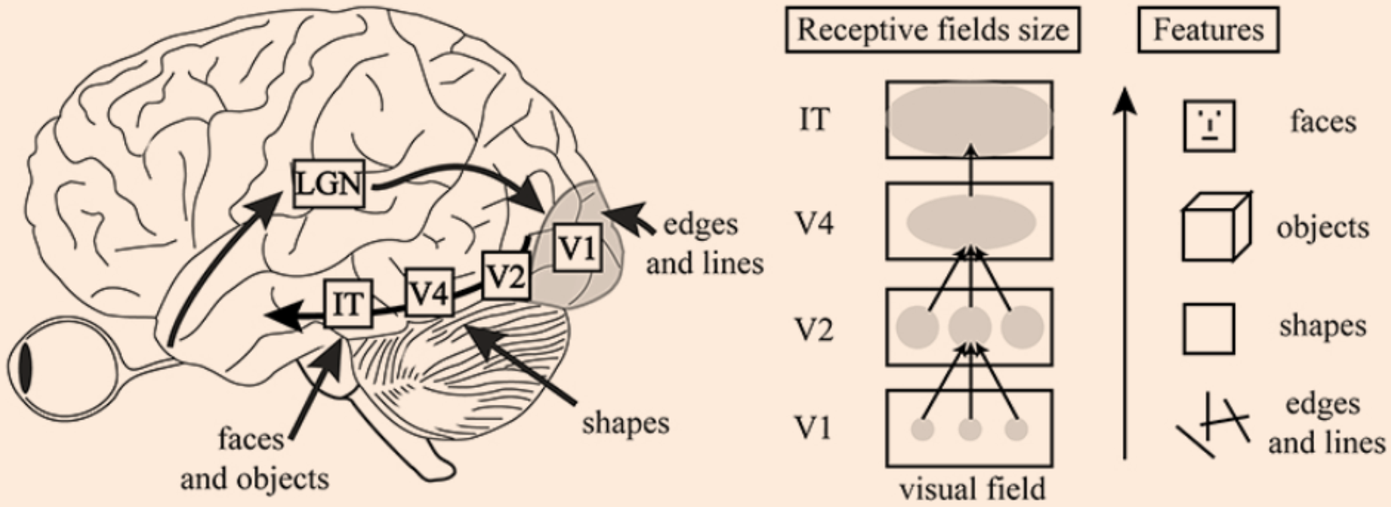


... but the way the brain does it behind the scenes
is very complicated and fascinating!

Visual information is processed in the **visual cortex**.



Features are extracted **hierarchically**, in local **receptive fields**.



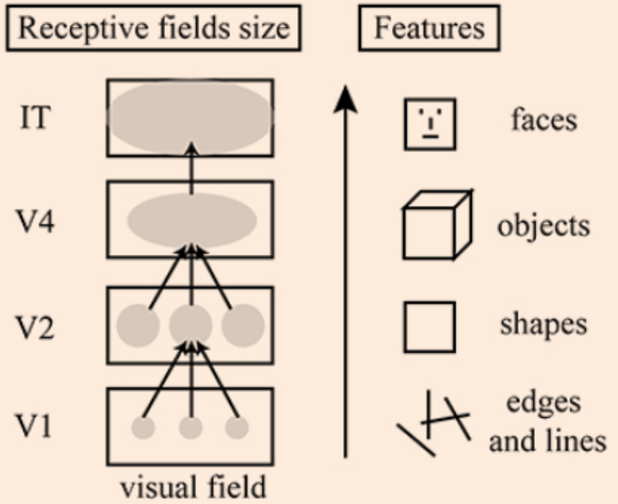
It has inspired the development of CNN

Convolutional Neural Network,

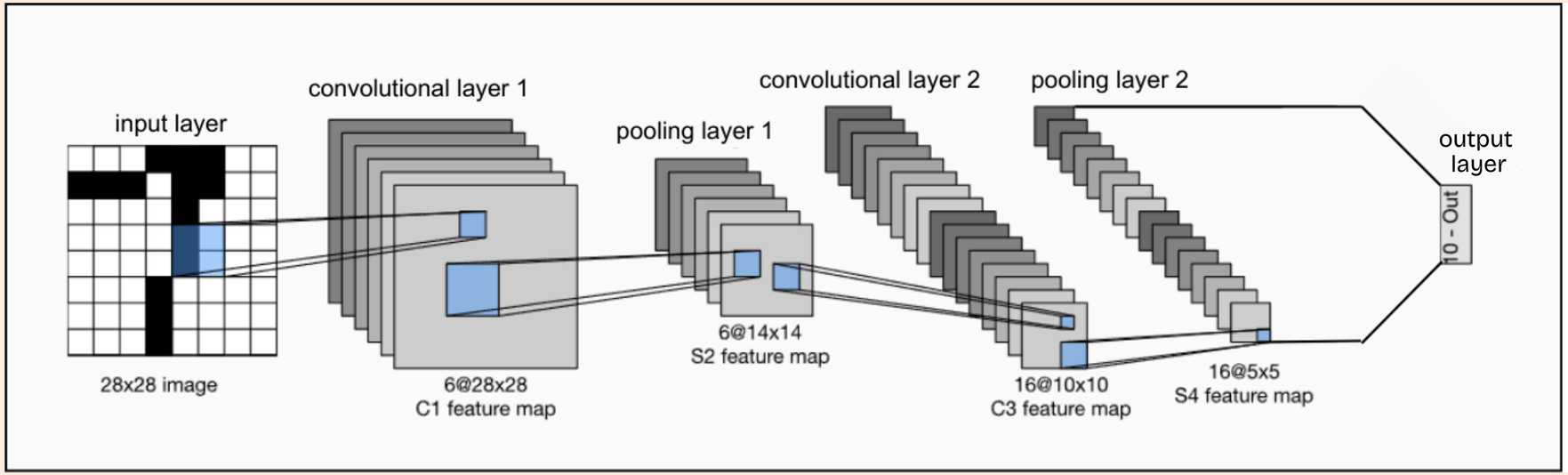
a deep learning model that performs
very well in **visual processing** tasks.

CNNs mimic how the visual cortex processes and recognizes visual patterns.

Local Receptive Fields

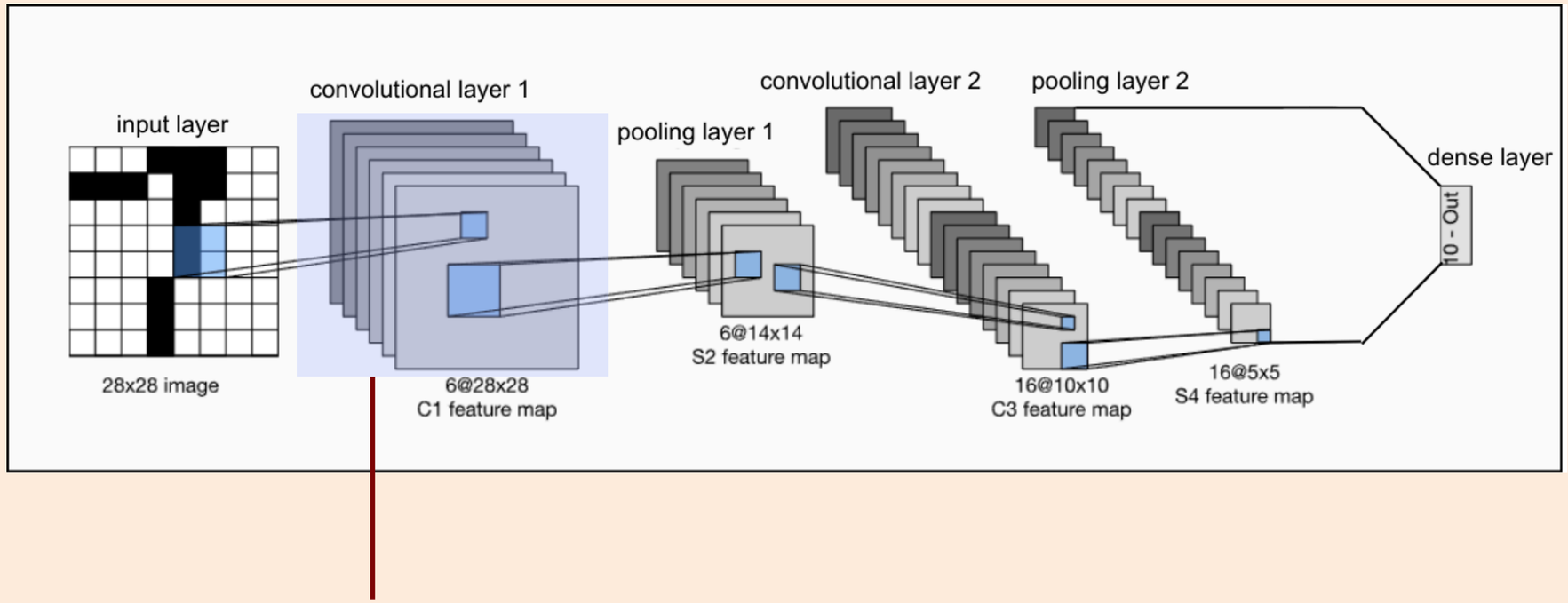


Hierarchical Processing



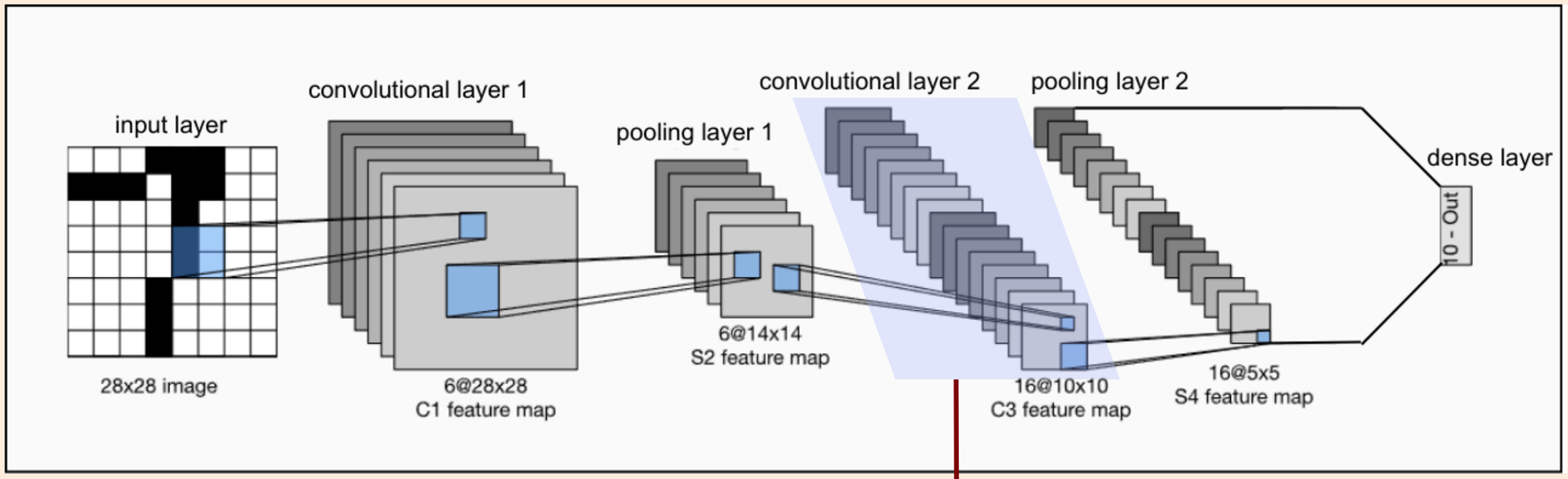
This is a **Convolutional Neural Network**.

