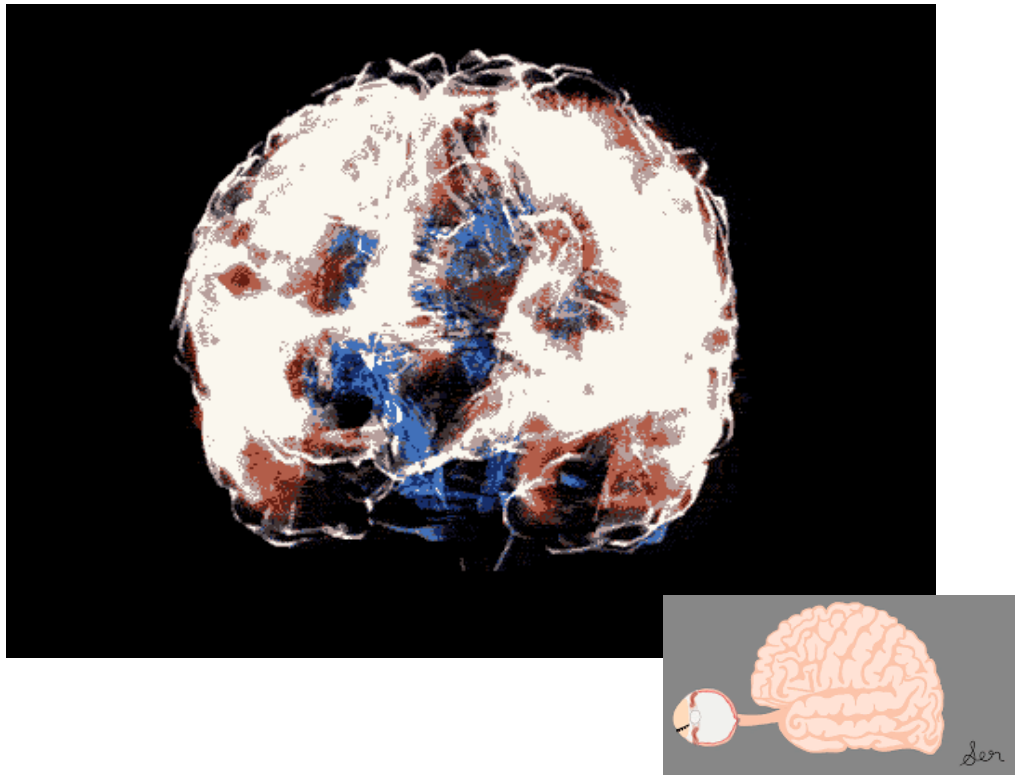
Ahmed Ibrahim

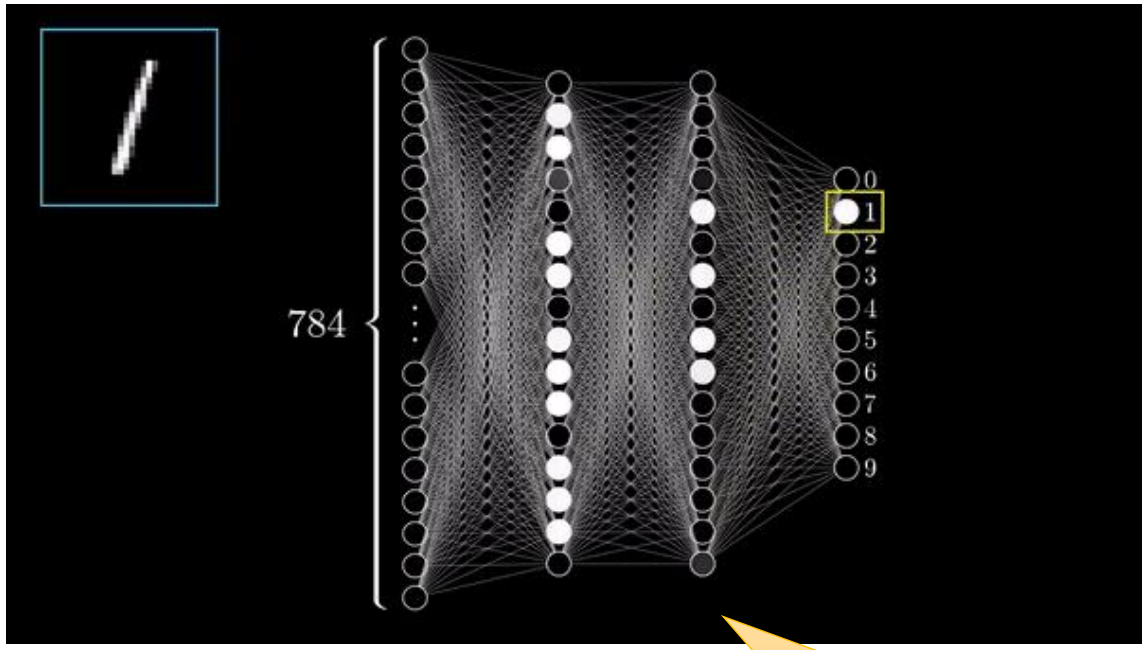
ECE 9039/9309 MACHINE LEARNING

Last Lecture

- Boosting
  - AdaBoosting
  - Gradient Boosting (Gboost)
  - eXtreme Gradient Boosting (XGBoost)
- Neural Network
  - Single-Layer vs. Multi-layer
  - Activation Functions
  - Backpropagation Algorithm

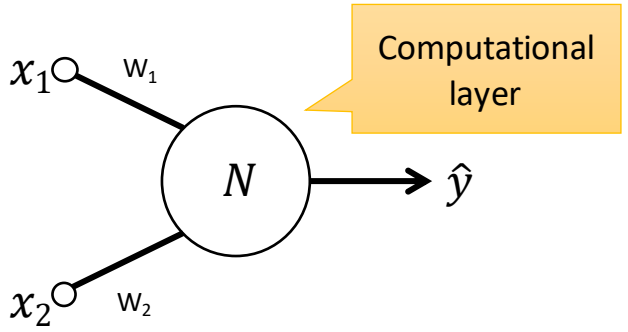
Biological vs. Artificial

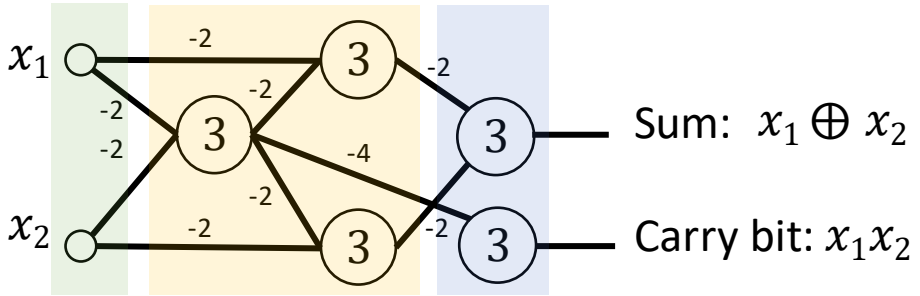




An artificial neural network tries to recognize handwritten numbers.

Recall: Single-layer Network vs. Multi-layer Network





1–bit Adder

