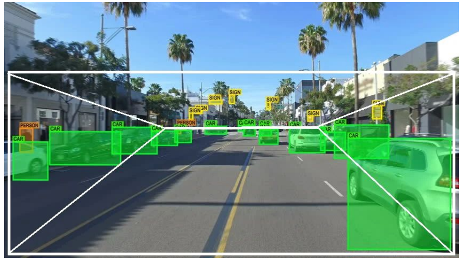
CNN Crash Course

Convolutional Neural Networks (CNNs)

CNNs are **machine learning models** designed to work better for visual data inputs

- **Input**: an image (or more generally a **tensor**)
- **Output**: some prediction related to the input image

Use case: Real or fake face?



Use case: Find cars and pedestrians

Intuition

We can think of the following decomposition relationships:

- **Image** is a collection of **objects**
- **Object** is a grouping of **shapes**
- **Shape** is a set of **lines and colors**

What if we *learn* the reverse relationship? Figure out how

- **Lines and color** combine to construct a **specific shape**?
- **Shapes** combine to construct a **specific object**?
- **Objects** combine to construct a **specific image?**

Intuition

What if we *learn* the reverse relationship? Figure out how

- **Lines and color** combine to construct a **specific shape**?
- **Shapes** combine to construct a **specific object**?
- **Objects** combine to construct a **specific image?**

If we could learn this relationship, we know how to **identify** an object (or groups of objects) in an image!

Q: How do we do this *algorithmically*?

Intuition

What if we *learn* the reverse relationship? Figure out how

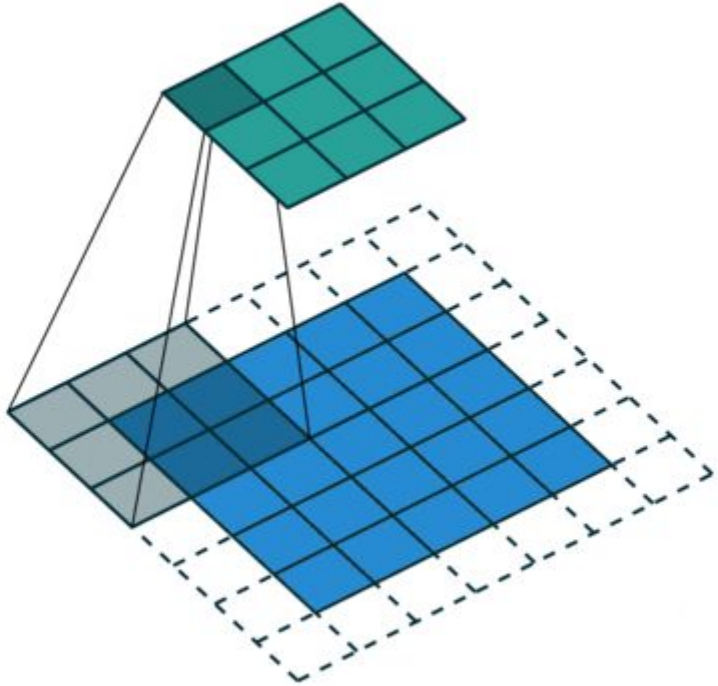
- **Lines and color** combine to construct a **specific shape**?
- **Shapes** combine to construct a **specific object**?
- **Objects** combine to construct a **specific image?**

Q: How do we do this *algorithmically*?

A: Need to solve two subproblems:

- How do we find a *pattern* (ex: a line) in an image?
- What patterns do we *want* to find to identify objects in an image?