Low-level feature extraction



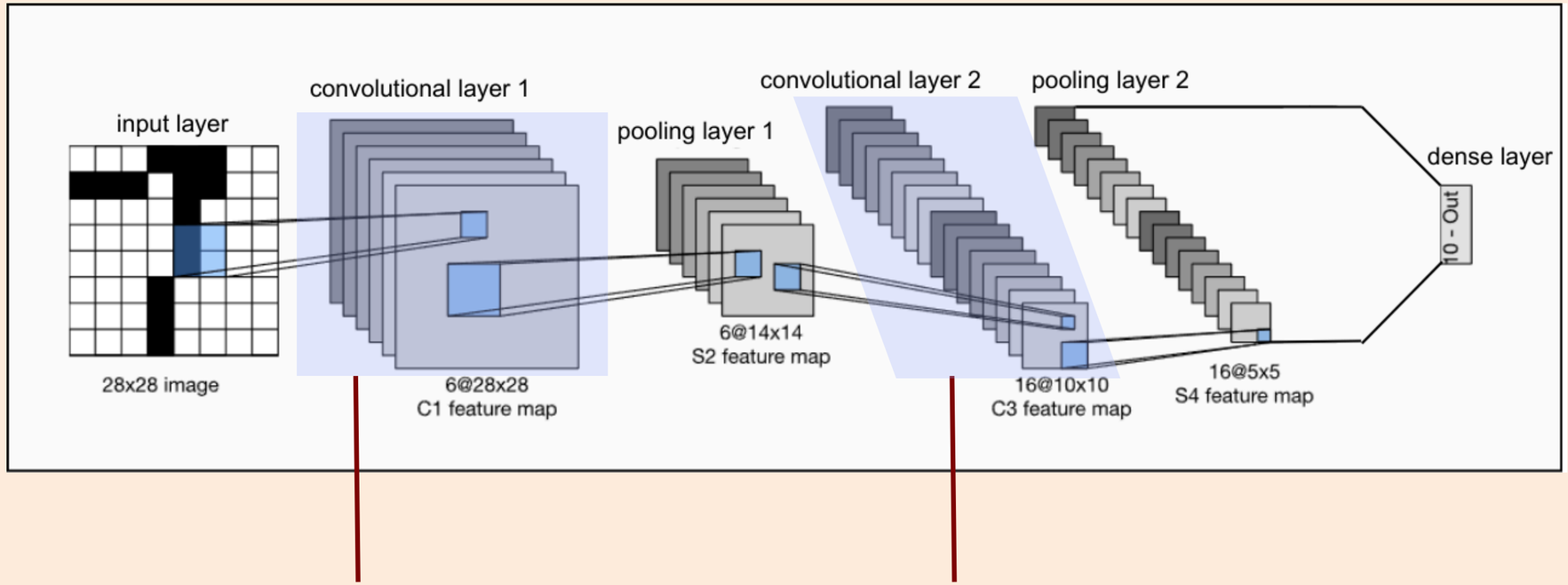
In this layer, **low-level** features (edges like vertical, horizontal and diagonal edges) are extracted.

High-level feature extraction



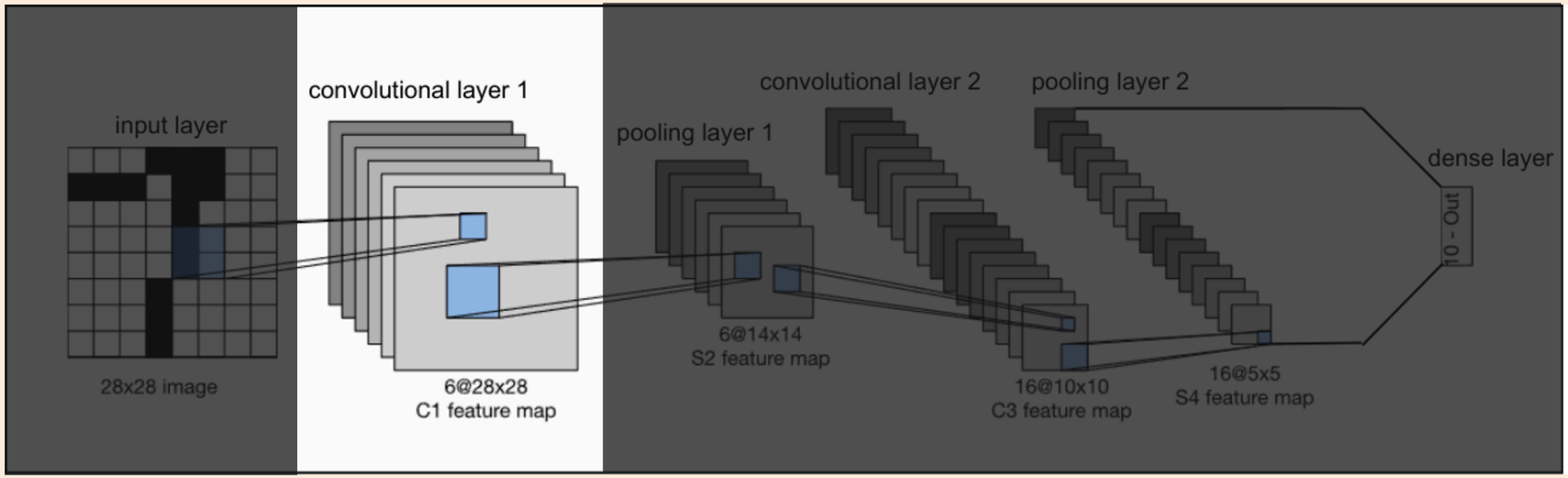
And in this layer, **higher-level** features
(shapes like circle, curve and line) are extracted.

Hierarchical feature extraction

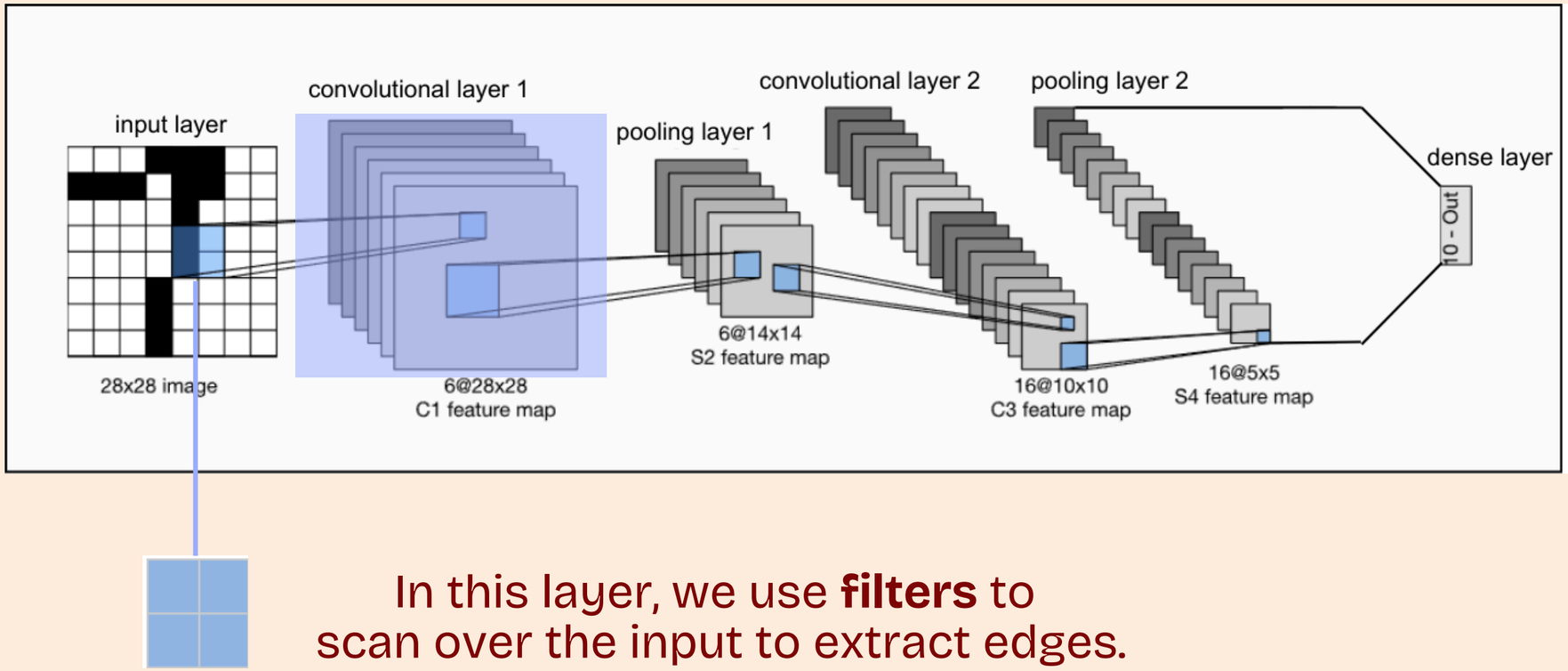


Features of the image are extracted **hierarchically**,
from low to high level.