- The Perceptron is a single-layer network.
- The input layer is not included in the count of the number of layers in a neural network (it does not perform any computation)
- Since the perceptron contains a single computational layer, it is considered a single-layer network.

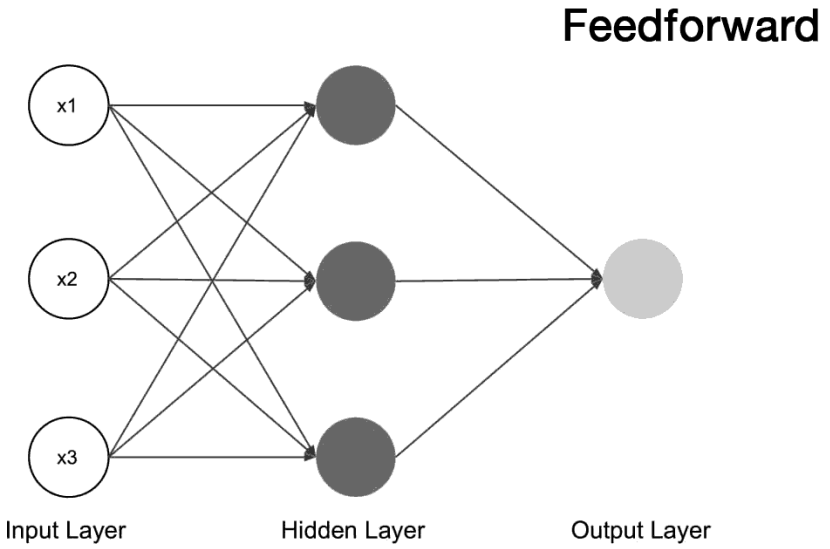
- Multi-layer Perceptrons (MLPs) are a class of **feedforward ANN** that consists of multiple layers of nodes, each layer fully connected to the next one.
- MLPs can model complex, non-linear relationships in data and are used in various applications, from speech recognition to image classification.

