Deep Learning for **Computer Vision**

Ahmed Hosny Abdel-Gawad
Senior AI/CV Engineer

Table Content

1 From Traditional CV to DL

1.1 From Traditional ConvNets to DL

Semantic gap and CV tasks

The gap between what a computer sees and what we want it to see.

Traditional CV Pipeline

Traditional images processing techniques and its limitations.

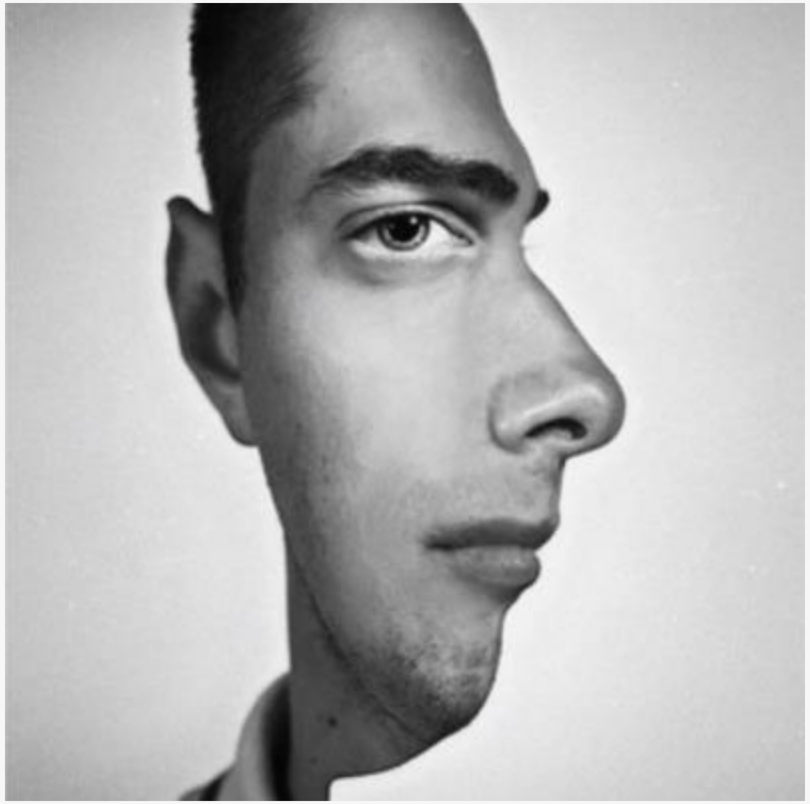
From traditional to learnable convolution filters

Introducing the learning processes and convolution neural networks.

ML: Data-Driven Approach

Engineered Features to do Face Detection: Viola Jones

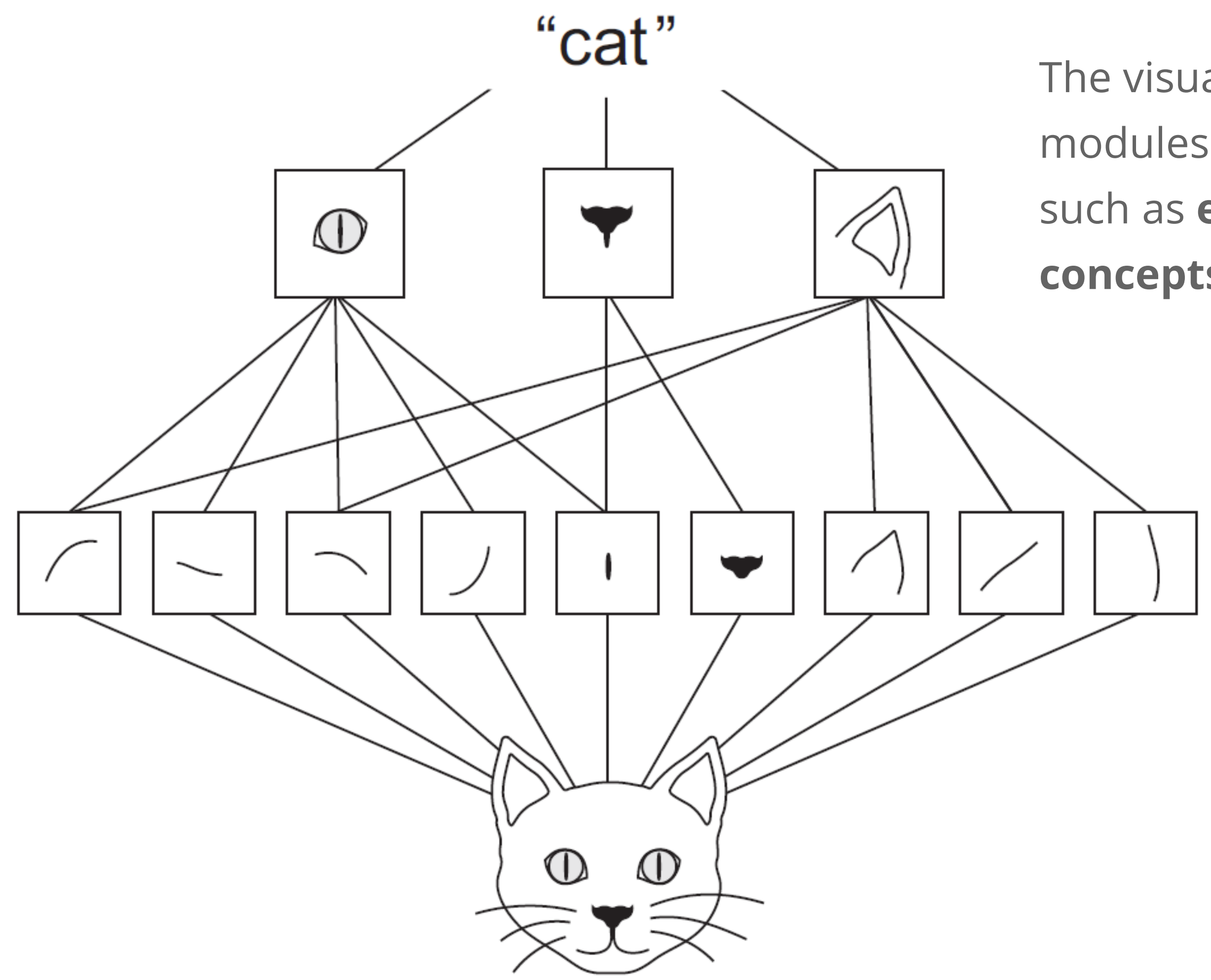
How do we recognize image?