Let’s zoom in this layer to understand how the model extract features.

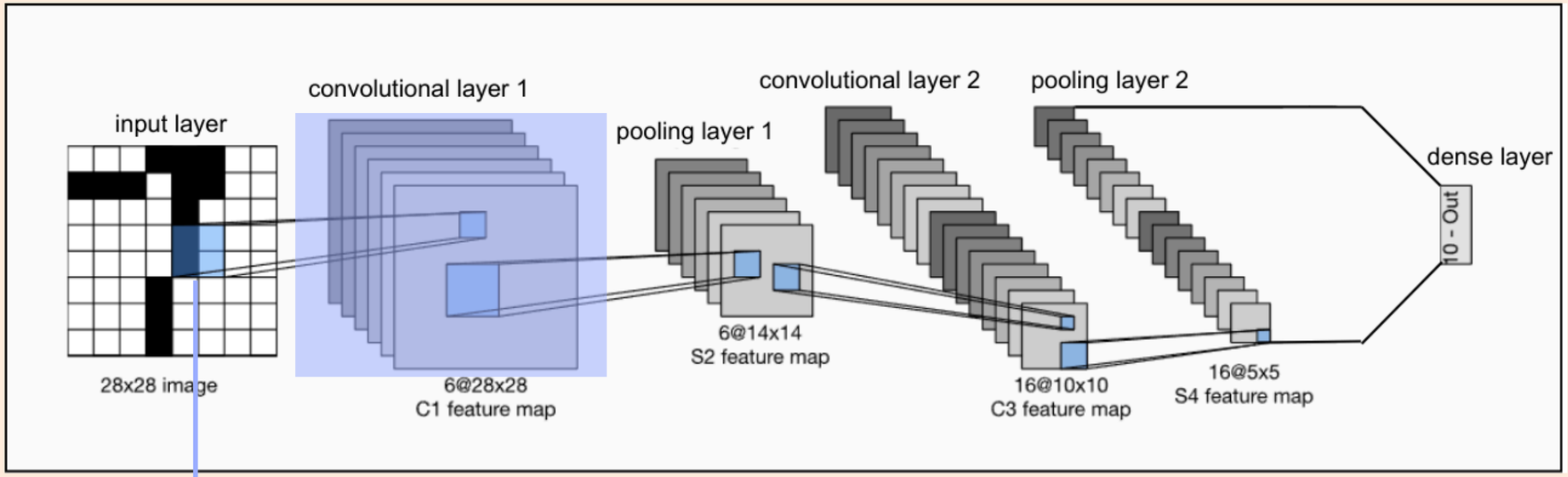


Filter



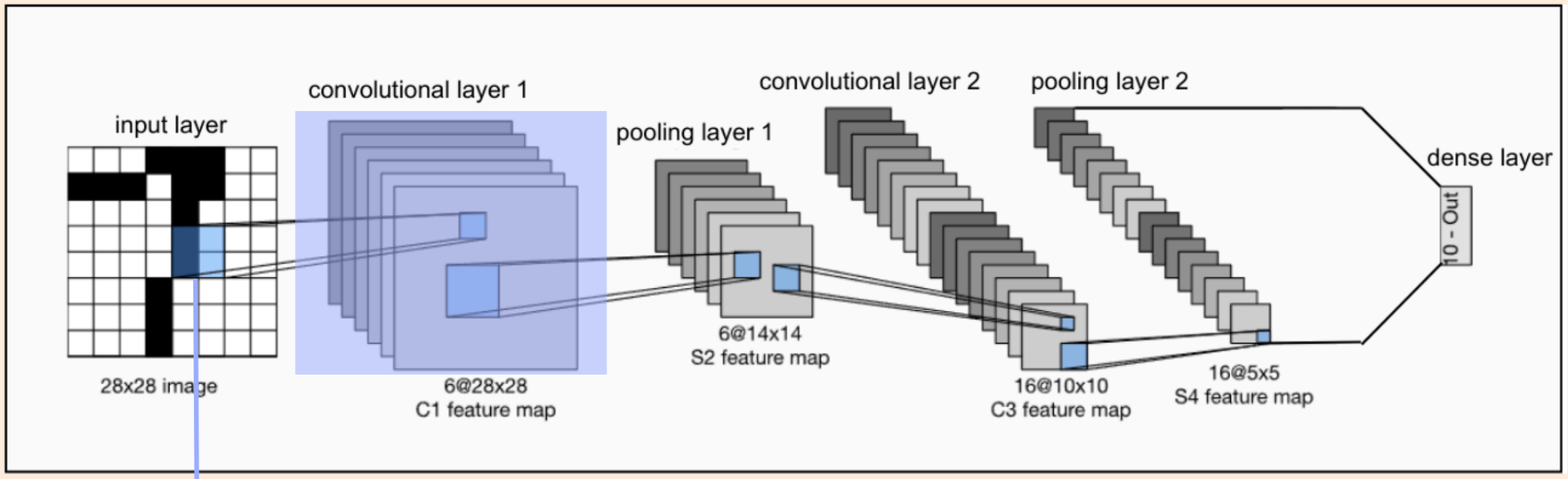
In this layer, we use **filters** to scan over the input to extract edges.

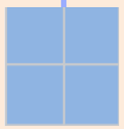
Filter



We call them **kernels.**

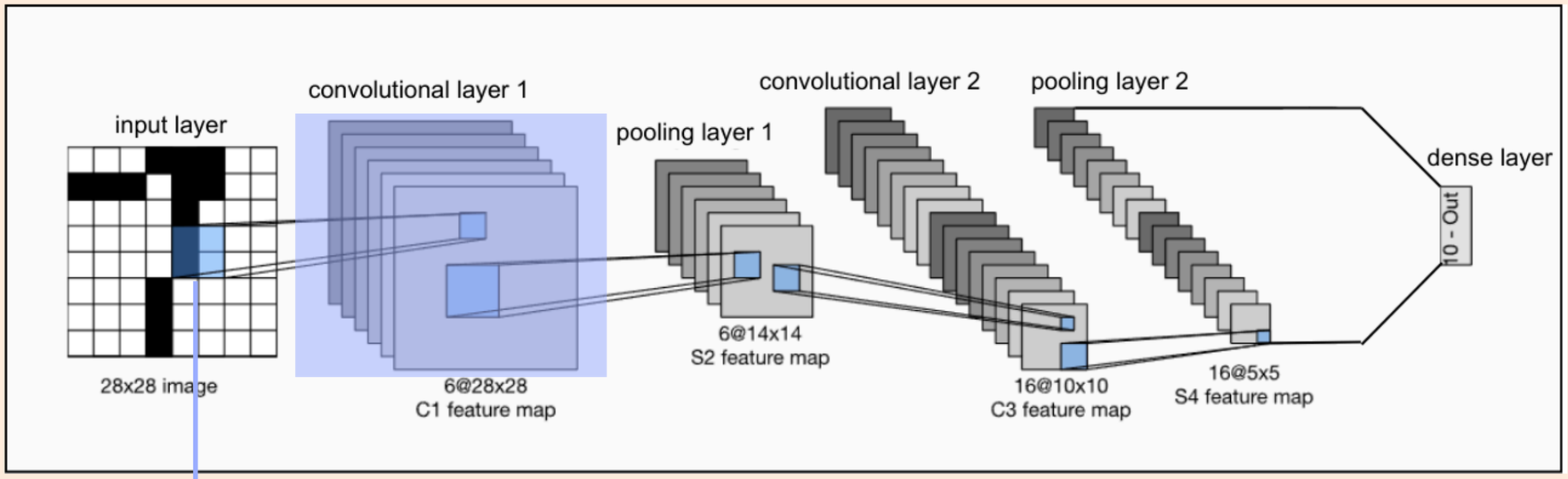
Kernel





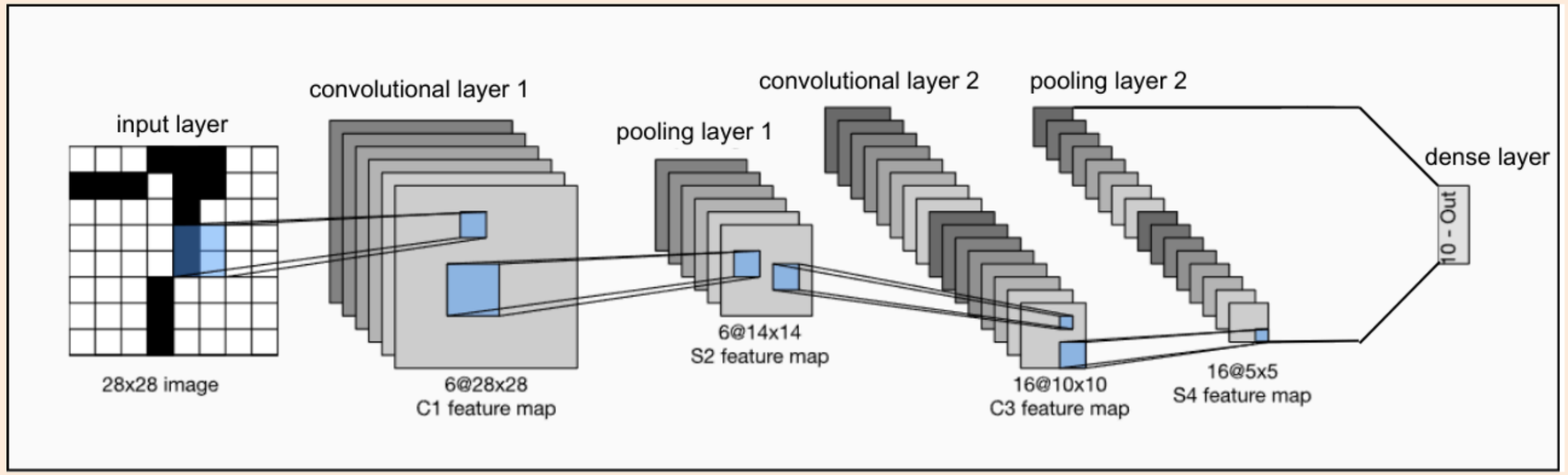
We have different kernels to filter different types of edges: vertical edge filter, horizontal edge filter, and so on.

