

Deep Learning for Computer Vision

Ahmed Hosny Abdel-Gawad

Senior AI/CV Engineer



Table Content

Part One

From Traditional Computer Vision to Deep Learning

Part Two

Practical Convolution Neural Nets

Part Three

Computer Vision Applications

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)

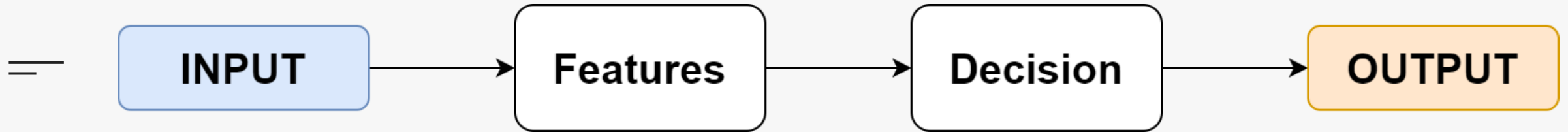
Motivations behind CNNs, Image Classifications, calculation of sizes of filters, input and output layers.

1.3

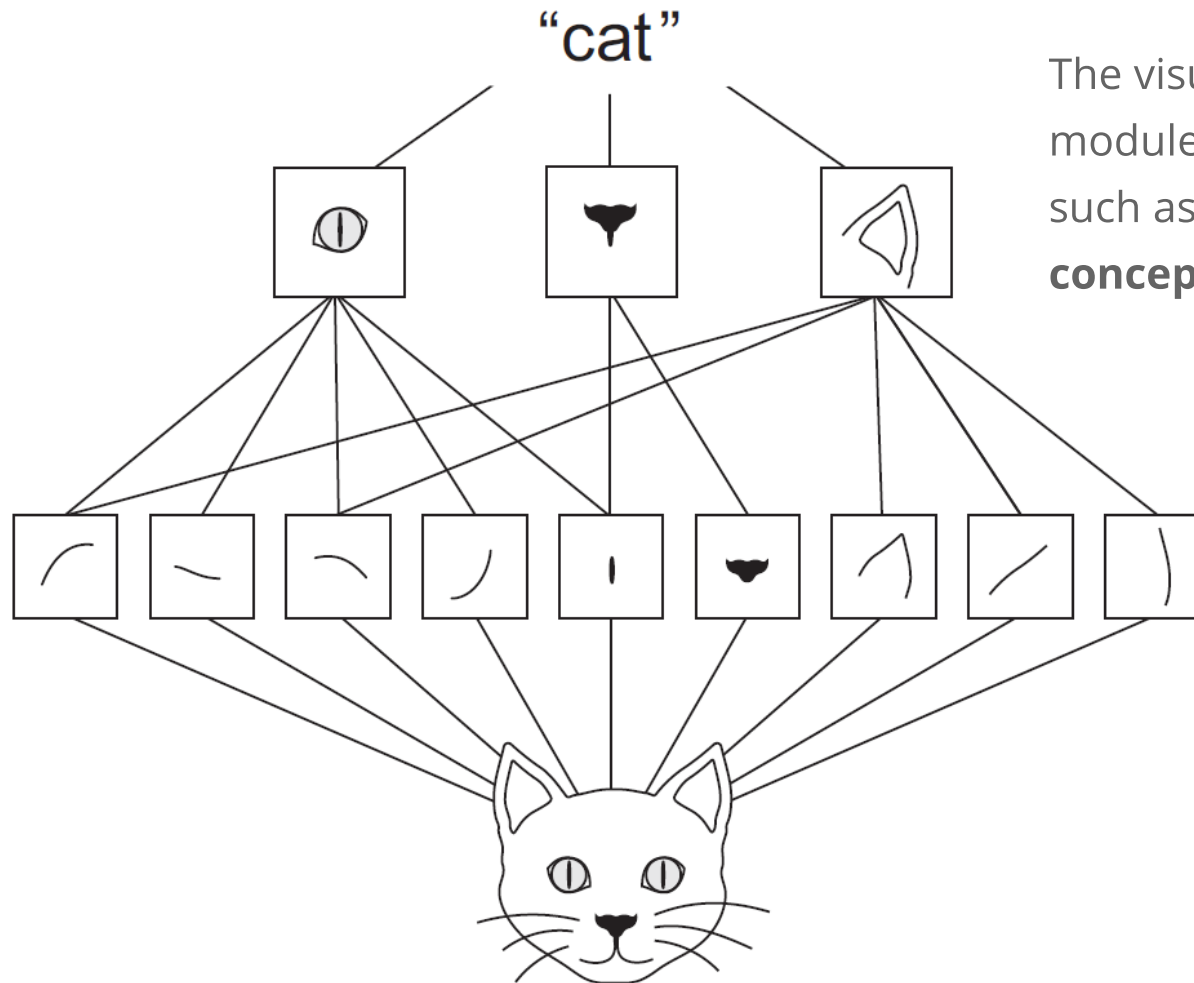
Convolution Neural Network Meta Architectures