Recall: Activation Functions

- The activation functions in ANN are mathematical equations that **determine** its output.
- These functions are crucial as they introduce **non-linearity** into the network, enabling it to learn complex patterns.
- Classical Activation Functions
  - Identity **sign** (or **signum**) activation
  - Identity or **Linear Activation**
  - **Sigmoid** activation is useful where the output is probability [0,1].
  - **Tanh** activation (the hyperbolic tangent function) is similar to the sigmoid function but zero-centered [-1,1].
  - **ReLu** activation – outputs the input directly if it is positive; otherwise, it outputs zero.

Recall: How does ANN learn?

- We need to calculate a Loss/Cost over the whole training set, which is the objective the neural network tries to optimize.
- Neural networks learn through a process called "**backpropagation**," which essentially applies the **chain rule** from calculus used to compute **gradients** for all weights in the network.
- The computed gradients are used to update the weights and biases. The basic form of this step is to subtract the gradient multiplied by a learning rate from each weight and bias.



Source: https://towardsdatascience.com/creating-neural-networks-from-scratch-in-python-6f02b5dd911

Outlines

- Some Practical Exercises
- Design Neural Networks
- Popular Frameworks for ANN
- ANN Applications: Image Classification
  - Common Feature Extraction Approaches
- Deep Learning
  - Convolutional Neural Networks
- Deep Learning Challenges

Recall: Feed-forward Neural Network

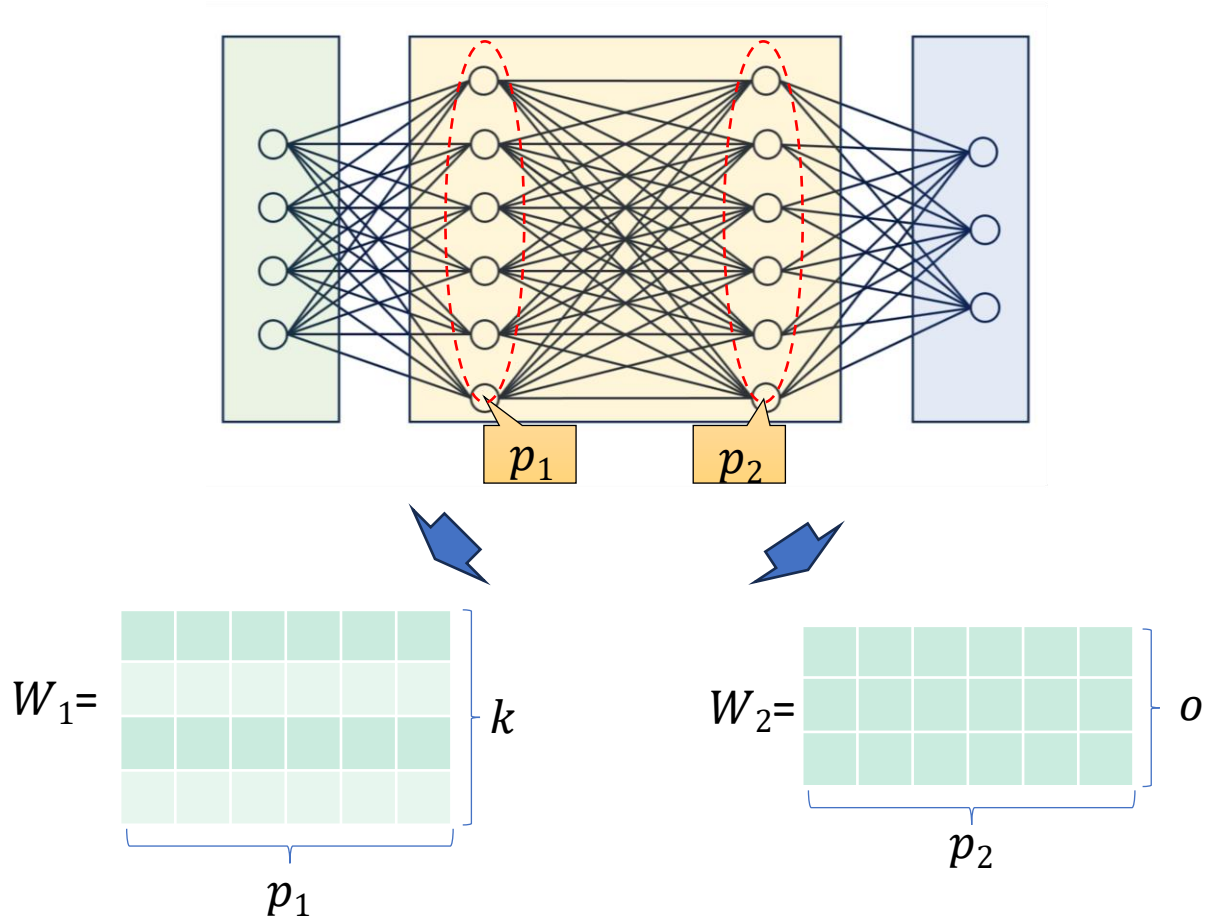
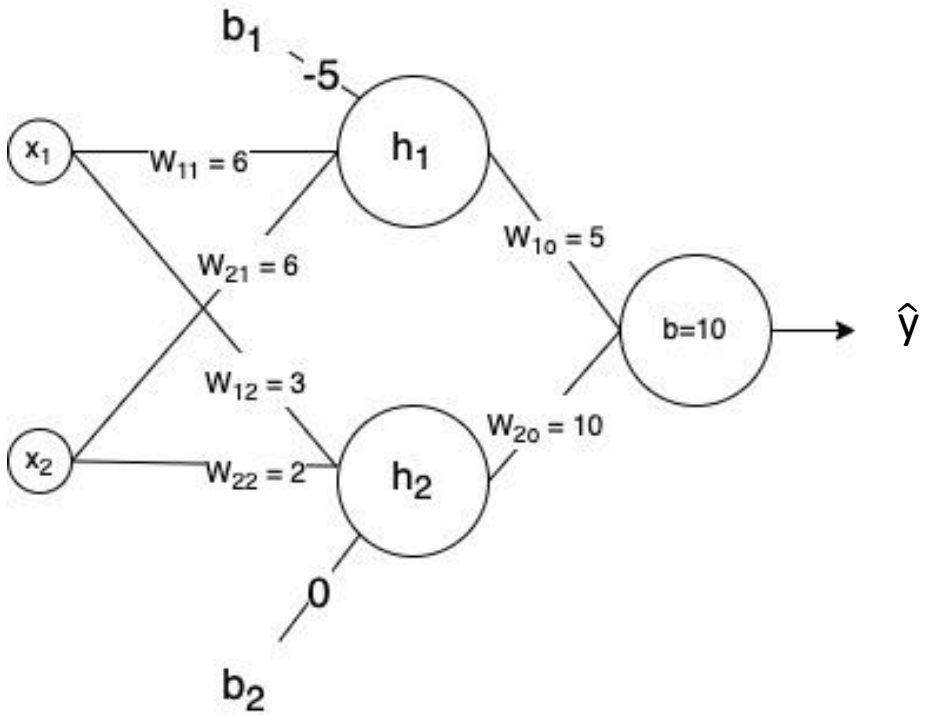