Is it a man with the right face? Or is it a man who is looking at you directly?

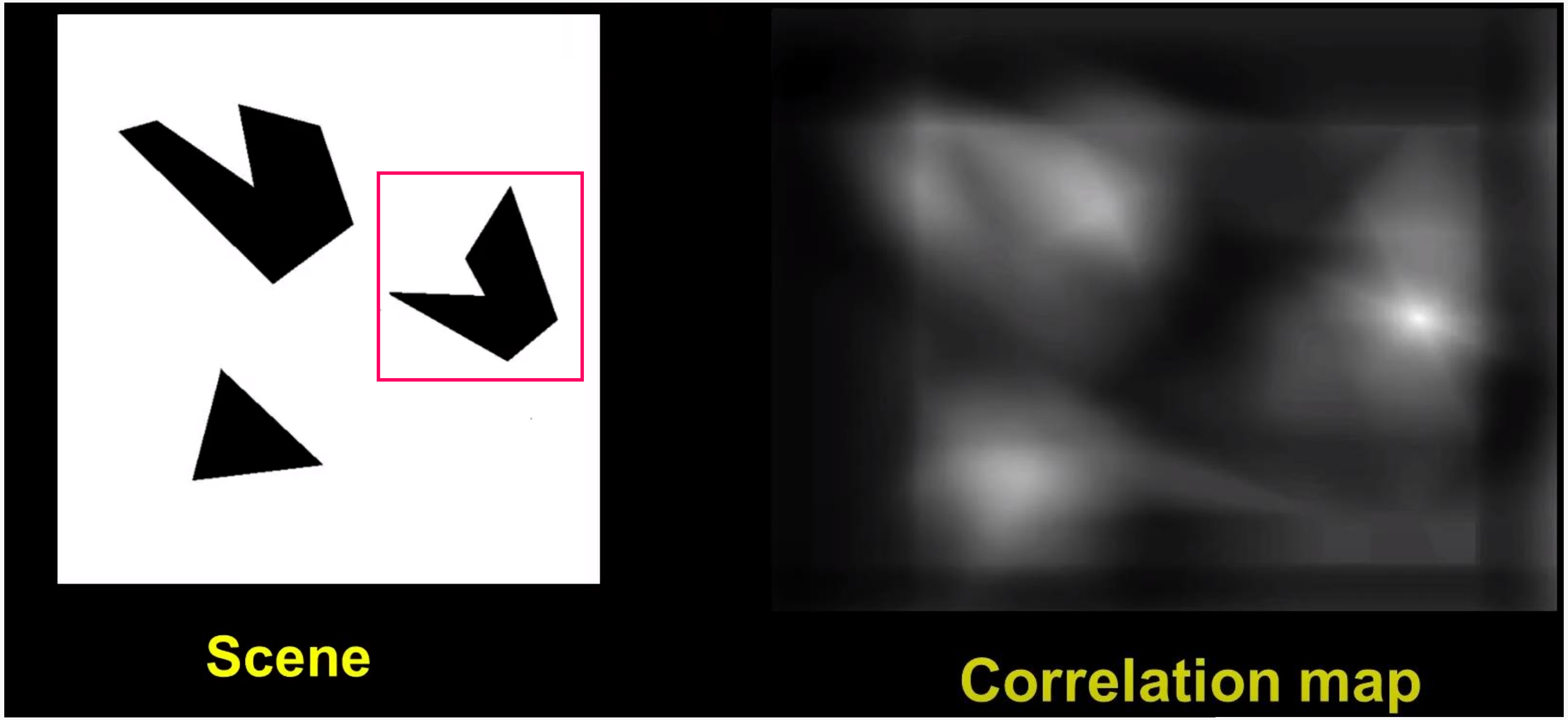
Just a lady who is walking behind the tree?

What could be good features?



The visual world forms a **spatial hierarchy** of visual modules: hyperlocal **edges** combine into **local objects** such as **eyes** or **ears**, which combine into high-level **concepts** such as "**cat**."

Features as template matching



Filter = kernel = Feature = Template

→ Convolution Operation = Feature Extraction = Template Matching

How to decide on the kernel weights?