Local receptive field

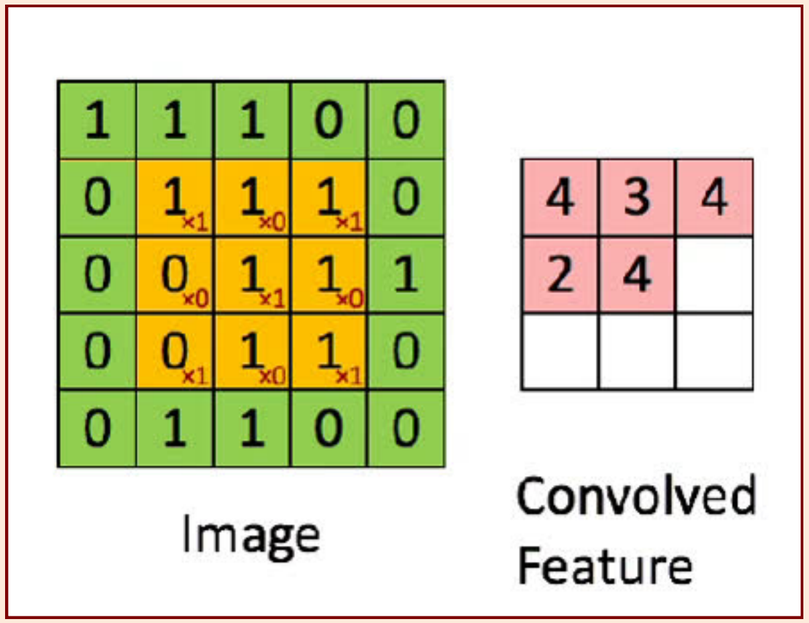


The kernel does not filter for the whole input image
at the same time, but slides over it field by field locally.

Convolution

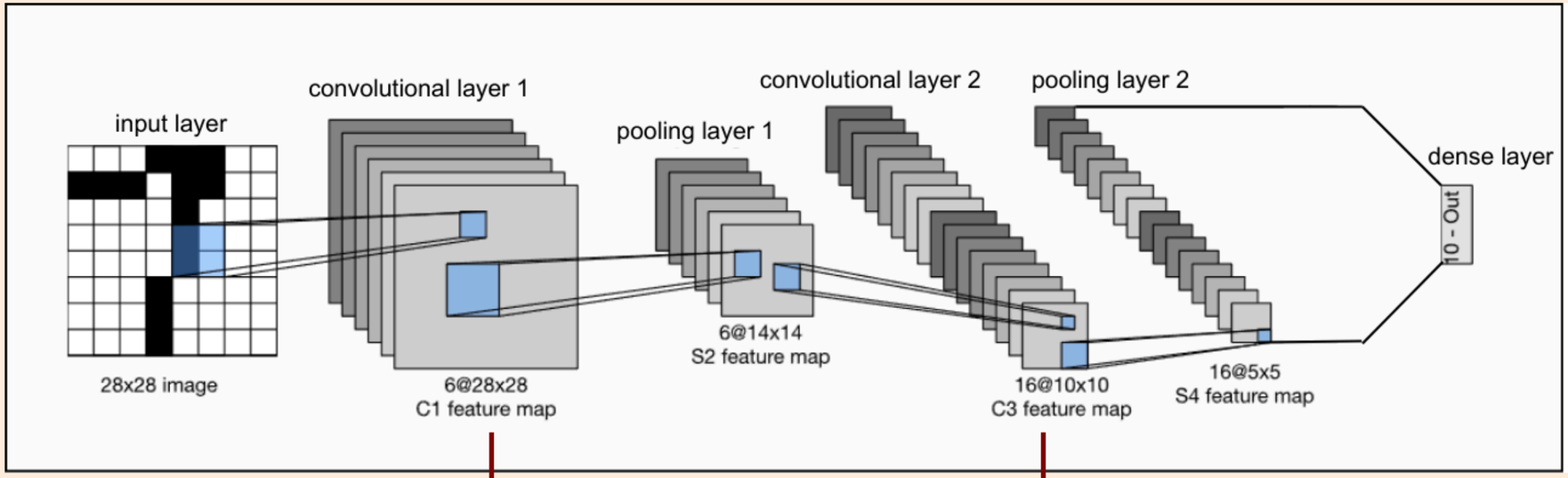


Our computer performs this using an operation called **convolution**.



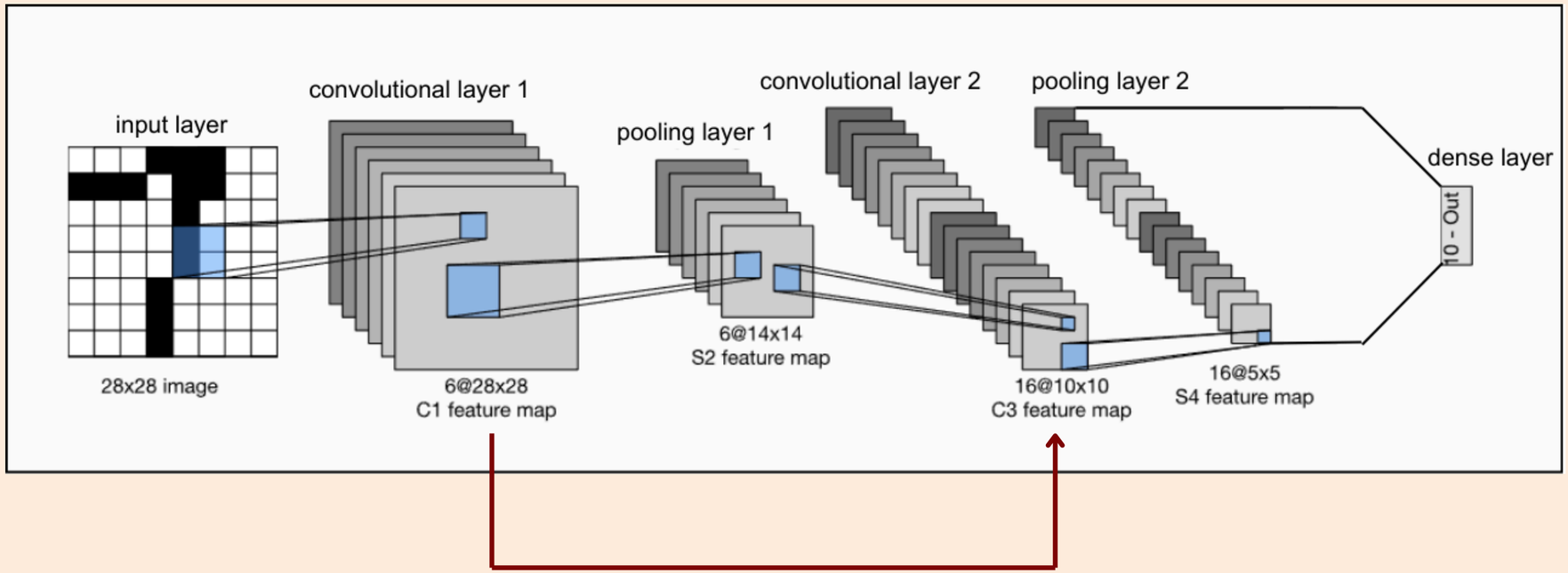
Convolution simply creates an output matrix by **sliding** over the input, **summing** up element-wise **products** between each receptive field and the kernel.

Convolutional layer



Therefore, these layers are called **convolutional layers.**

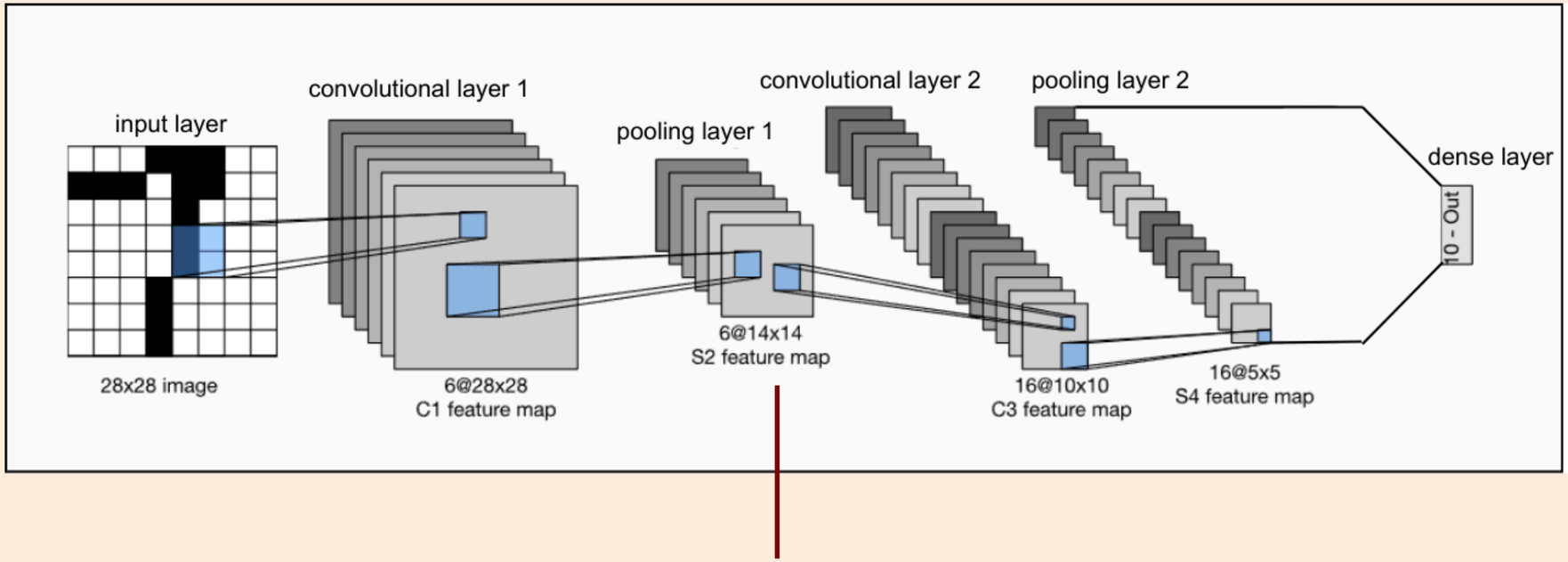
Downsampling



We may want to **reduce the size*** of outputted feature maps before inputting them into deeper layers.