Image source: https://vitalflux.com/feed-forward-neural-network-python-example/

Practical Exercise #1



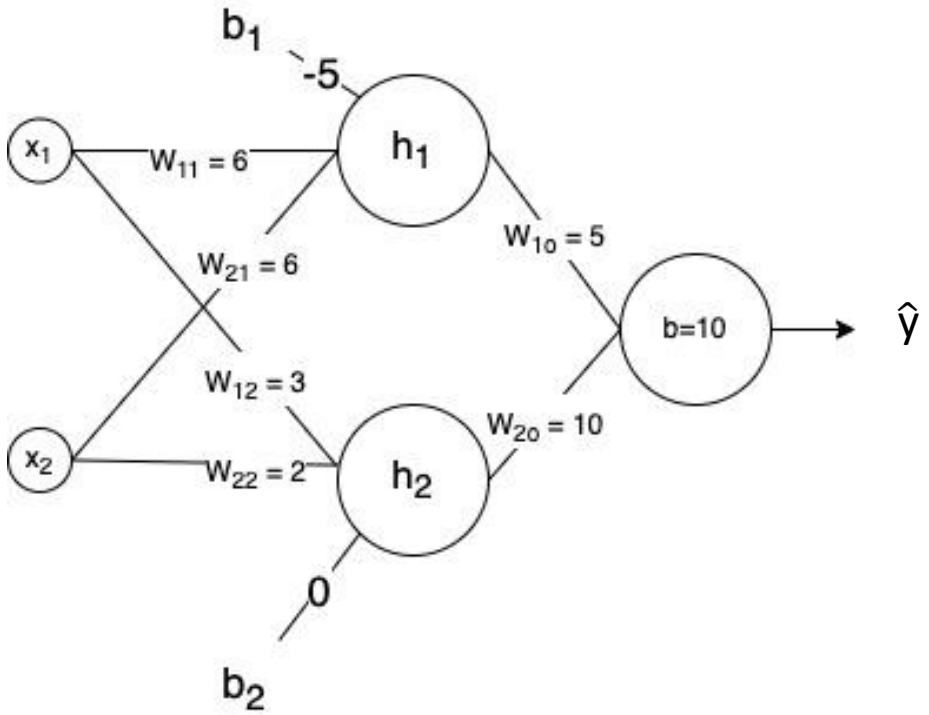
- Assume a sigmoid activation function; what is the predicted output for the following inputs (0,0),(0,1),(1,0),(1,1), and round to the nearest integer?
- Weight Matrix W: $W = \begin{bmatrix} 6 & 3 \\ 6 & 2 \end{bmatrix}$
- For Input (0,0):
  - Hidden layer input (before activation):