Method 1: Hand Crafted, e.g. Thresholding or Edges

Method 2: **Machine** Learning, e.g. Engineered Features

Method 3: **Deep** Learning, e.g. Convolution Neural Networks (CNNs)

Why learn the weights not hand craft them?

Because we cannot encode all cases!
Like pose, rotation, illumination, ...etc.

→ **Leave it to data to guide the algorithm!**

1.1 From Traditional ConvNets to DL

Semantic gap and CV tasks

The gap between what a computer sees and what we want it to see.

Traditional CV Pipeline

Traditional images processing techniques and its limitations.

From traditional to learnable convolution filters

Introducing the learning processes and convolution neural networks.

ML: Data-Driven Approach

Engineered Features to do Face Detection: Viola Jones

Machine Learning: Data-Driven Approach

Face Detection: Viola Jones Algorithm

Face detection is basically a **classification** task so it's trained to classify whether there is a target object or not.

Given an image, the algorithm looks at many smaller subregions and tries to find a face by looking for specific features in each subregion. It needs to check many different positions and scales because an image can contain many faces of various sizes.

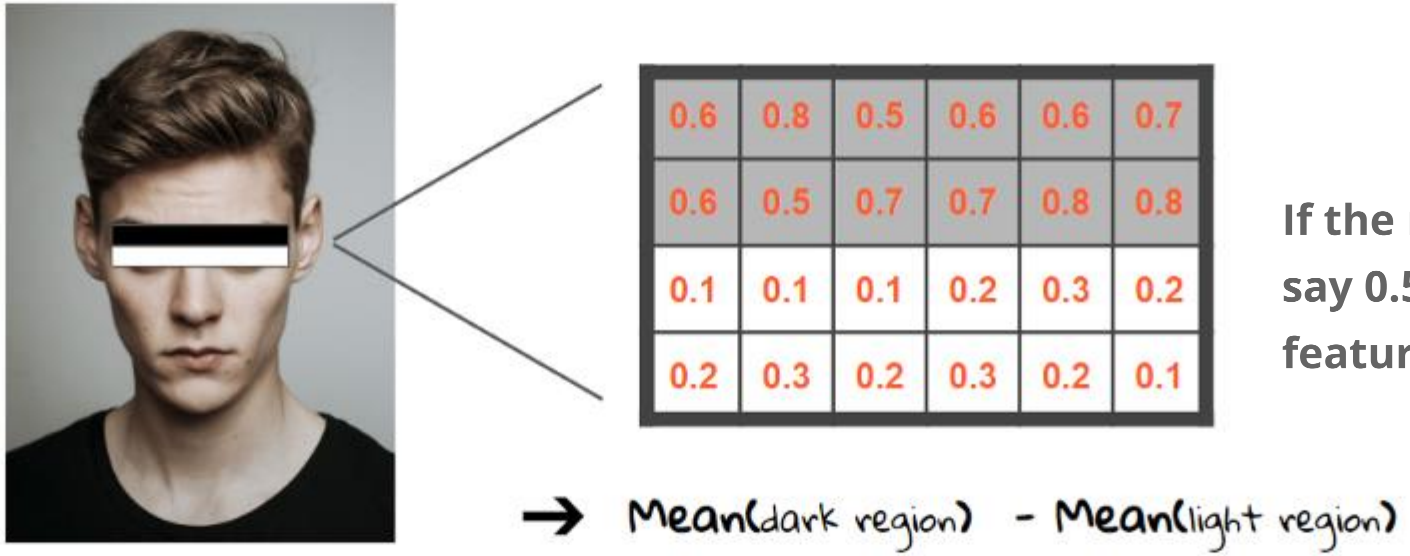
The 4 key points for understanding this algorithm are:

1. **Haar features extraction**
2. **Integral image**
3. **Adaboost**
4. **Cascade classifiers.**