How to Design CNNs and the well-known architectures.

Global Framework

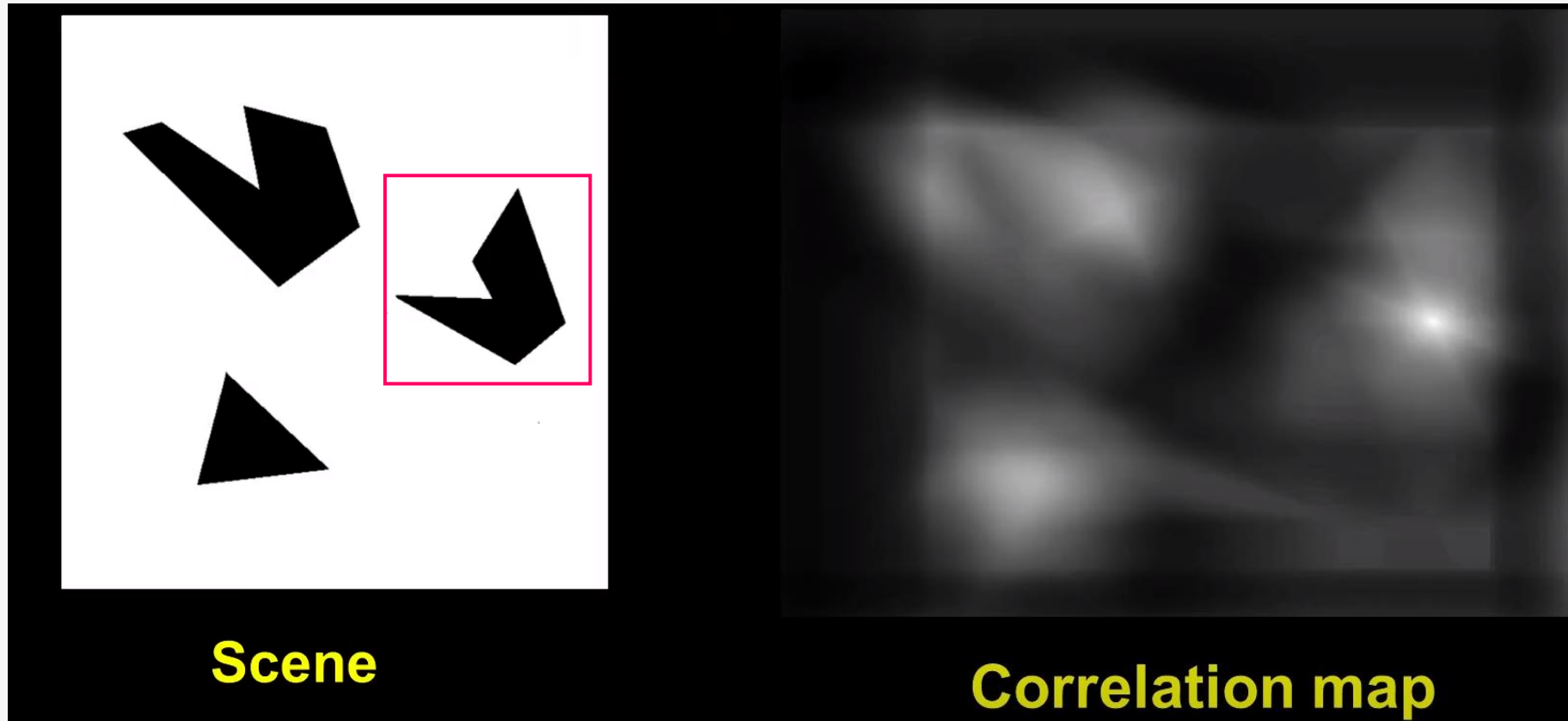


What could be **good features**?