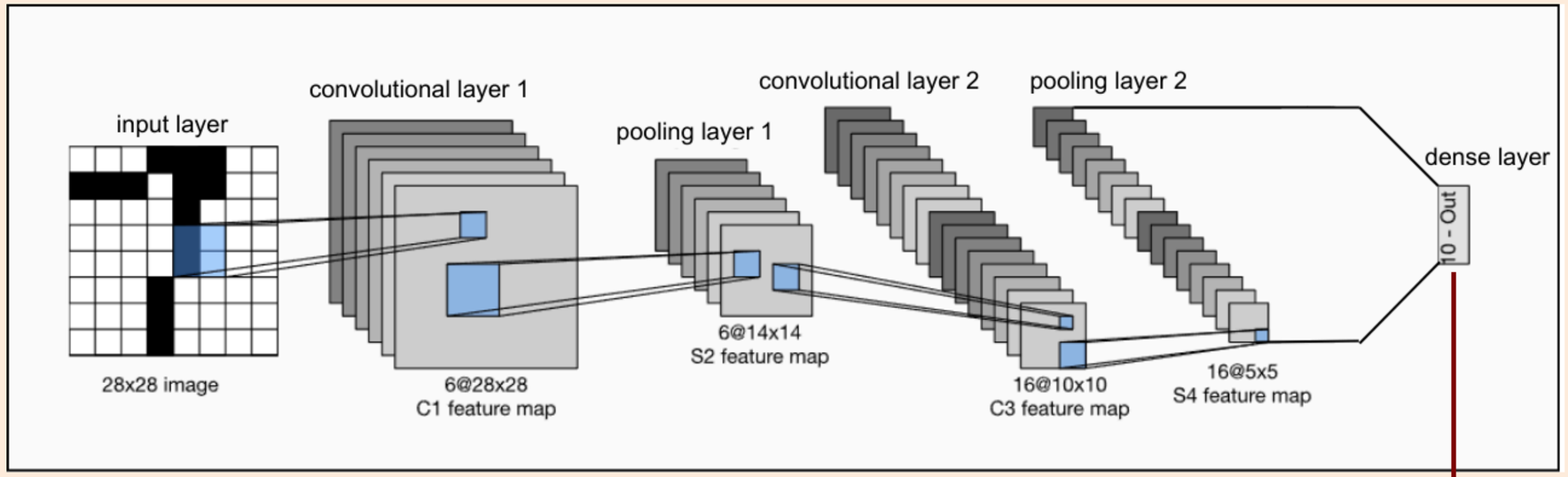
* to reduce computational cost, sensitivity to noise and overfitting

Pooling layer



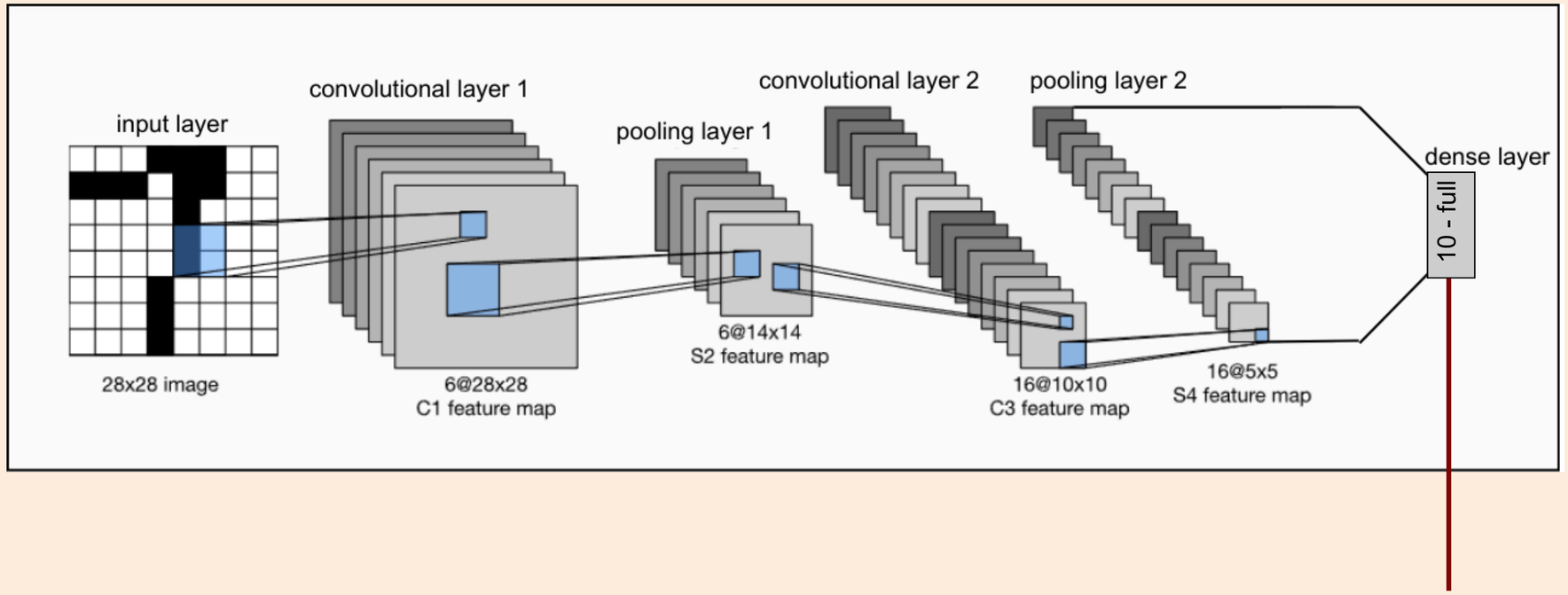
Therefore, we use an intermediate layer called **pooling layer.**
Pooling simply means reducing the size!

Dense layer



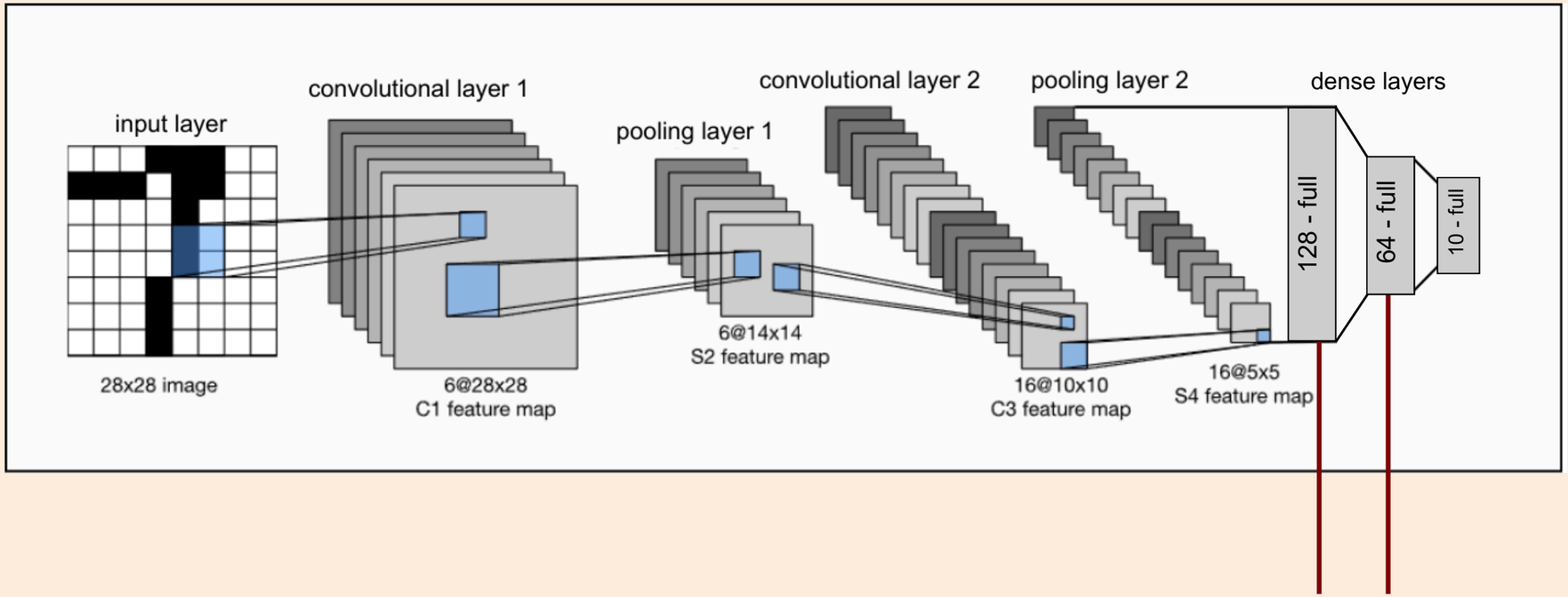
At the end of the network, we perform reasoning based on **all** features to do the classification.