Face Detection: Haar-like features

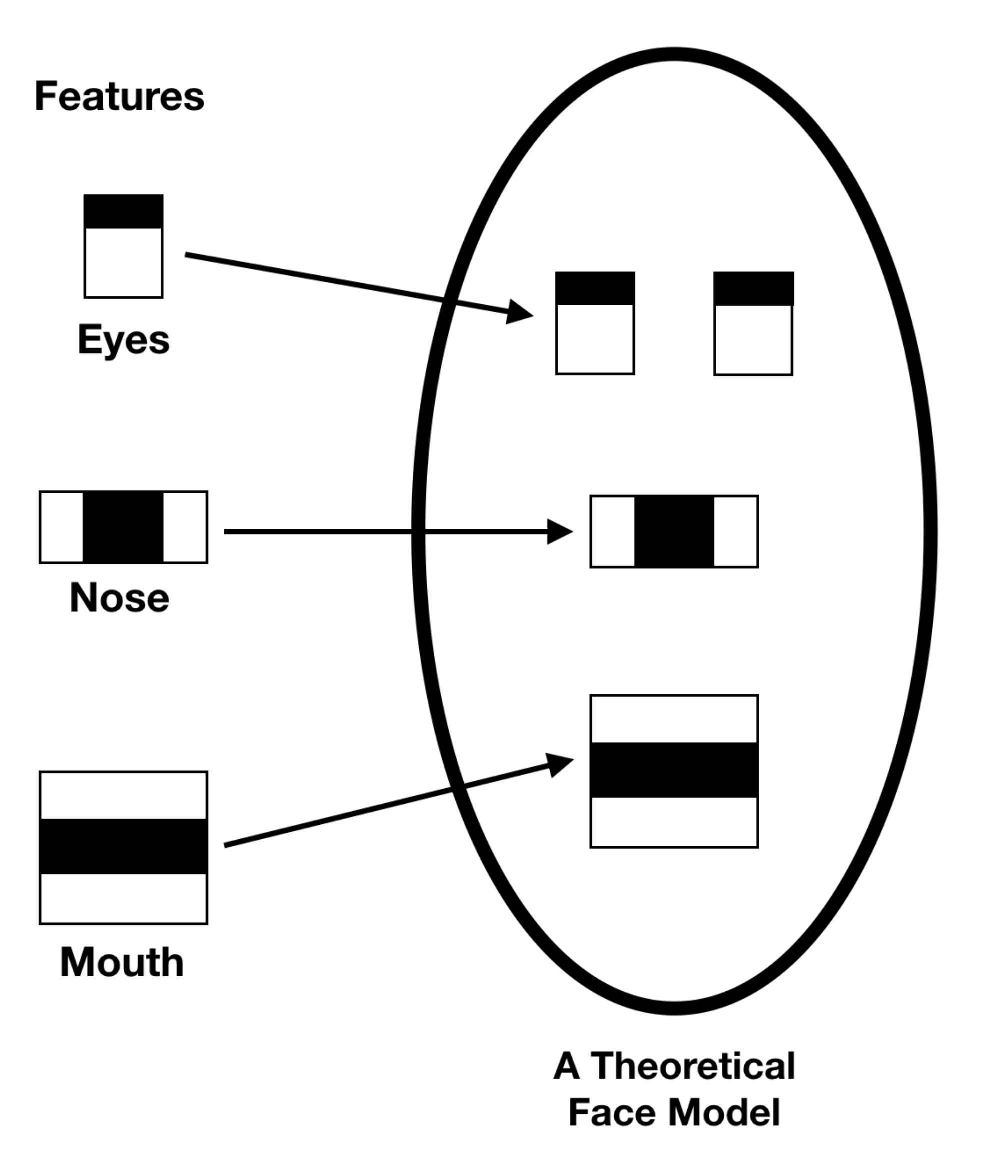
Haar-like features are digital image features used in object recognition. All human faces share some universal properties of the human face like the eyes region is darker than its neighbor pixels, and the nose region is brighter than the eye region.

During detection, we pass the window on an image and do the convolutional operation with the filters to see if there’s the feature we’re looking for is in the image. [Video]



If the result is higher than a threshold, say 0.5, then we conclude there’s the feature we’re detecting.

Face Detection: Haar-like features



Face Detection: Integral image

The integral image is a way of image representation which is derived to make the feature evaluation faster and more effective.

15	12	11	15	13
28	18	13	14	14
11	15	10	20	42
10	30	17	25	33
21	50	20	17	19

Original Image

15	27	38	53	66
43	73	97	126	153
54	99	133	182	251
64	139	377	264	366
85	210	281	372	493

Integral Image

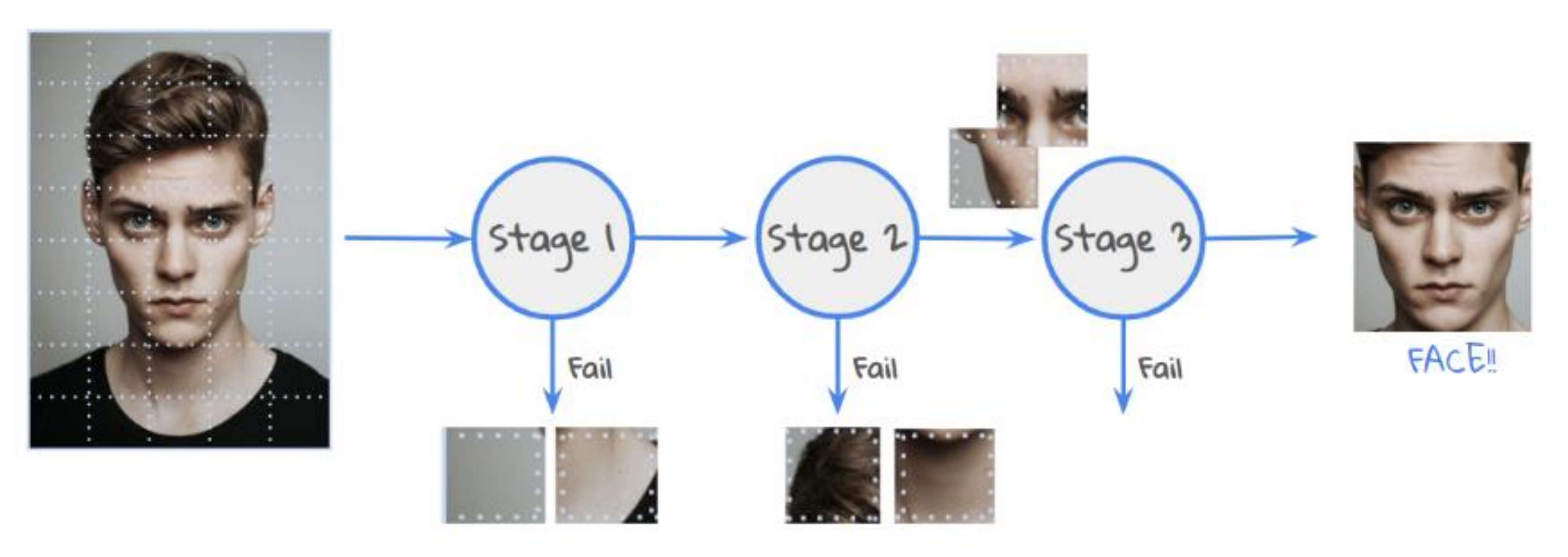
Sum of pixels in orange area

$= I(D) + I(A) - I(B) - I(C)$

With this pre-calculated table, we can simply get the summed value for a certain area by the values of sub-rectangles (the red, orange, blue and purple box).