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?

=

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Dot Product

1	0	1
0	1	0
1	0	1

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved
Feature

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!

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.

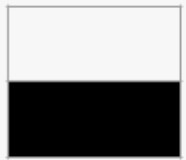
Given an image, the algorithm looks at many smaller subregions and tries to find a face by looking for specific features in each subregion. It needs to check many different positions and scales because an image can contain many faces of various sizes.

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. There are 3 types of Haar-like features that Viola and Jones identified in their research:



1. Edge Features

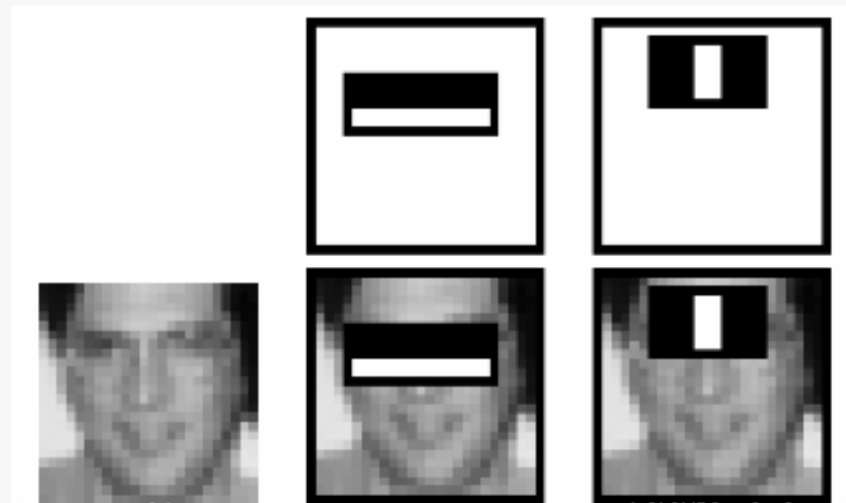


2. Line Features

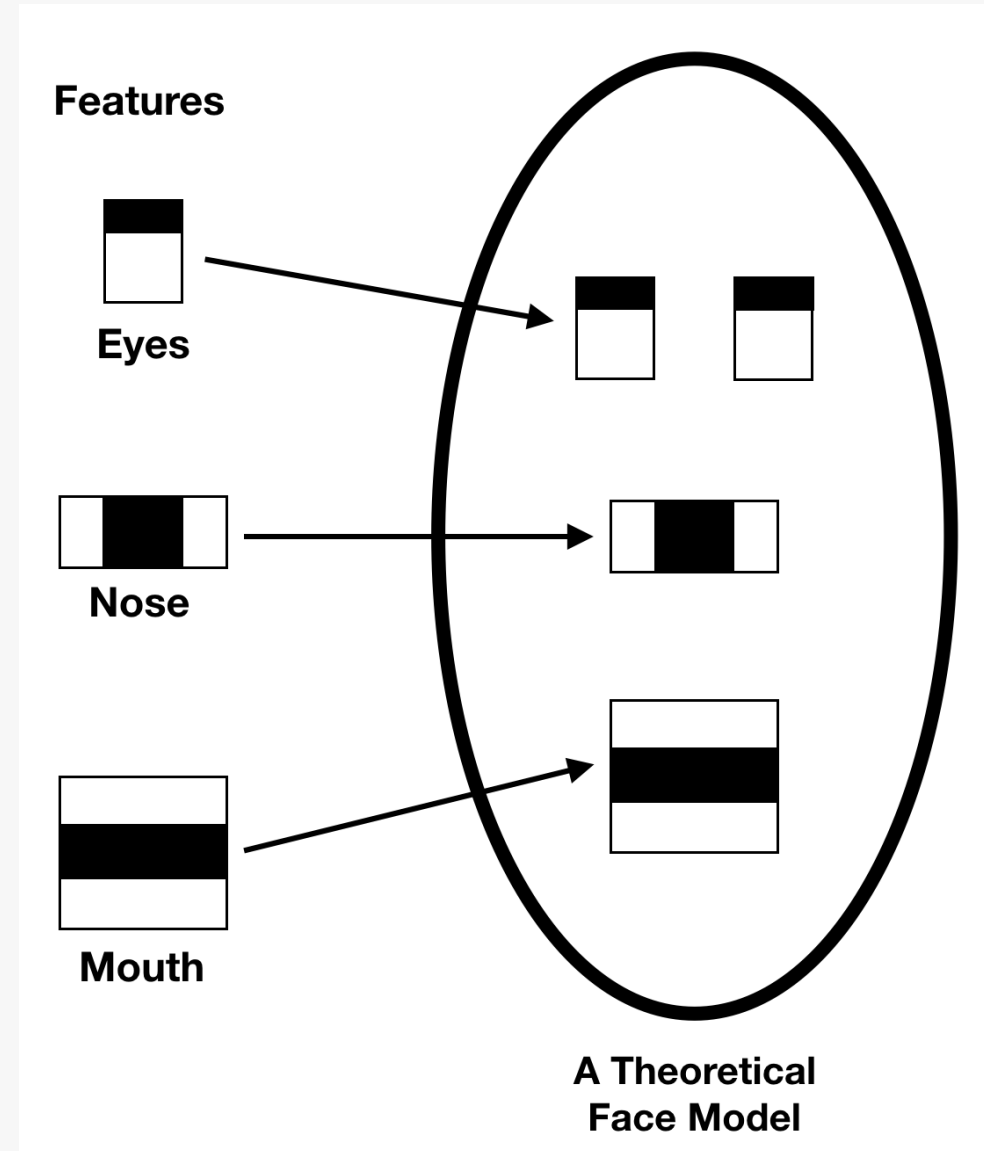


3. Four rectangle Features

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.