$$a = W \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -5 \\ 0 \end{bmatrix} = \begin{bmatrix} -5 \\ 0 \end{bmatrix}$$

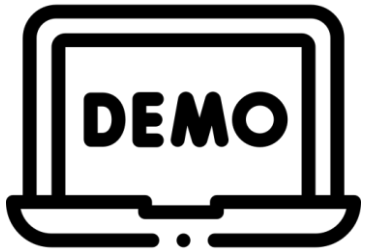
  - Hidden layer output (after sigmoid activation):

$$\sigma(a) = \begin{bmatrix} \sigma(-5) \\ \sigma(0) \end{bmatrix} \approx \begin{bmatrix} 0.00669285 \\ 0.5 \end{bmatrix}$$

Practical Exercise #1 (cont.)

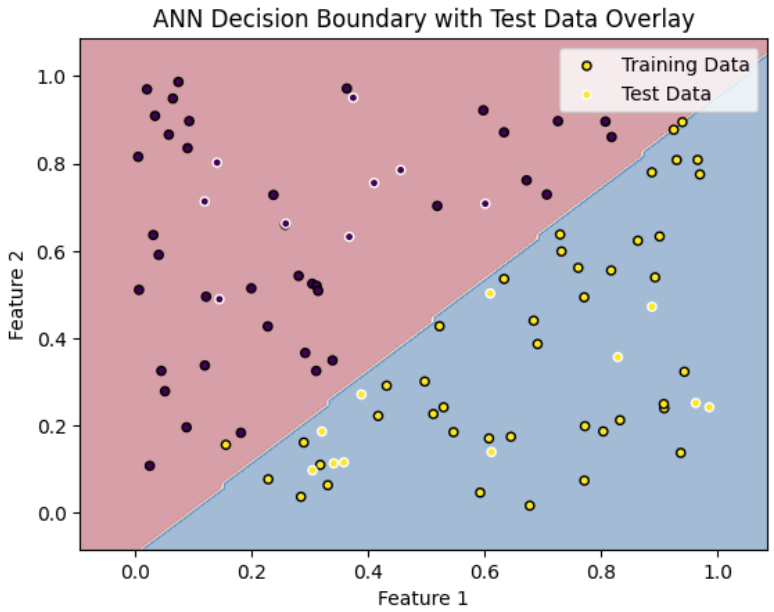


- For Input (0,0):
  - Output layer input (before activation):
    $= (0.00669285 \times 5) + (0.5 \times 10) + 10$
    $\approx 15.033464254621425$
  - Output layer output (after sigmoid activation):
    $= \sigma(15.033464254621425)$
    $\approx 0.9999997041651716 \longrightarrow 1$