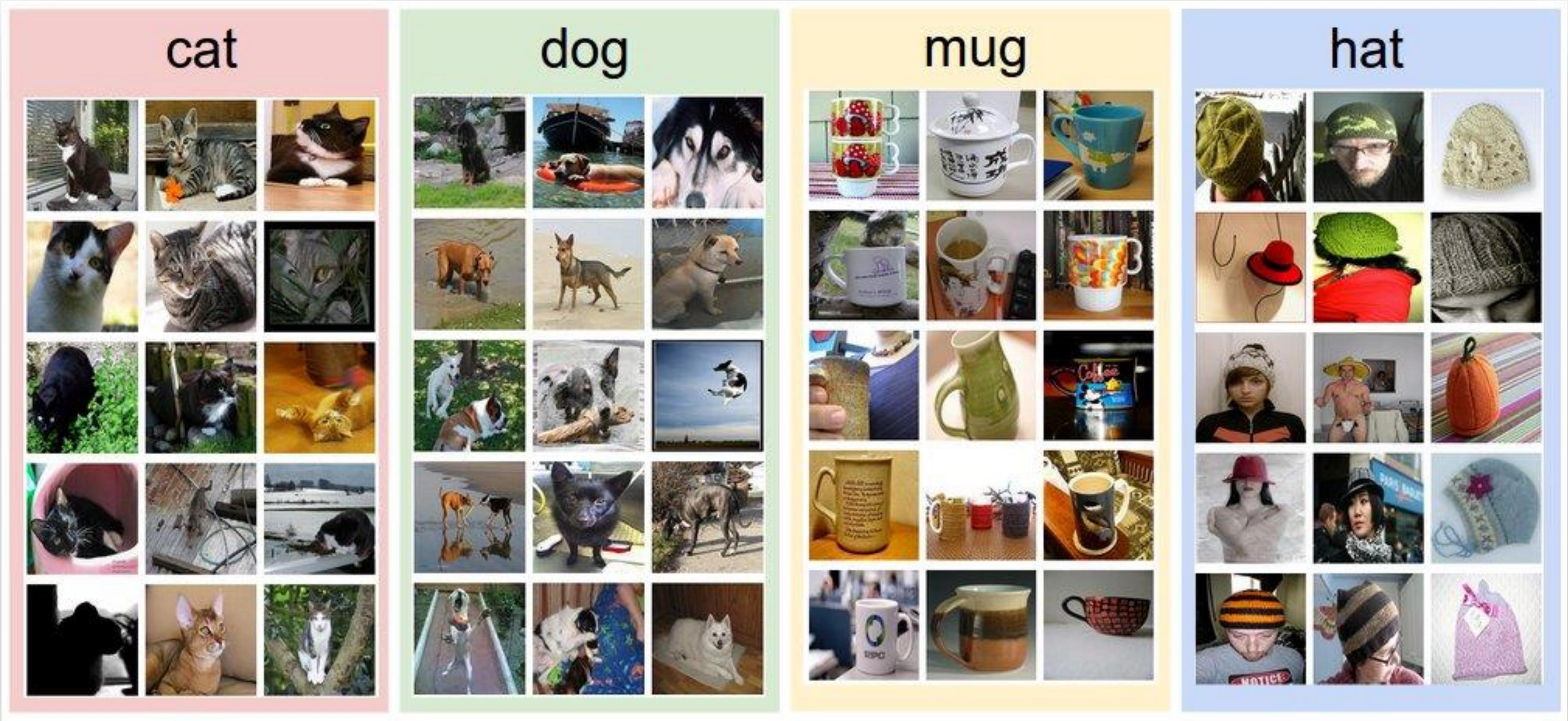
Face Detection: Adaboost

A **cascade classifier** constructs stepwise stages and gives an order among Haar-like features. Basic forms of the features are implemented at the early stages and the more complex ones are applied only for those promising regions. And at each stage, the **Adaboost** model will be trained by **ensembling** weak learners. If a subpart, or a sub-window, is classified as 'not a face-like region' at the prior stage, it's rejected to the next step. By doing so, we can only consider the survived ones and achieve much higher speed.



Machine Learning: Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images



1 From Traditional CV to DL

1.1

From Traditional Convolution Neural Networks to Deep Learning

Soft introduction about OpenCV and what exactly is traditional computer vision.

1.2

Convolution Neural Networks (CNNs)

Motivations behind CNNs, Image Classifications, calculation of sizes of filters, input and output layers.