Face Detection: Haar-like features



Face Detection: Haar-like features

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](#)



0.6	0.8	0.5	0.6	0.6	0.7
0.6	0.5	0.7	0.7	0.8	0.8
0.1	0.1	0.1	0.2	0.3	0.2
0.2	0.3	0.2	0.3	0.2	0.1

→ $\text{Mean}(\text{dark region}) - \text{Mean}(\text{light region})$

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

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.



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**

The number of features that are present in the 24×24 detector window is nearly 160,000, but only a few of these features are important to identify a face. So we use the **AdaBoost** algorithm to identify the best features in the 160,000 features.

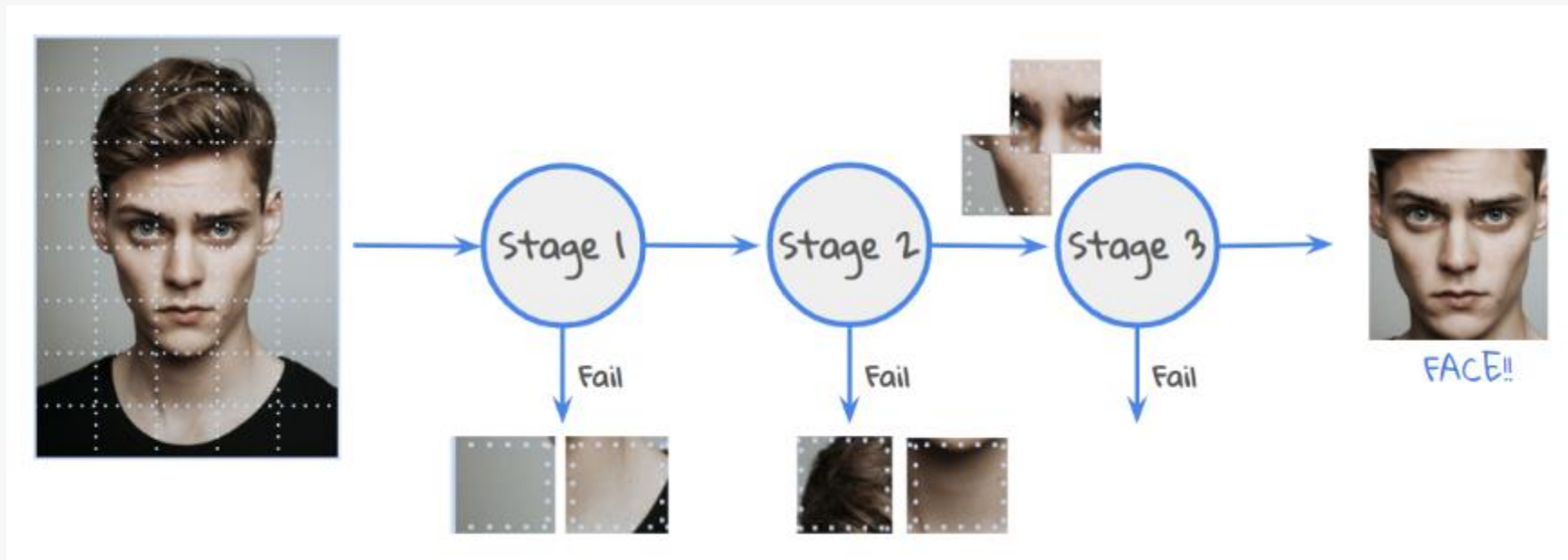
In the Viola-Jones algorithm, each Haar-like feature represents a weak learner. To decide the type and size of a feature that goes into the final classifier, AdaBoost checks the performance of all classifiers that you supply to it.