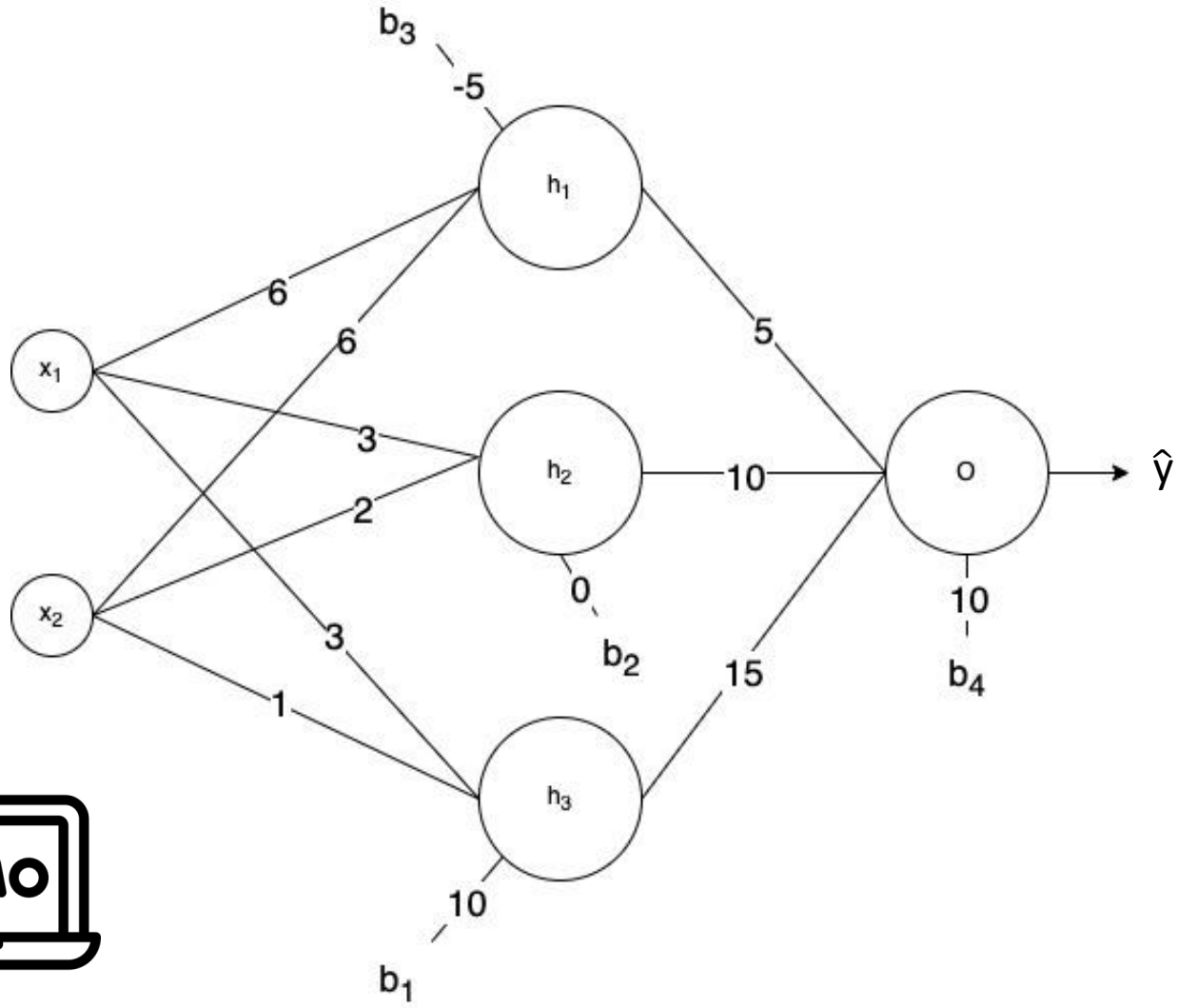


Design Neural Networks

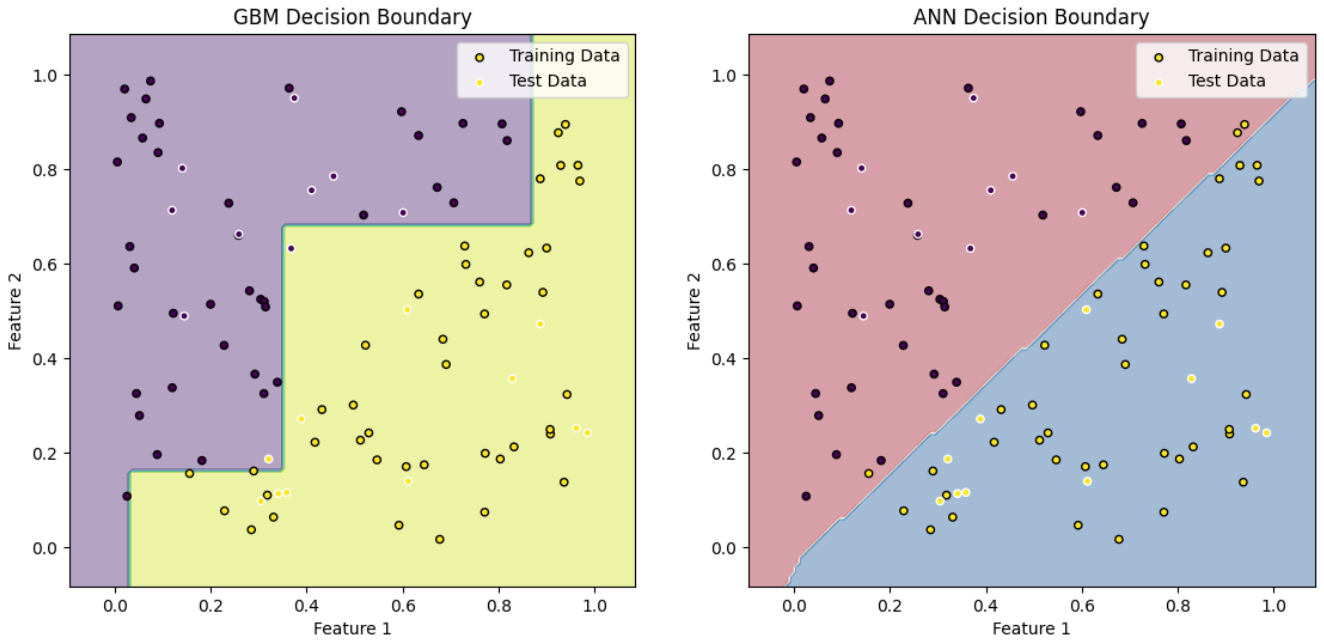
Practical Exercise #2



DEMO

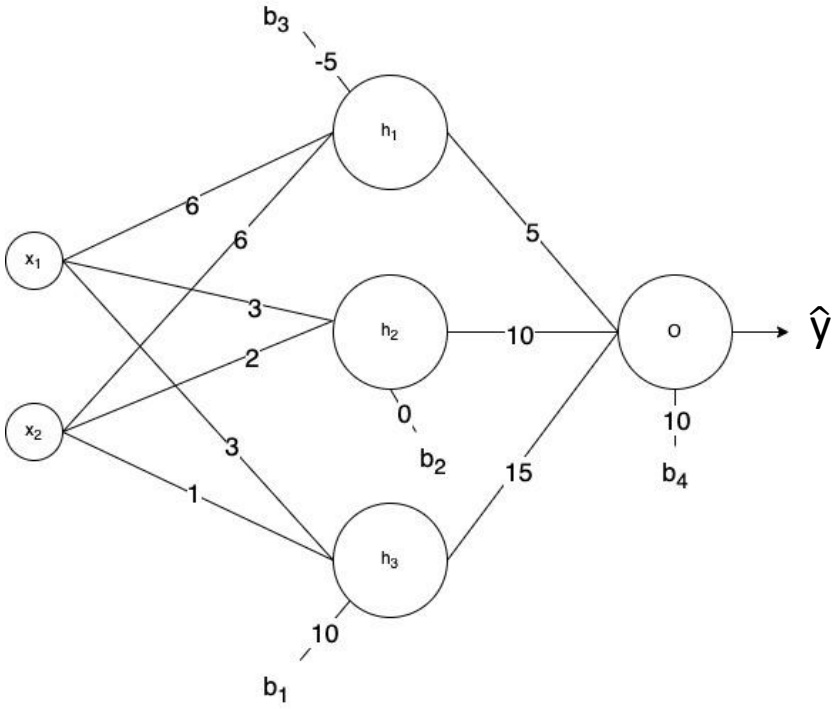


Practical Exercise #3



Gradient Boosting Classifier

ANN Classifier



Popular Frameworks for ANN

- Keras is known for its ease of use and simplicity, making it a great choice for beginners.
- It was first released in 2015, and the library's creator is French programmer **François Chollet**.
- It acts as a high-level API capable of running on top of **TensorFlow**, allowing fast experimentation with deep neural networks.
- **Keras** is designed to be user-friendly, modular, and extensible

- **TensorFlow**, developed by **Google**, is a more comprehensive framework offering high-level and low-level APIs.
- It excels in large-scale machine learning projects and is known for its flexibility and extensive functionality, including support for GPUs and TPUs for accelerated computing.
- TensorFlow is suitable for both research and production, offering tools like TensorFlow Serving for deploying models easily

- PyTorch, developed by **Facebook's** AI Research lab, is favored for research and development due to its flexibility, ease of use, and dynamic computation graph.
- It allows for easier debugging and has been gaining popularity for its support for GPU acceleration.
- PyTorch is particularly appreciated in the academic community for enabling rapid prototyping and experimentation.

ANN Applications

Example: Classifying Images From Pixels

Input Data:
64x64 Images

ANN

Target:
200 Class Labels
(e.g. cat)

ANN Goal:
Train a classifier to predict the class label for a given image, assuming it prominently features an object from one of *C* classes.