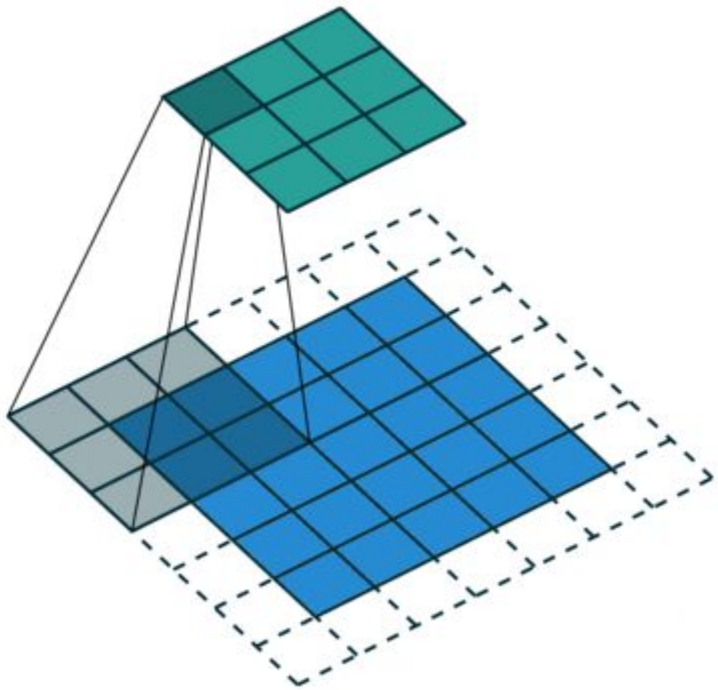
Q: How to find a pattern in an image?
A: ***Convolution***



Convolution

Operation to slide a **filter** over an **input** to record *where* filter pattern exists in input image to **output**

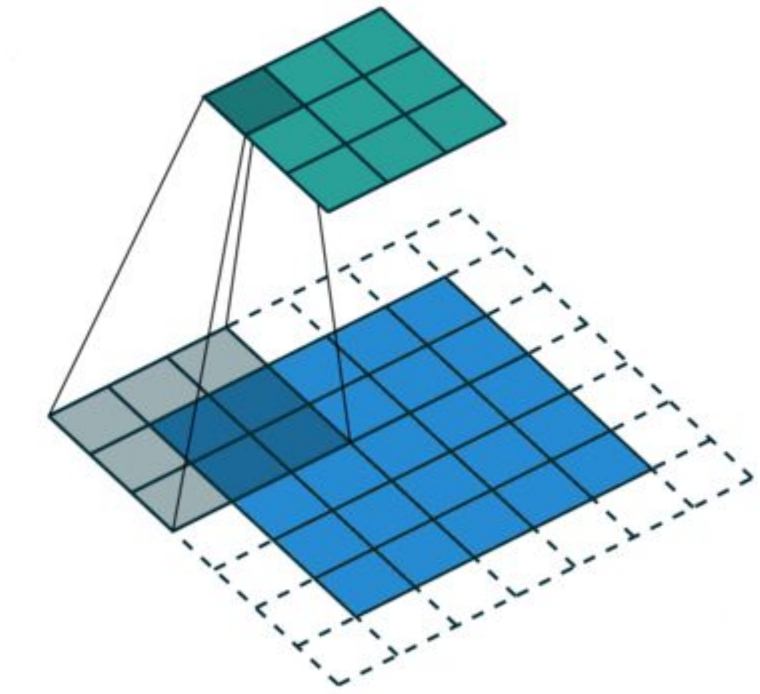
- The **blue square** is the **input**
- The **gray square** is a **filter**
    - the *pattern* that we want to find in the input (e.g. a line)
- The **green square** is the **output**
    - tracks *where* the pattern was found in the input



Convolution

- The **blue square** is the **input**
- The **gray square** is a **filter**
- The **green square** is the **output**

Example: if the top-right cell of output (green square) has a large value, the filter pattern likely exists in the top right window of the image