Dense layer



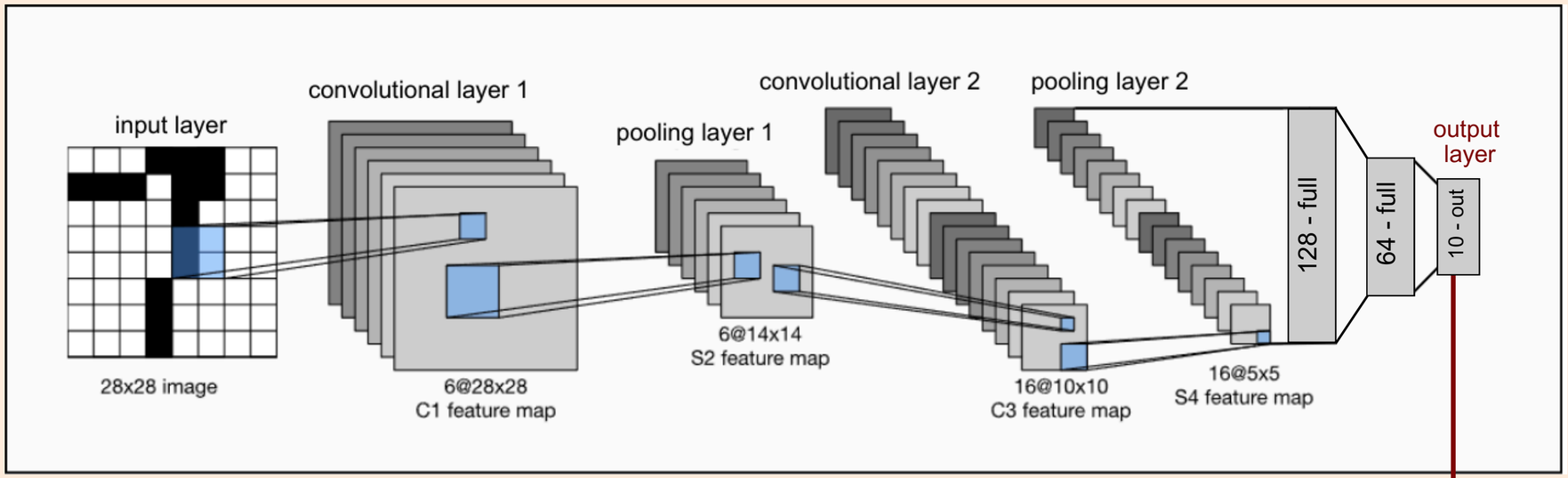
We **connect** each neuron in this layer to **all neurons** of the previous layer.
Therefore, we call it a **fully connected** layer, or **dense** layer.

Dense layer



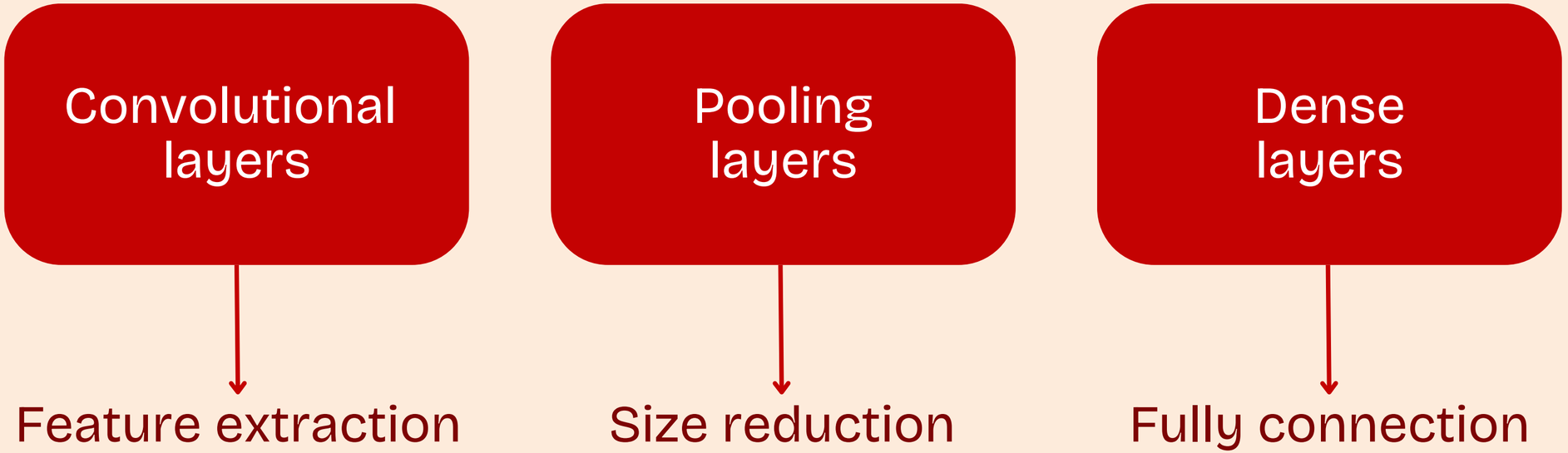
We can add multiple **intermediate dense layers** to make the final classification decision gradually.

Output layer



The final dense layer is called **output layer.**

The idea of our CNN model
is very simple and straightforward!



```
max_epoch = 10
batch_size = 32
tf.reset_default_graph()
input = tf.placeholder(dtype=tf.float32, shape=(None,28,28,1), name='input')
output = tf.placeholder(dtype=tf.float32, shape=(None,10), name='output')

def train_neural_network(x):

    prediction = convolutional_neural_network(x)
    cost = CrossEntropy(prediction, output)
    optimizer = tf.train.AdamOptimizer(learning_rate=0.01).minimize(cost)
    correct = tf.equal(tf.argmax(prediction, 1), tf.argmax(output, 1))
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        indices = np.arange(len(X_train))

        for epoch in range(max_epoch):

            for i in range(0, len(X_train),  batch_size):

                epoch_x = X_train[indices[i : i+batch_size]]
                epoch_y = y_train[indices[i : i+batch_size]]
                a,b,c = sess.run([accuracy, optimizer, cost], feed_dict={input: epoch_x, output:
                    epoch_y})

            result = sess.run([cost, accuracy],feed_dict={input:X_test, output:y_test})
```

But it’s quite complicated to implement, especially for beginners like us.

Don’t worry. Keras will help you!

02 Keras

Keras was created by **Fracois Chollet,**
an AI researcher and sofware engineer, in **2015**.

At that time, Theano was a common framework for DL.

However, it's a **low-level** framework.

Working with low-level framework seems,
most of the time, complicated and time-consuming!

Caffe

He wanted a high-level tool, and Caffe was an option at that time.

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
```

```
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
```

```
layer {
  name: "prob"
  type: "Softmax"
  bottom: "ip2"
  top: "prob"
}
```

In Caffe, we have to define the model architecture in a configuration file called **prototxt**.

It is **inconvenient** and **inflexible.**

And this motivated Francois to create an
easy-to-use and **flexible** high-level deep learning framework...

Keras

Theano (backend engine)

... on top of Theano as backend engine, in 2015.

In 2023, Keras 3 was released as a multi and cross-backend DL framework.

Keras

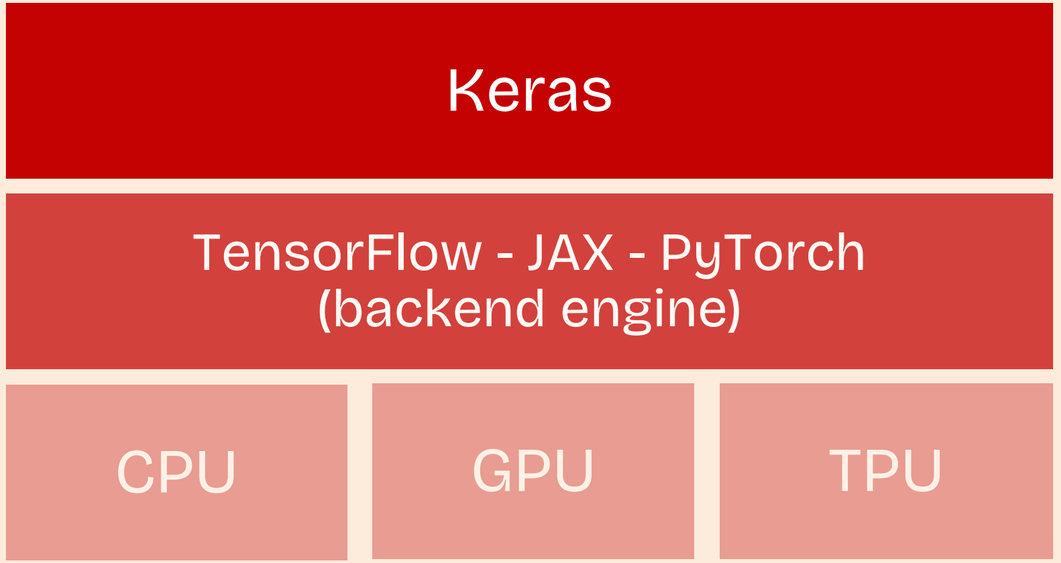
TensorFlow - JAX - PyTorch

* Note: since version 2, Keras has also been integrated into core TensorFlow.

Keras is a

HIGH-LEVEL CROSS-BACKEND

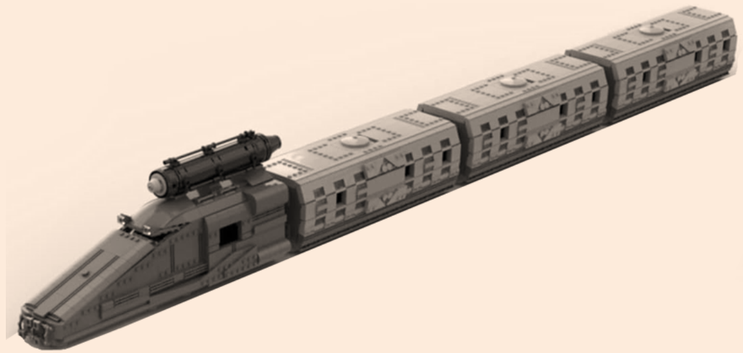
deep learning framework.

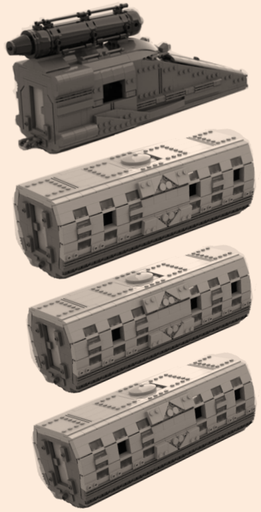


HIGH LEVEL

Keras abstracts deep learning **Layers** and **Models.**

Developer





Keras

Developer