1.3

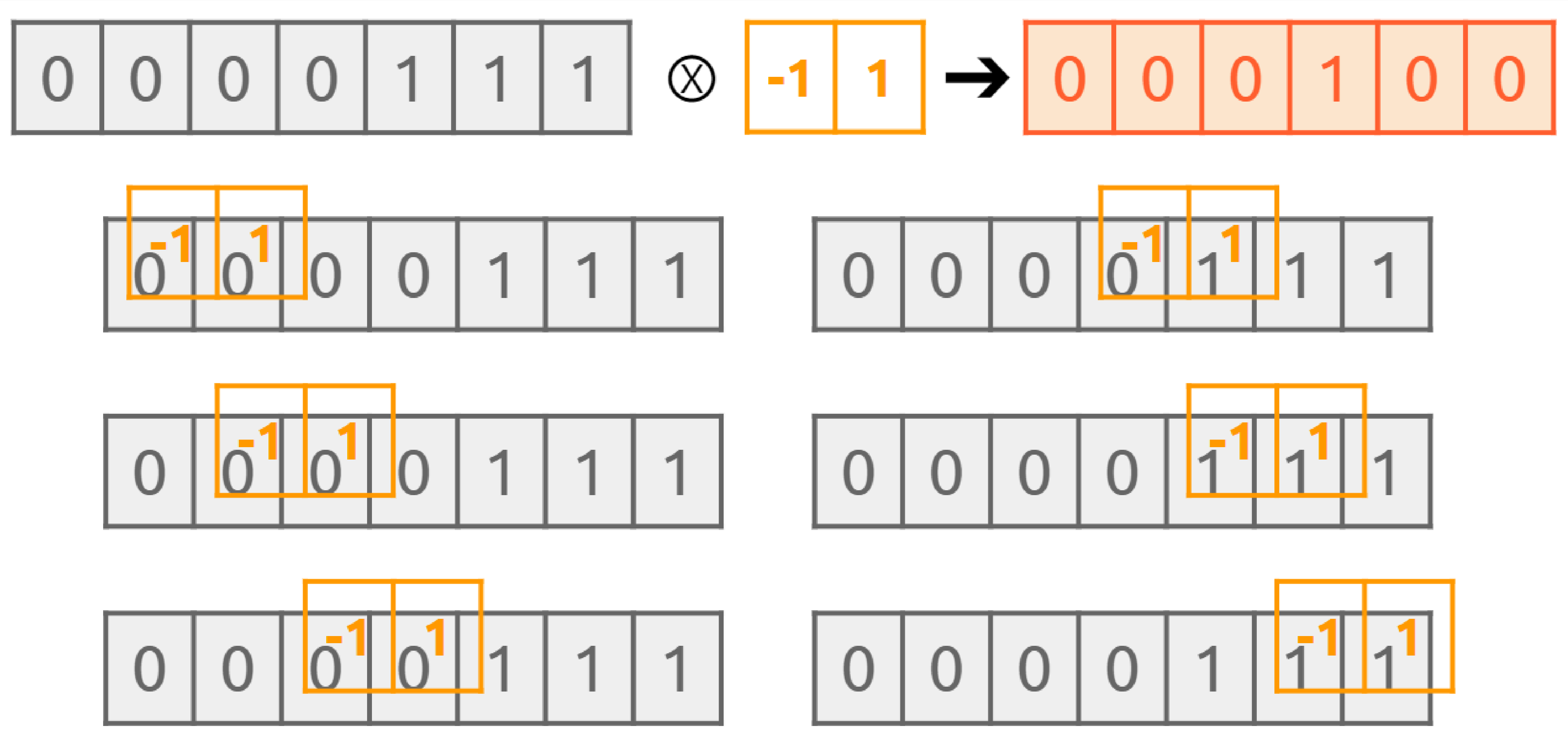
Convolution Neural Network Meta Architectures

How to Design CNNs and the well-known architectures.

Deep Learning: ConvNets

So what is Convolution?

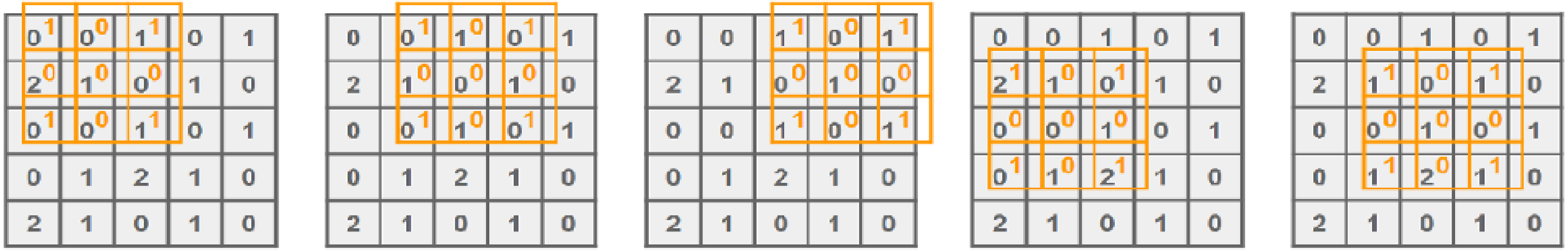
Let's try to understand the concept of convolution with a simple **1-dimensional** case first.



So what is Convolution?

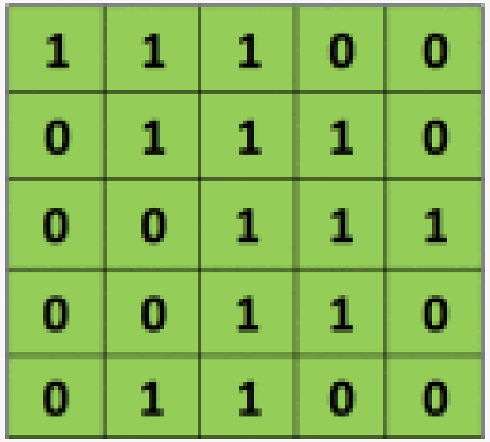
Then let’s try a **2-dimensional** case.