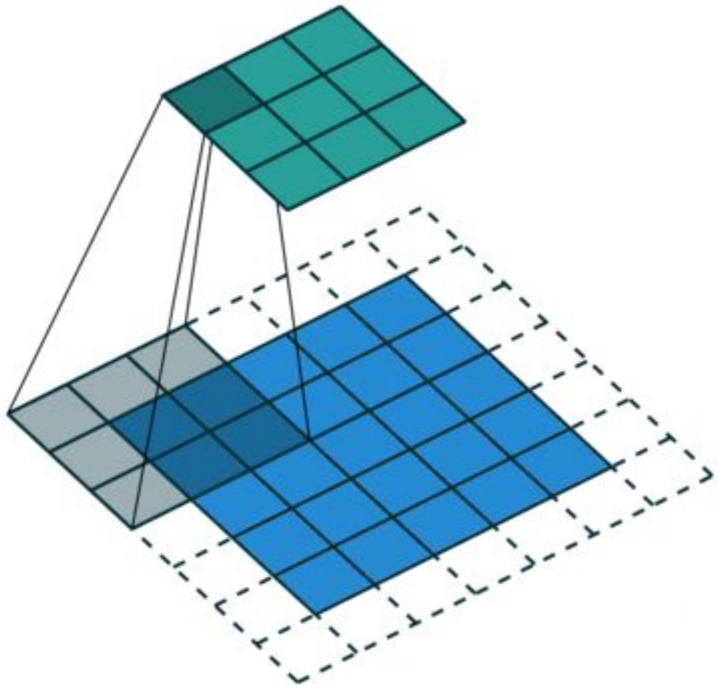
Convolution

- The **blue square** is the **input**
- The **gray square** is a **filter**
- The **green square** is the **output**

Convolution Hyperparameters:

- **Filter size** - pattern resolution
- **Stride** - how far we jump between windows
- **Padding** - extra space around edges

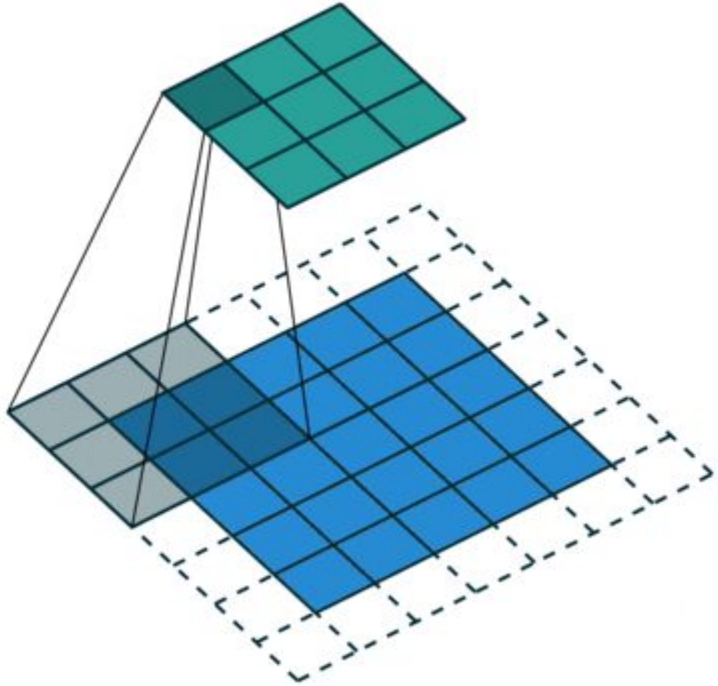
We control these as ML Architects!



Convolution

- The **blue square** is the **input**
- The **gray square** is a **filter**
- The **green square** is the **output**

TLDR - Convolution allows us to search for patterns in the input image (e.g. lines, nose, circles) and record where we found them



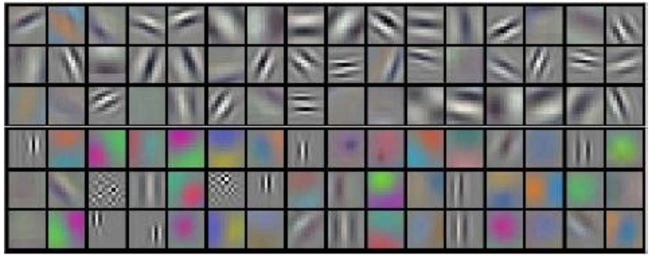
Convolution

Crazy Idea - Why not apply the convolution operation to the output of *a previous* convolution operation?