```python
def conv2d(x, kernel, strides=(1, 1), border_mode='valid',
    dim_ordering='th', image_shape=None, filter_shape=None):

    if border_mode == 'same':
        padding = 'SAME'
    elif border_mode == 'valid':
        padding = 'VALID'
    else:
        # raise exception

    strides = (1,) + strides + (1,)

    if _FLOATX == 'float64':
        x = tf.cast(x, 'float32')
        kernel = tf.cast(kernel, 'float32')

    if dim_ordering == 'th':
        x = tf.transpose(x, (0, 2, 3, 1))
        kernel = tf.transpose(kernel, (2, 3, 1, 0))
        x = tf.nn.conv2d(x, kernel, strides, padding=padding)
        x = tf.transpose(x, (0, 3, 1, 2))
    elif dim_ordering == 'tf':
        x = tf.nn.conv2d(x, kernel, strides, padding=padding)
    else:
        raise Exception('Unknown dim_ordering: ' + str(dim_ordering))

    if _FLOATX == 'float64':
        x = tf.cast(x, 'float64')
    return x
```

Keras abstracts **layers**

```python
Conv2D(32, kernel_size=(3,3),
       activation='relu',
       input_shape=(28,28,1),
       padding='SAME')
```

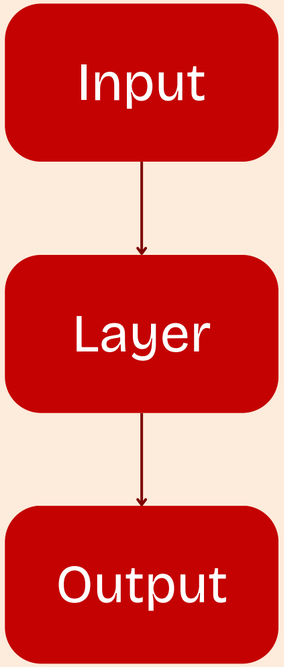
```python
class Sequential(Model):
    def __init__(self, layers=[], name=None)...
    def add(self, layer)...
    def call(self, x, mask=None)...
    def build(self, input_shape=None)...
    def get_weights(self)...
    def set_weights(self, weights)...

    @property
    def validation_data(self)...
    @property
    def training_data(self)...

    def compile(self, optimizer, loss, metrics=[],
                sample_weight_mode=None, **kwargs)...
    def fit(self, x, y, batch_size=32, nb_epoch=10, verbose=1, callbacks=[],
            validation_split=0., validation_data=None, shuffle=True,
            class_weight=None, sample_weight=None, **kwargs)...
    def evaluate(self, x, y, batch_size=32, verbose=1,
                 sample_weight=None, **kwargs)...
    def predict(self, x, batch_size=32, verbose=0)...
    def predict_on_batch(self, x)...
    def test_on_batch(self, x, y, sample_weight=None, **kwargs)...
```

```python
def add(self, layer):
    if not self.outputs:
        if len(layer.inbound_nodes) == 0:
            # create an input layer
            if not hasattr(layer, 'batch_input_shape'):
                # raise Exception
            batch_input_shape = layer.batch_input_shape
            if hasattr(layer, 'input_dtype'):
                input_dtype = layer.input_dtype
            else:
                input_dtype = None
            layer.create_input_layer(batch_input_shape, input_dtype)

        if len(layer.inbound_nodes) != 1:
            # raise Exception

        if len(layer.inbound_nodes[0].output_tensors) != 1:
            # raise Exception

        self.outputs = [layer.inbound_nodes[0].output_tensors[0]]
        self.inputs = get_source_inputs(self.outputs[0])

        # create an input node
    else:
        # add output layer

    self.layers.append(layer)
    self.built = False
```

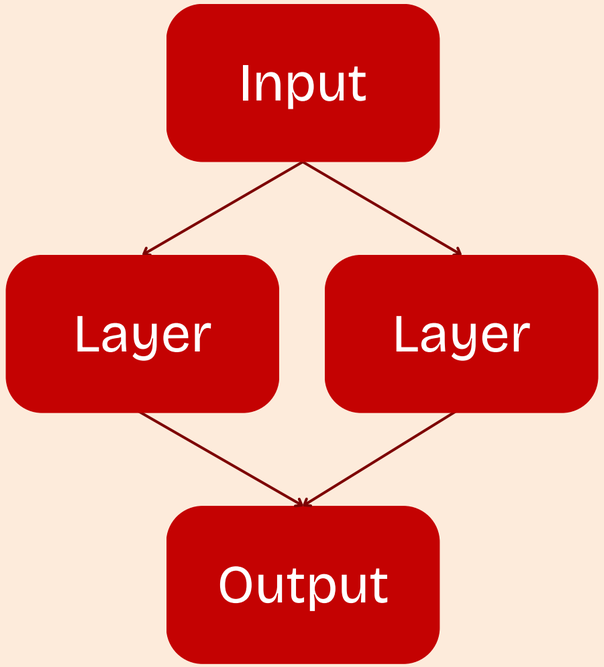
Keras abstracts **models**

```python
model = Sequential()
model.add(Conv2D(...))
model.add(MaxPooling2D(...))
model.add(Conv2D(...))
...
model.add(Dense(...))

model.compile(loss=..., optimizer=..., metrics=...)

history_data = model.fit(...)

loss,accuracy = model.evaluate(...)
```

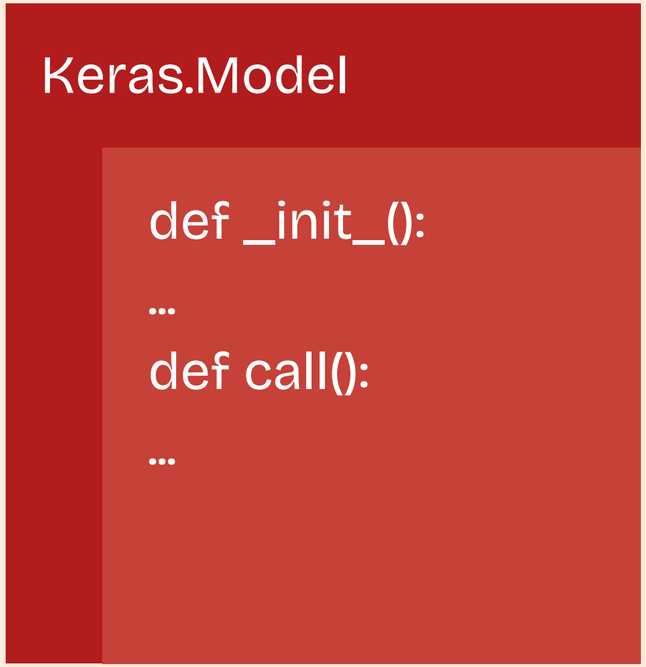
Keras abstracts 3 types of models



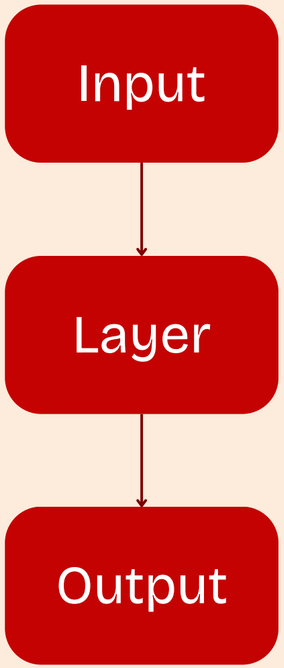
Sequential



Functional API



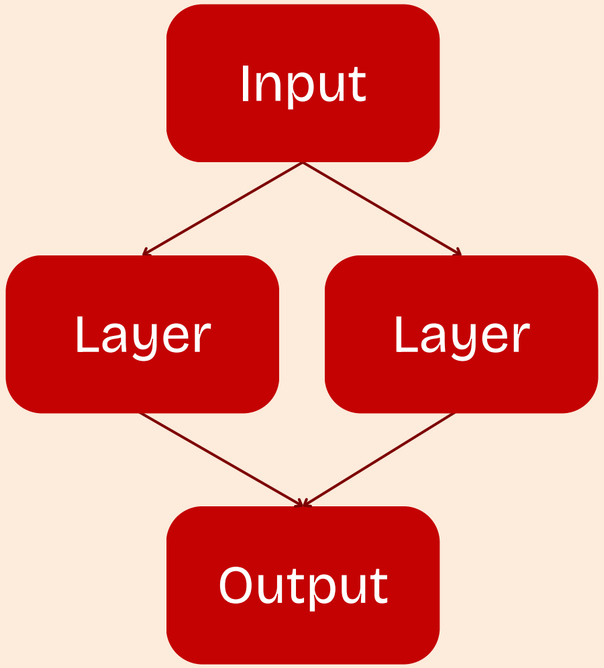
Model Subclassing



Sequential Model

```
# Build the model
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])
```

Functional API




```
# Define the input
inputs = layers.Input(shape=(28, 28, 1))

# Define two parallel layers
layer1 = layers.Conv2D(32, (3, 3), activation='relu')(inputs)
layer2 = layers.Conv2D(32, (5, 5), activation='relu')(inputs)

# Merge the outputs of the two layers
merged = layers.concatenate([layer1, layer2])

# Add more layers after merging
x = layers.MaxPooling2D((2, 2))(merged)...
outputs = layers.Dense(10, activation='softmax')(x)
model = Model(inputs=inputs, outputs=outputs)
```

Model Subclassing

Keras.Model

```
def _init_():
...
def call():
...
```

```python
class MyModel(keras.Model):
    def __init__(self):
        super(MyModel, self).__init__()
        self.flatten = Flatten()
        self.dense1 = Dense(128, activation='relu')
        self.dense2 = Dense(10, activation='softmax')

    def call(self, inputs):
        x = self.flatten(inputs)
        x = self.dense1(x)
        return self.dense2(x)

# Instantiate the model
model = MyModel()
```

We can compile all these models with this simple code!

```
# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train, epochs=5, batch_size=32, validation_data=(x_test, y_test))

# Evaluate model
model.evaluate(x_test, y_test)
```

CROSS BACKEND

Keras allows working across TensorFlow, PyTorch and JAX.

One codebase for all backends with **keras.ops**