To calculate the performance of a classifier, you evaluate it on all subregions of all the images used for training. Some subregions will produce a strong response in the classifier. Those will be classified as positives, meaning the classifier thinks it contains a human face. Subregions that don't provide a strong response don't contain a human face, in the classifiers opinion. They will be classified as negatives.

The classifiers that performed well are given higher importance or weight. The final result is a strong classifier, also called a boosted classifier, that contains the best performing weak classifiers.

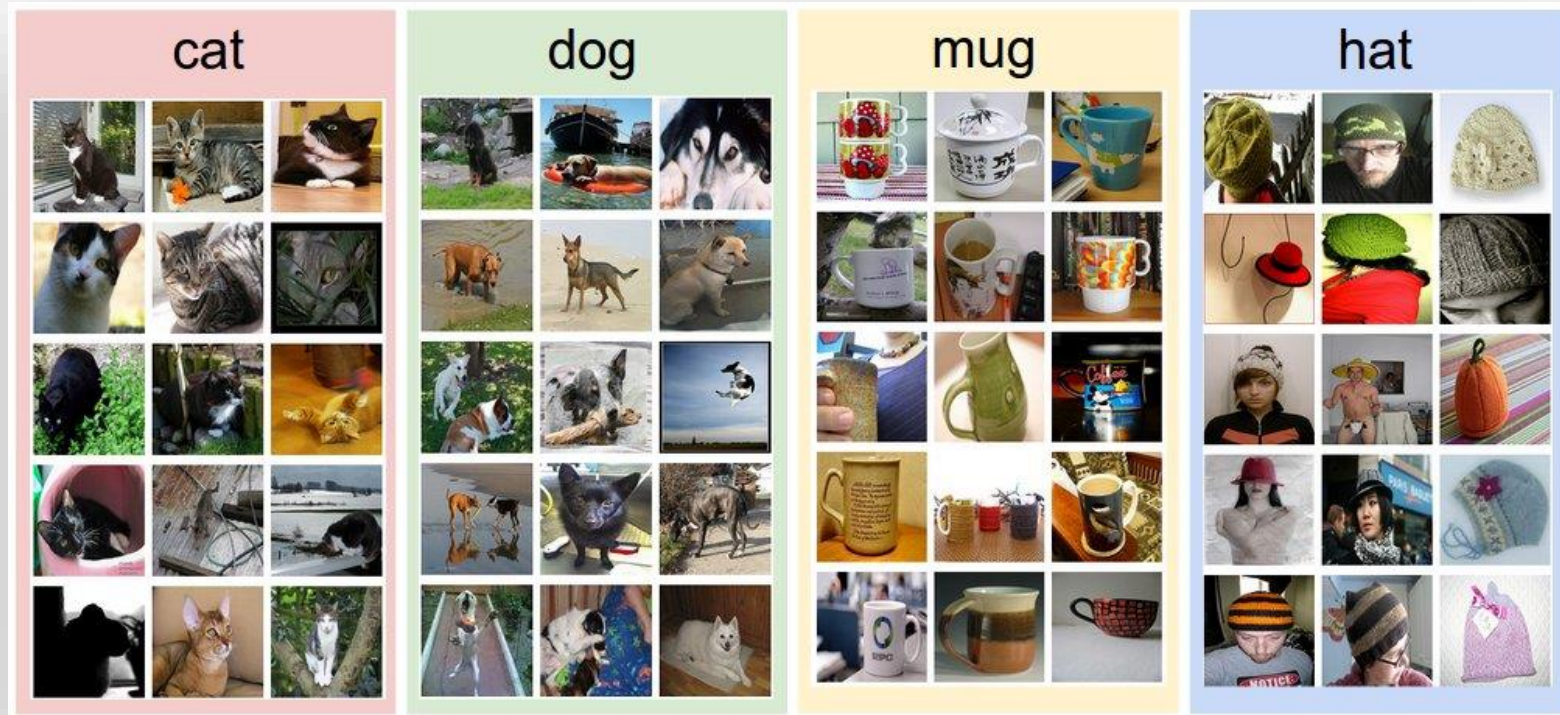
Face Detection: Cascading Classifiers

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.



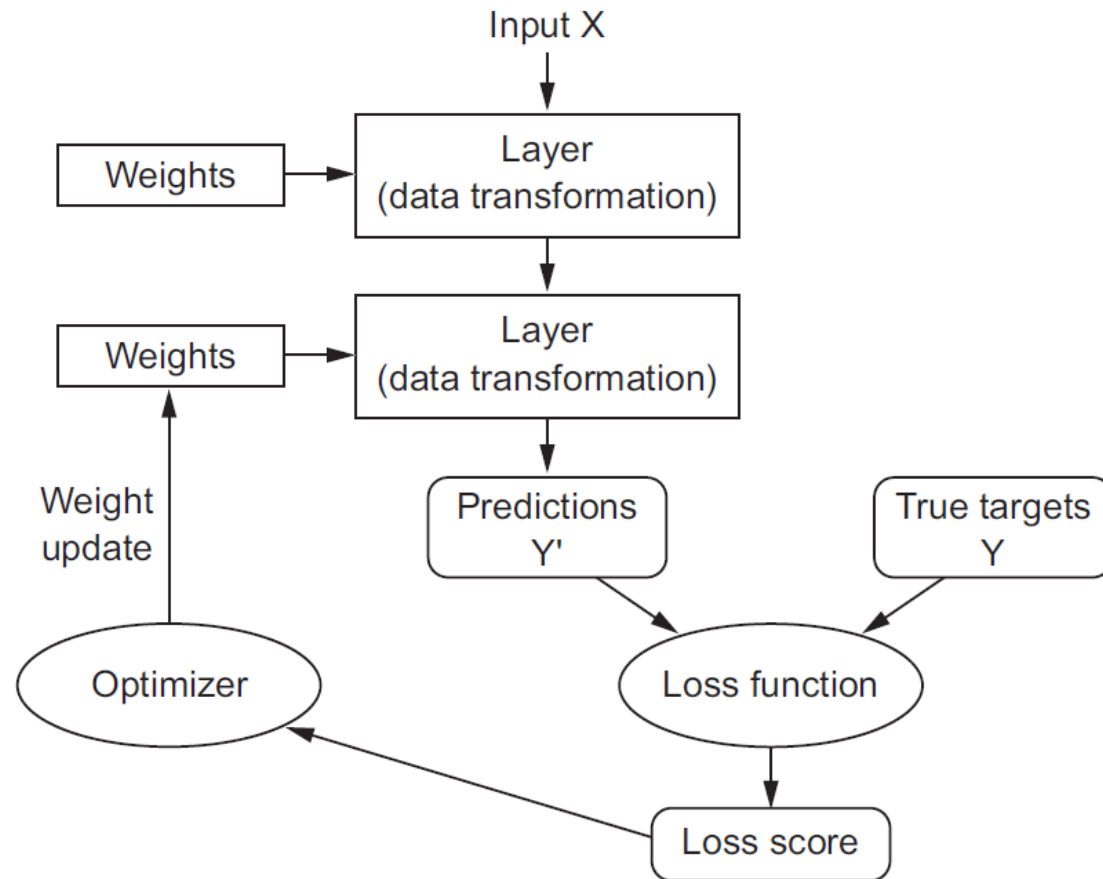
Machine Learning: **Data-Driven** Approach