What is the intuition?

- **Convolution 1:** convolve over **input image** to record instances of **basic lines/colors**
- **Convolution 2:** convolve over details of **basic lines/colors** to record instances **shapes**
- **...**
- **Convolution *n*:** convolve over details of **shapes** to record instances of **objects**

We now know exactly where objects are in an image!



Actual **basic line/color** filters from AlexNet

Convolution

Crazy Idea - Why not apply the convolution operation to the output of *a previous* convolution operation?

What is the intuition?

- **Convolution 1:** convolve over **input image** to record instan
- **Convolution** lines/colors
- ...
- **Convolution** to record instances of **objects**

By applying repeated convolution, we can learn to ***recognize*** objects and complex patterns in images!

ilters from AlexNet

We now know exactly where objects are in an image!

Intuition

What if we *learn* the reverse relationship? Figure out how

- **Lines and color** combine to construct a **specific shape**?
- **Shapes** combine to construct a **specific object**?
- **Objects** combine to construct a **specific image?**

Q: How do we do this *algorithmically*?

A: Need to solve two subproblems:

- ~~How do we find a *pattern* (ex: a line) in an image?~~
- What patterns do we *want* to find to identify objects in an image?

Convolution

Q: What patterns do we *want* to find to identify objects in an image?

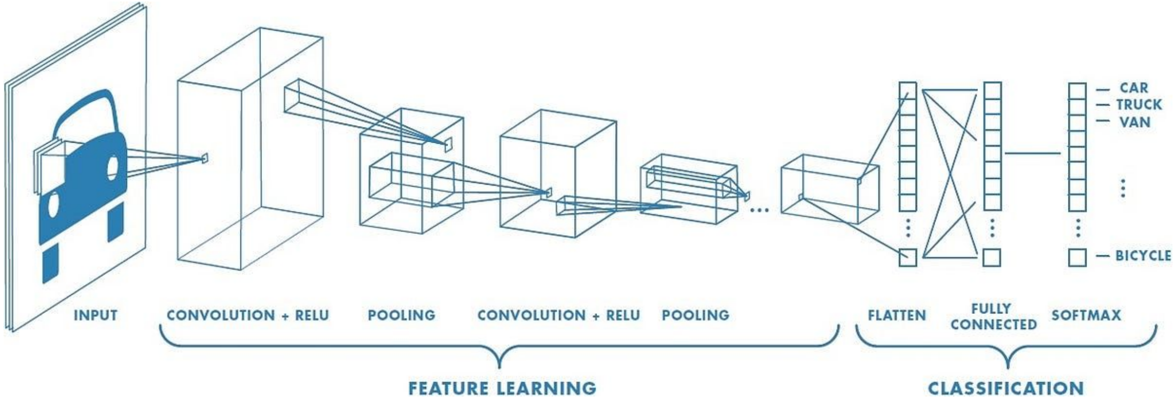
A: Use ✨ **machine learning** ✨ !

- Collect images with corresponding labels
- Tune each filter used in convolution to find the filters that allow us to predict labels the best!
    - Values in filters are parameters of the machine learning model

Convolution is the *engine* powering modern computer vision.

Convolutional Neural Networks

Formalize the application of convolution via a **convolutional neural network**.