```python
import keras
from keras import ops

class TokenAndPositionEmbedding(keras.Layer):
    def __init__(self, max_length, vocab_size, embed_dim):
        super().__init__()
        self.token_embed = self.add_weight(
            shape=(vocab_size, embed_dim),
            initializer="random_uniform",
            trainable=True,
        )
        self.position_embed = self.add_weight(
            shape=(max_length, embed_dim),
            initializer="random_uniform",
            trainable=True,
        )

        def call(self, token_ids):
            # Embed positions
            length = token_ids.shape[-1]
            positions = ops.arange(0, length, dtype="int32")
            positions_vectors = ops.take(self.position_embed, positions, axis=0)
            # Embed tokens
            token_ids = ops.cast(token_ids, dtype="int32")
            token_vectors = ops.take(self.token_embed, token_ids, axis=0)
            # Sum both
            embed = token_vectors + positions_vectors
            # Normalize embeddings
            power_sum = ops.sum(ops.square(embed), axis=1, keepdims=True)
            return embed / ops.sqrt(ops.maximum(power_sum, 1e-7))
```

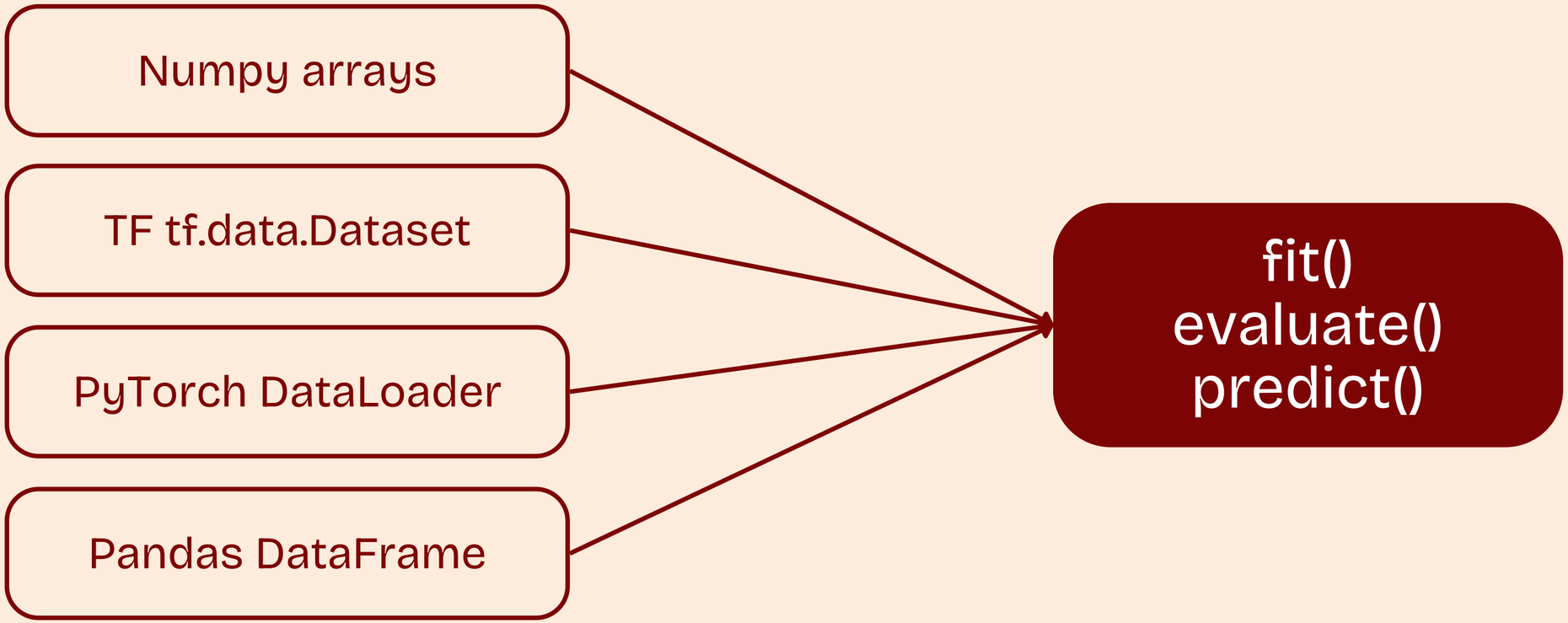
```python
import os
os.environ["KERAS_BACKEND"] = "jax"
import keras
```

```python
import os
os.environ["KERAS_BACKEND"] = "tensorflow"
import keras
```

```python
import os
os.environ["KERAS_BACKEND"] = "torch"
import keras
```

You just have to specify which backend to run the code with!

Cross-framework **data pipelines**



This helps leverage the backends easily!

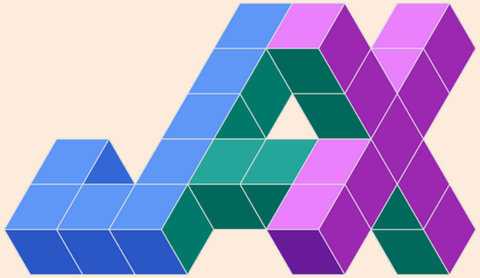
PyTorch

PyTorch geometric

OpenMined

Skorch

Development



Large scale TPU training

Large scale parallelism

Training

TensorFlow

TF-Serving

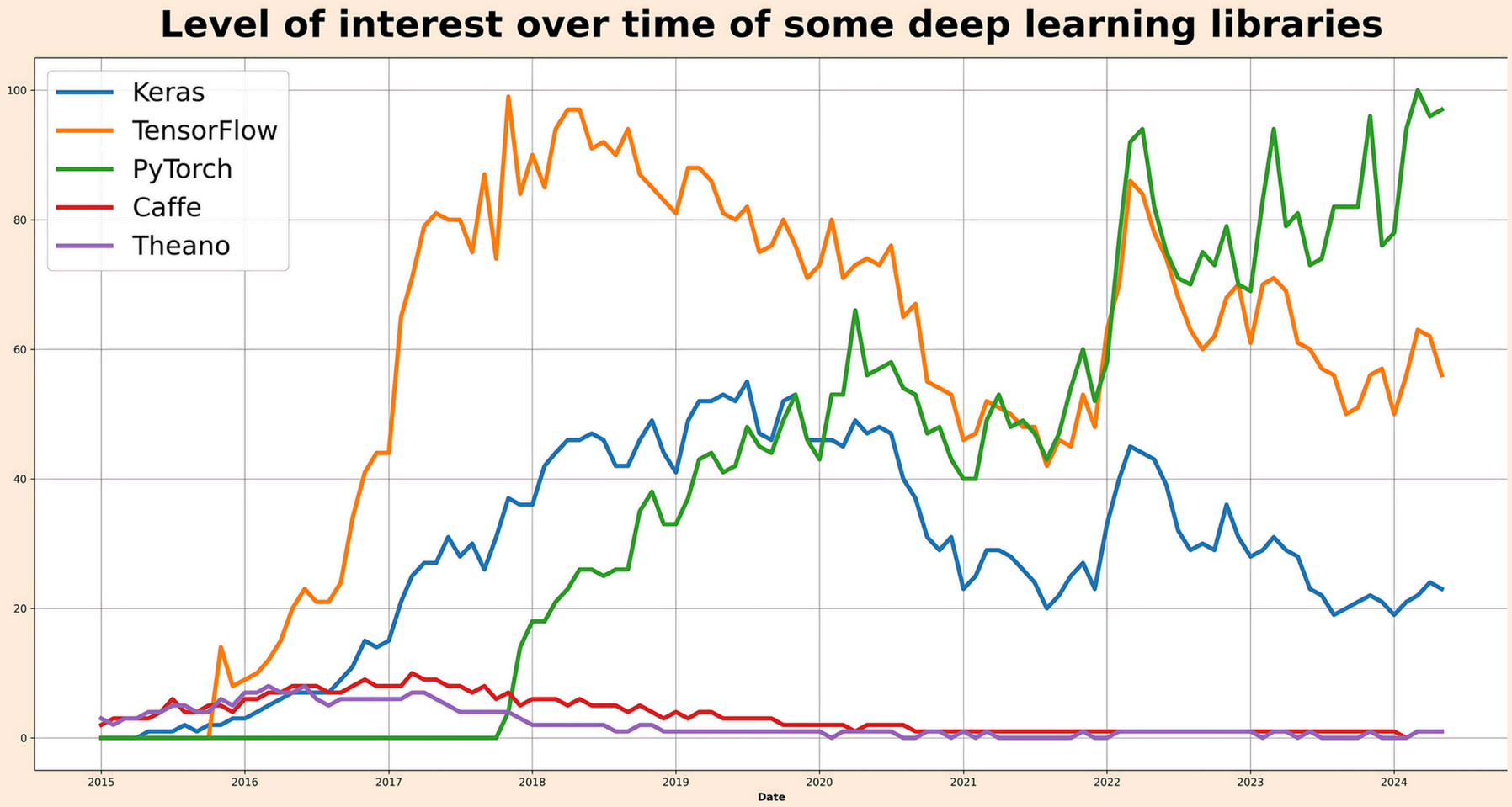
TF.js

TFLite

Deployment

03 Demonstration

Conclusion


Level of interest over time of some deep learning libraries
Keras
TensorFlow
PyTorch
Caffe
Theano
100
80
60
40
20
0
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
Date

Keras VS TensorFlow

Developed by Google Brain
Released in 2015
Supply Tensors library
High & Low-level DL framework

PyTorch

Developed by Facebook
Released in 2016
Based on Torch library
Low-level DL framework

Comparison

	Keras	TensorFlow	Pytorch
Level	High	High, Low	Low
Written in	Python	C++, CUDA, Python	Python, C++
Ease of use	Very high	Low	High
Development speed	Faster	Slower	
Flexibility	Lower	Higher	

When to use

	Keras	TensorFlow	Pytorch
Beginners	✓		✓
Rapid prototyping	✓		✓
Flexibility required		✓	✓
Scalability required		✓	
Deployment required		✓	

Keras' Application

YouTube

Uber

Industrial Applications

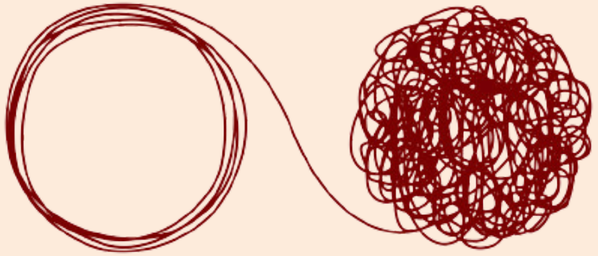
CERN

NASA

Scientific Organizations

Thank you

for your attention!



Group CQ2021-01

Q&A