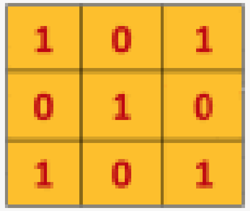
$$\begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 2 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 2 & 1 & 0 \\ 2 & 1 & 0 & 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 2 & 0 & 4 \\ 4 & 4 & 2 \\ 3 & 2 & 2 \end{bmatrix}$$

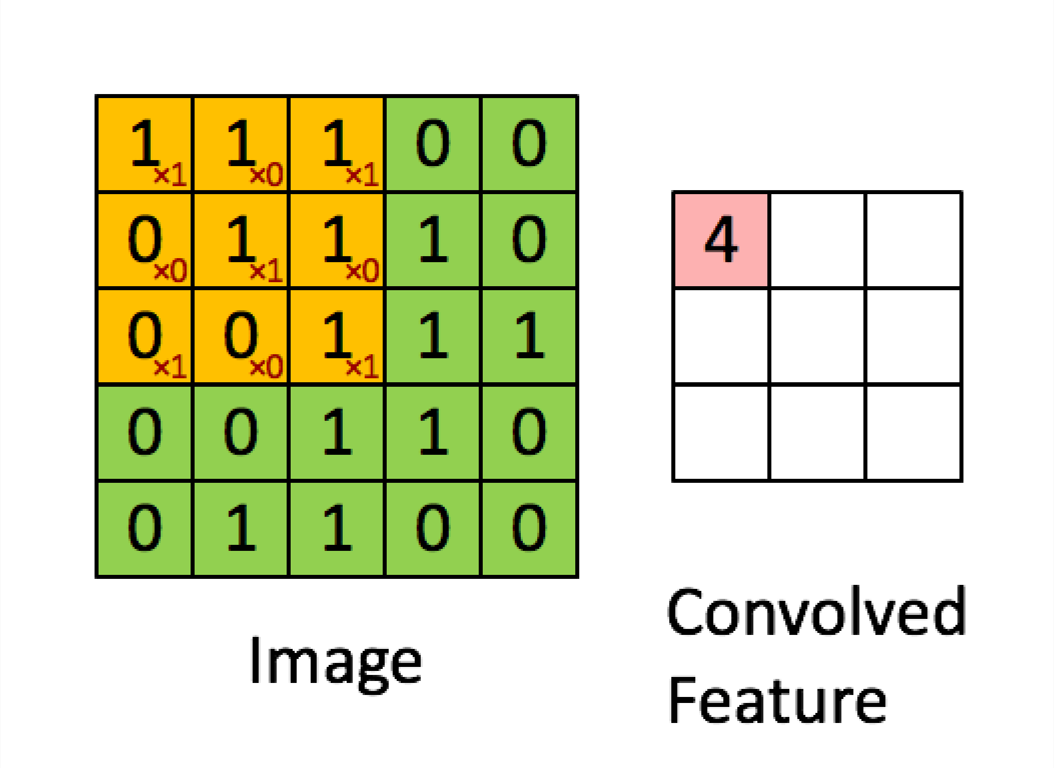


So what is Convolution?

Then let’s try a **2-dimensional** case.







The Convolution Operation - Why?

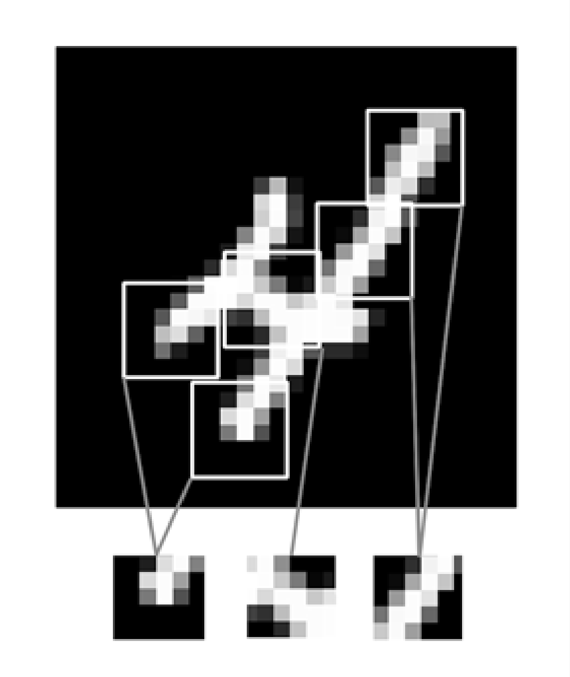
Densely connected layers learn **global** patterns in their input feature space (for example, for a MNIST digit, patterns involving all pixels), whereas **convolution layers** learn **local** patterns.