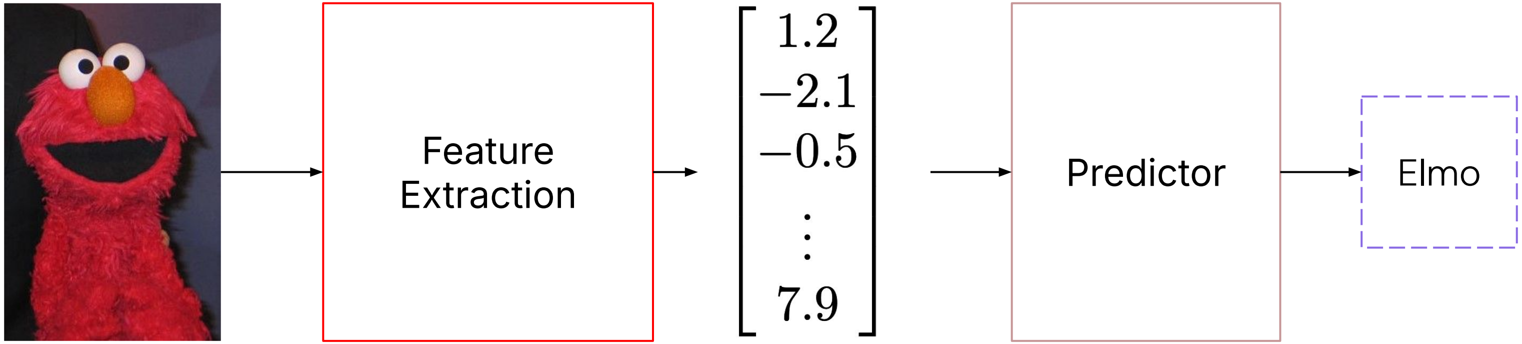
Input: Image

Output: Predicted Class

Convolutional Neural Networks

A CNN can be thought as having two parts:

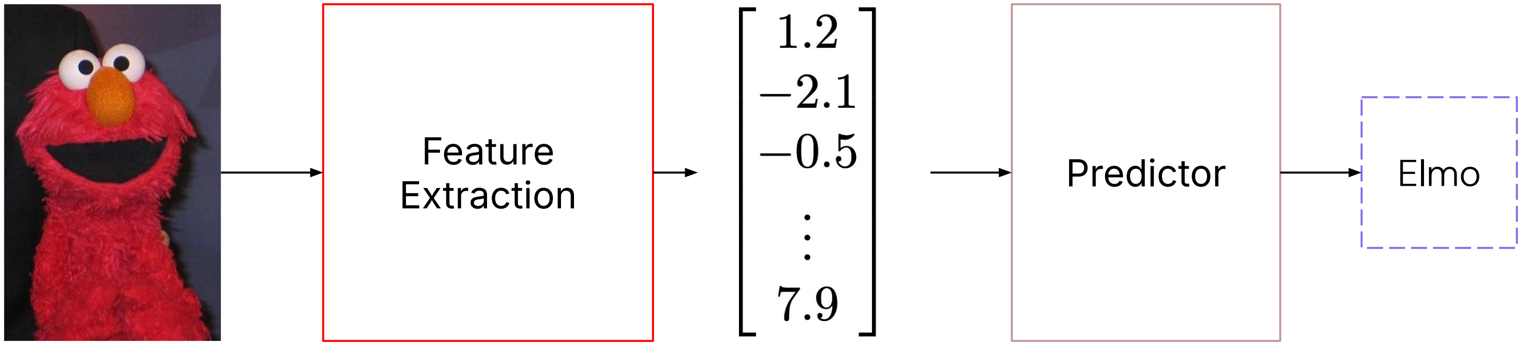
- **Feature Extraction** - convert image to a vector
- **Predictor** - convert a vector to a predicted class