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



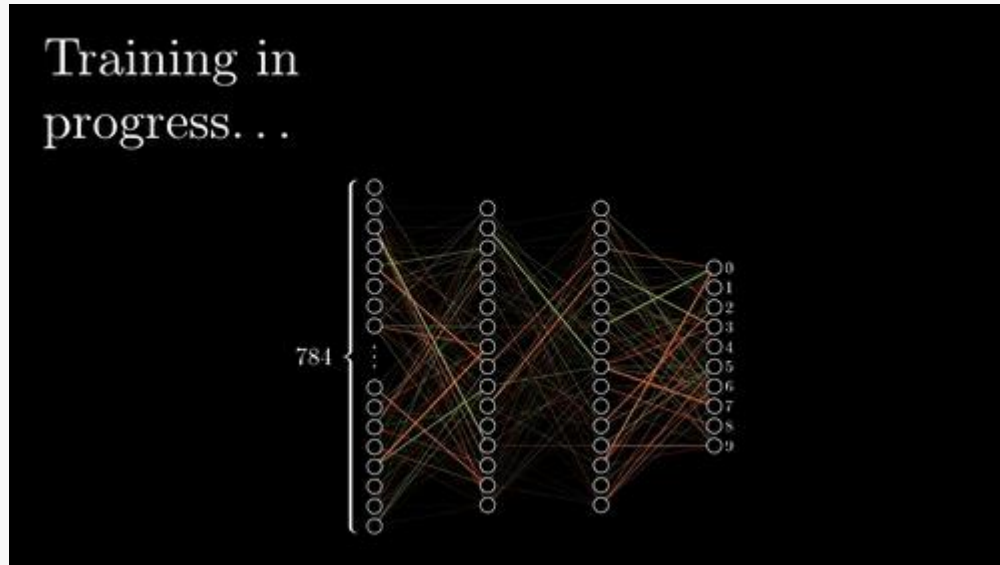
Deep Learning: FCNN

Anatomy of a **neural network**



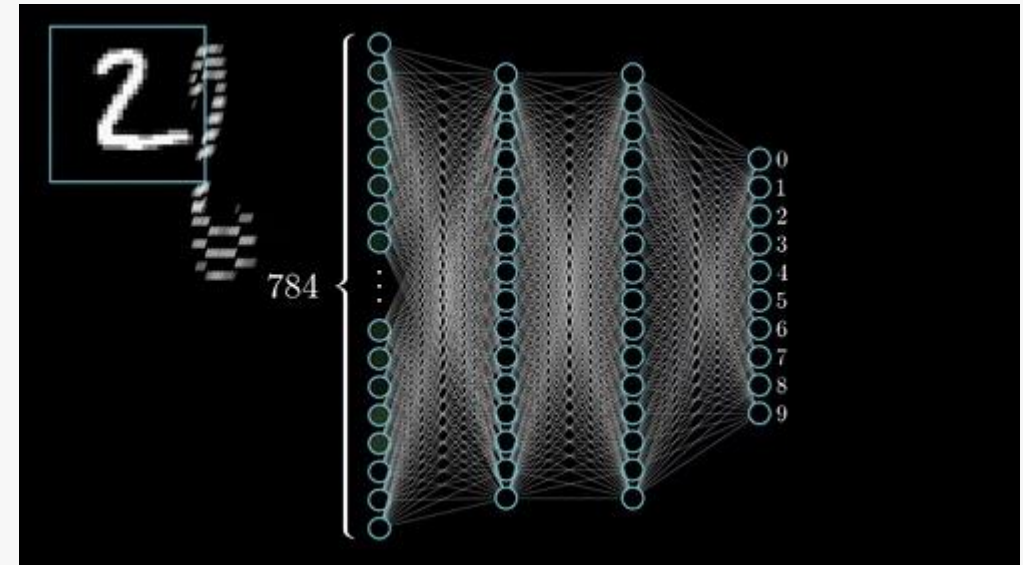
- ❑ **Layers**, which are combined into a network (or model)
- ❑ The **input data** and corresponding targets
- ❑ The **loss function**, which defines the feedback signal used for learning
- ❑ The **optimizer**, which determines how learning proceeds

Fully Connected Neural Networks



Input: 2D Image

- Vector of pixel values



Fully Connected: Connect neurons in hidden layer to all neurons in input

- No spatial information!
- And many, many parameters.

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



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




```
# 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!