Key Convolution Properties:

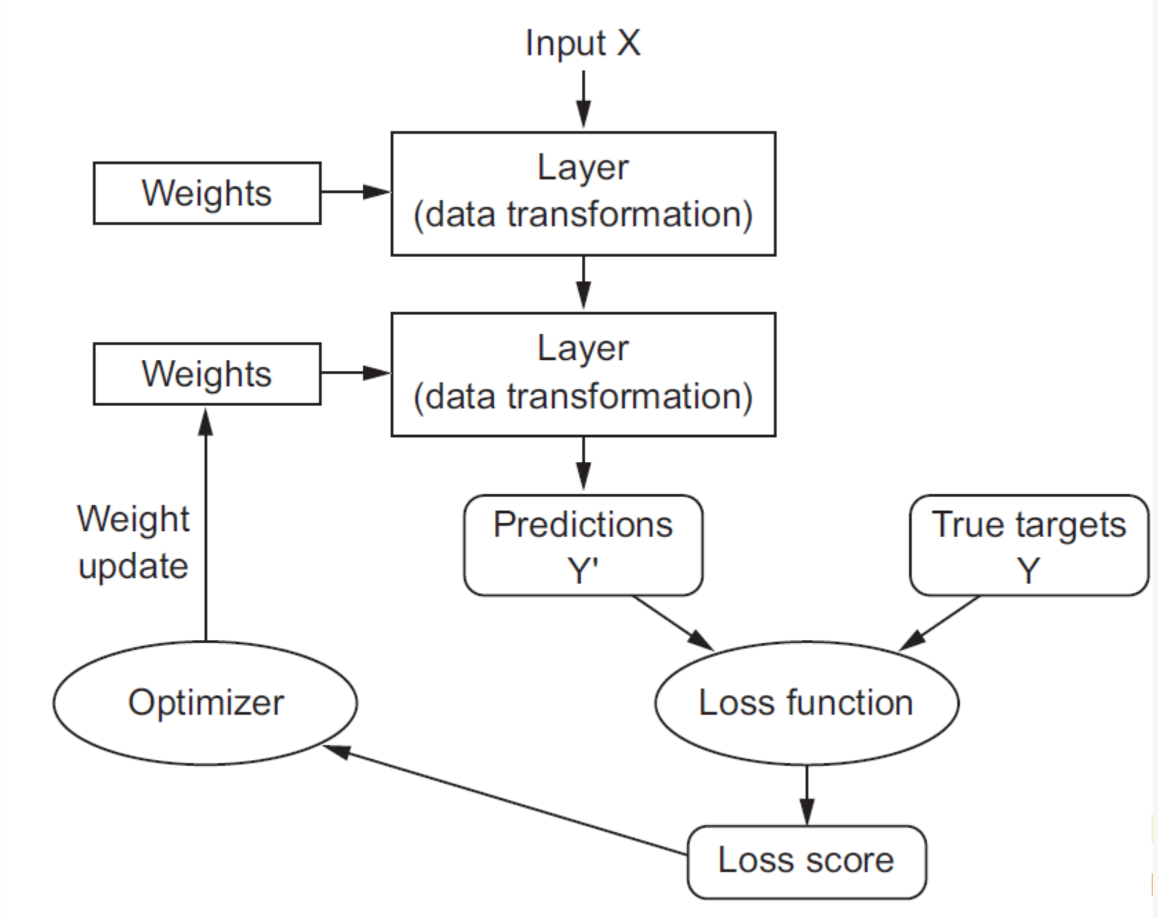
1. The patterns they learn are Translation Invariant

For example, in the upper-left corner. A **densely** connected network would have to learn the pattern **anew** if it appeared at a new location.

After learning a certain pattern in the lower-right corner of a picture, a **convnet** can recognize it **anywhere**: this makes convnets data efficient, they need fewer training samples to learn representations that have generalization power.



A step back - Anatomy of a neural network



- ***Layers***, which are combined into a network (or model)
- The ***input data*** and corresponding targets
- The ***loss function***, which defines the feedback signal used for learning
- The ***optimizer***, which determines how learning proceeds

Developing with Keras: a quick overview

Keras workflow:

1. Define your training data: input tensors and target tensors.
2. Define a network of layers (or model) that maps your inputs to your targets.
3. Configure the learning process by choosing a loss function, an optimizer, and some metrics to monitor.
4. Iterate on your training data by calling the fit() method of your model.

There are two ways to define a model:

1. Using the **Sequential** class (only for linear stacks of layers, which is the most common network architecture by far).

2. **Functional API** (for directed acyclic graphs of layers, which lets you build completely arbitrary architectures).

Developing with Keras: a quick overview

```python
# Sequential class
model = models.Sequential()
model.add(layers.Dense(32, activation='relu', input_shape=(784,)))
model.add(layers.Dense(10, activation='softmax'))
```

```python
# Functional API
input_tensor = layers.Input(shape=(784,))
x = layers.Dense(32, activation='relu')(input_tensor)
output_tensor = layers.Dense(10, activation='softmax')(x)

model = models.Model(inputs=input_tensor, outputs=output_tensor)
```

Developing with Keras: a quick overview

The learning process is configured in the compilation step, where you specify the **optimizer** and **loss function**(s) that the **model** should use, as well as the **metrics** you want to monitor during training.

```
# Optimizer and loss
from keras import optimizers
model.compile(optimizer=optimizers.RMSprop(lr=0.001), loss='mse', metrics
['accuracy'])
```

Developing with Keras: a quick overview

Finally, the learning process consists of passing Numpy arrays of input data (and the corresponding target data) to the model via the **fit()** method

```
# Fit
model.fit(input_tensor, target_tensor, batch_size=128, epochs=10)
```

Thank You!