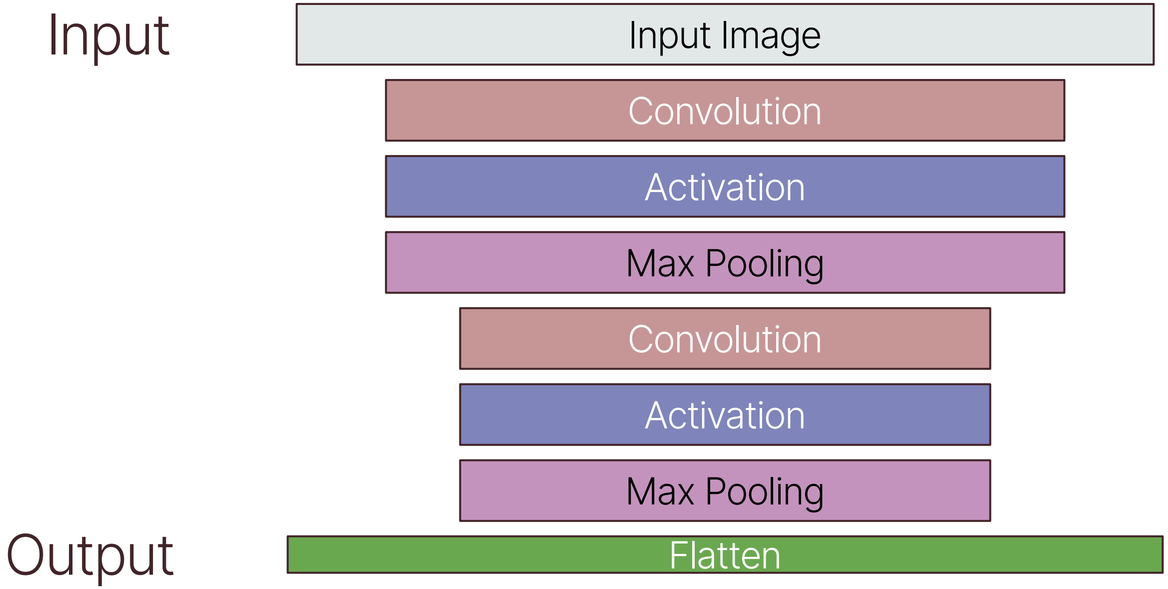
Convolutional Neural Networks

A CNN can be thought as having two parts:

- **Feature Extraction** - convert image to a vector
- **Predictor** - convert a vector to a predicted class



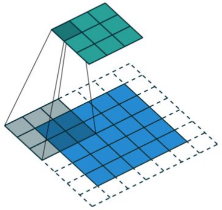
Feature Extraction - Example



Feature Extraction

Contains **convolutional layers**:

- **Input:** Tensor from previous layer ***(D x H x W)***
- **Process**: Convolutional layers include ***k*** filters, and convolution is performed for each of those ***k*** filters!
- **Output:** Output tensor from convolving each filter ***(k x H' x W')*** - there are ***k*** green squares!

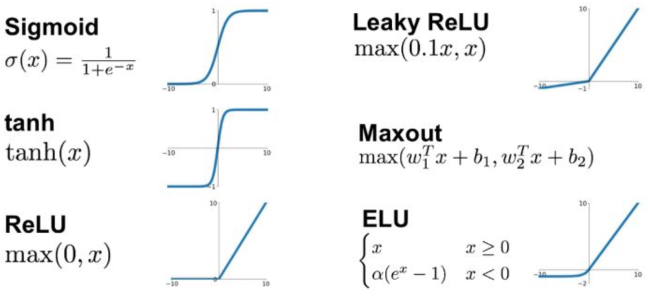


Feature Extraction

Contains **activation layers**:

- **Input:** Tensor from previous layer ***(D x H x W)***
- **Process**: Apply a ✨ nifty ✨ function to each number in tensor
- **Output:** Output tensor after activation ***(D x H x W)***

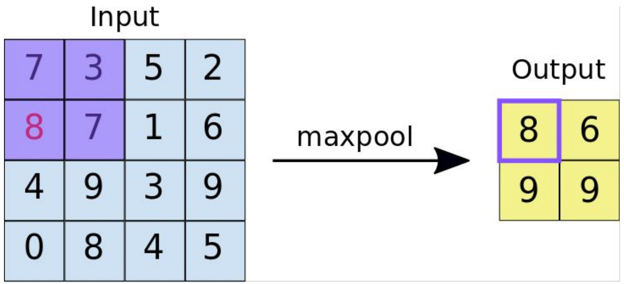
Examples of activation functions



Feature Extraction

Contains **pooling layers**:

- **Input:** Tensor from previous layer ***(D x H x W)***
- **Process**: Makes the height and width smaller to reduce info. Often done by looking at max of each window of input
- **Output:** Output tensor after pooling ***(D x H' x W')***

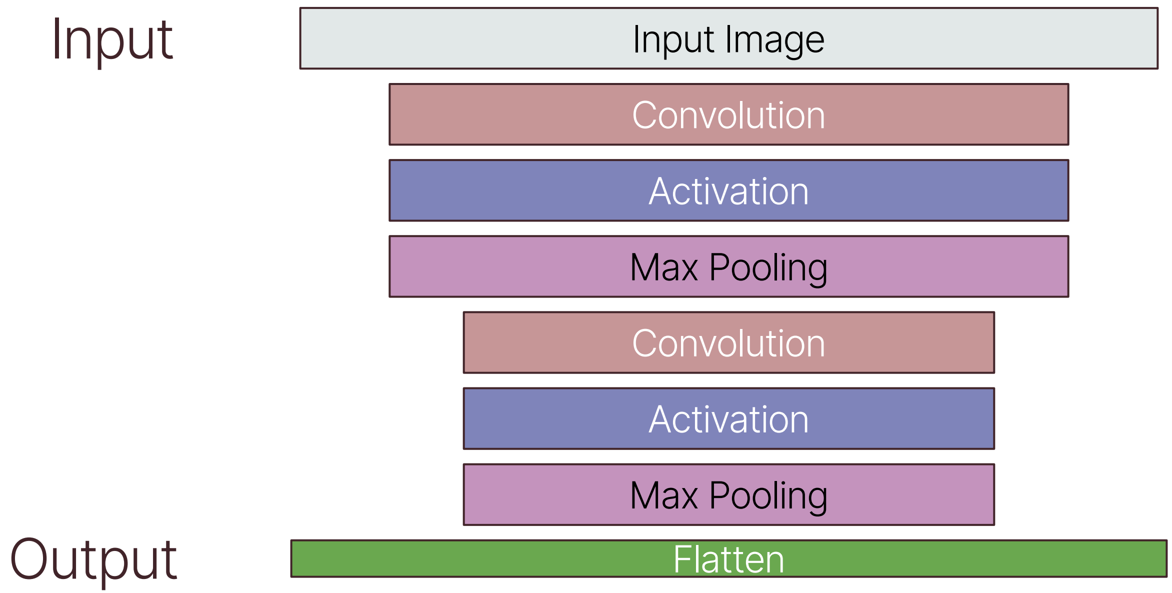


Feature Extraction

Contains **flatten layers**:

- **Input:** Tensor from previous layer ***(D x H x W)***
- **Process**: Smush into a single dimension vector
- **Output:** Output vector ***(DHW)***

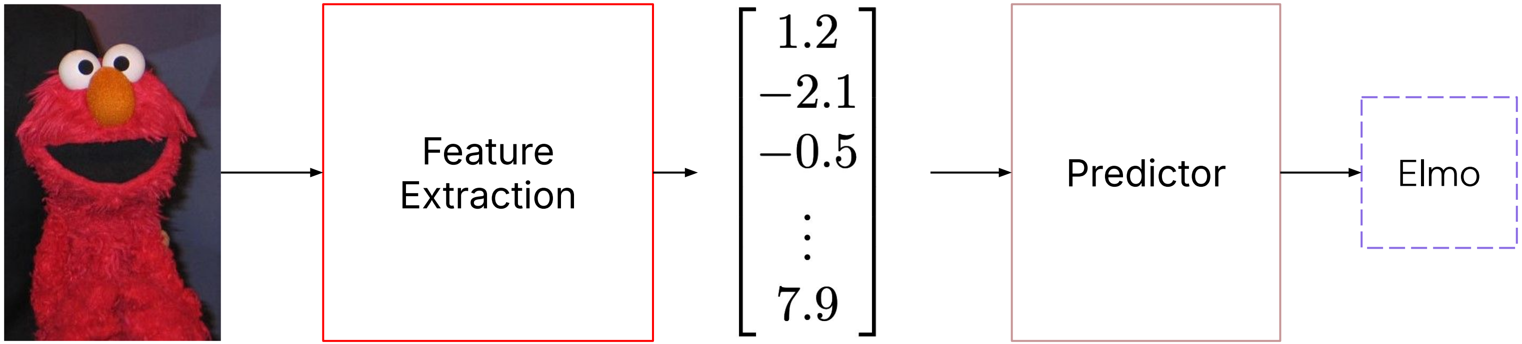
Feature Extraction - Example



Convolutional Neural Networks

A CNN can be thought as having two parts:

- **Feature Extraction** - convert image to a vector
- **Predictor** - convert a vector to a predicted class



Predictor

Contains **dense layers**

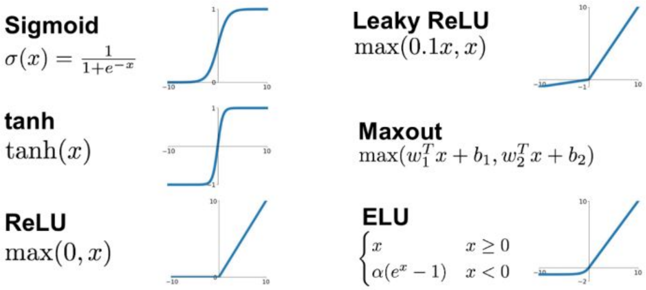
- **Input:** Vector from previous layer ***(D)***
- **Process:** Transform vector from one size to another
- **Output:** Vector output of new size ***(D')***

Predictor

Contains **activation layers***

- **Input:** Vector from previous layer ***(D)***
- **Process:** Apply a ✨ nifty ✨ function to each number in tensor
- **Output:** Activated output of same size ***(D)***

Examples of activation functions



* Most literature considers the activation function as part of a dense layer. We separate it out here to match PyTorch better.

Predictor

The final output is a single **vector** where each component corresponds to how “likely” this image is to belong in *x* class

$$\begin{bmatrix} 1.3 \\ 5.1 \\ 2.2 \\ 0.7 \\ 1.1 \end{bmatrix} \begin{array}{l} \longrightarrow \text{“Likelihood” image is of “abby”} \\ \longrightarrow \text{“Likelihood” image is of “elmo”} \\ \longrightarrow \text{“Likelihood” image is of “zoe”} \\ \longrightarrow \text{“Likelihood” image is of “grover”} \\ \longrightarrow \text{“Likelihood” image is of “oscar”} \end{array}$$

Recap

- CNNs use repeated convolution operations to search for patterns in an image
  - The filters used to search for patterns are learned via ML!
- Information about patterns are then used to make predictions!
- CNNs can be thought of having two major parts
  - Feature extraction - uses convolutional, activation, pooling, flatten layers to convert image to a vector
  - Predictor - converts vector to a predicted class

Practice: How do we implement CNNs in PyTorch? [Notebook]

More Resources

- Convolutional Neural Networks (CNNs / ConvNets) (Stanford)
- Schedule | EECS 498-007 / 598-005: Deep Learning for Computer Vision (Michigan - Lecture 7)

Both the resources above (made by the same person actually) are extremely comprehensive and fill in some of the gaps that we could not address in this crash course.