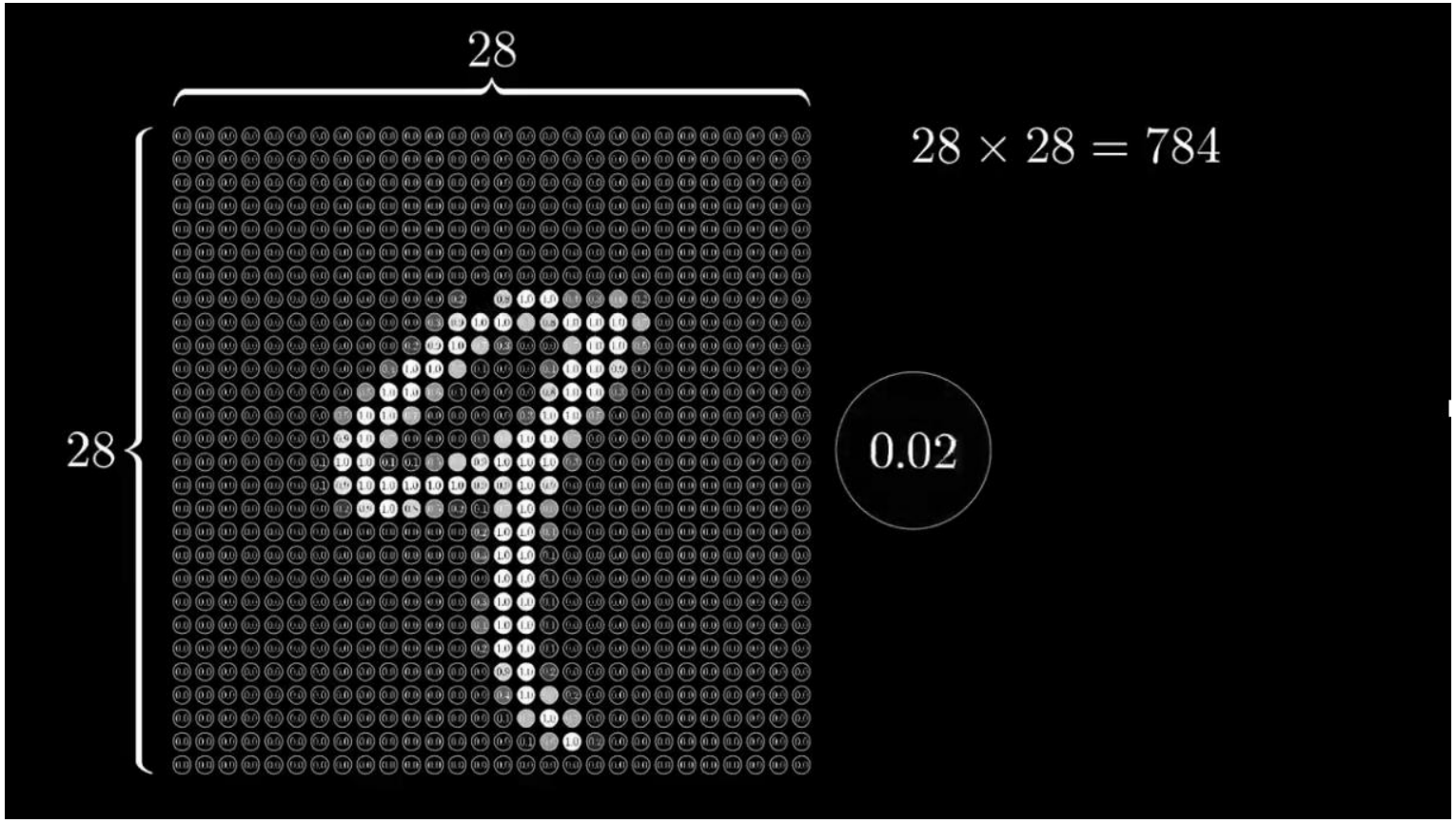
Famous datasets: ImageNet, CIFAR-10 dataset, MNIST, Visual Genome, Celebfaces, Labelled Faces in the Wild

- Subset of **ImageNet** challenge dataset
  - 500 x 200 Training Images
  - 50 x 200 Testing Images (Not Used) We will use this in the Tutorial. https://tiny-imagenet.herokuapp.com/
- **ImageNet** is a large-scale, structured dataset of annotated images to help develop and evaluate algorithms capable of recognizing and understanding objects in images.
- The dataset contains more than 14 million images and was developed by researchers from Stanford University, Princeton University, and other institutions.

Building an Image Classifier

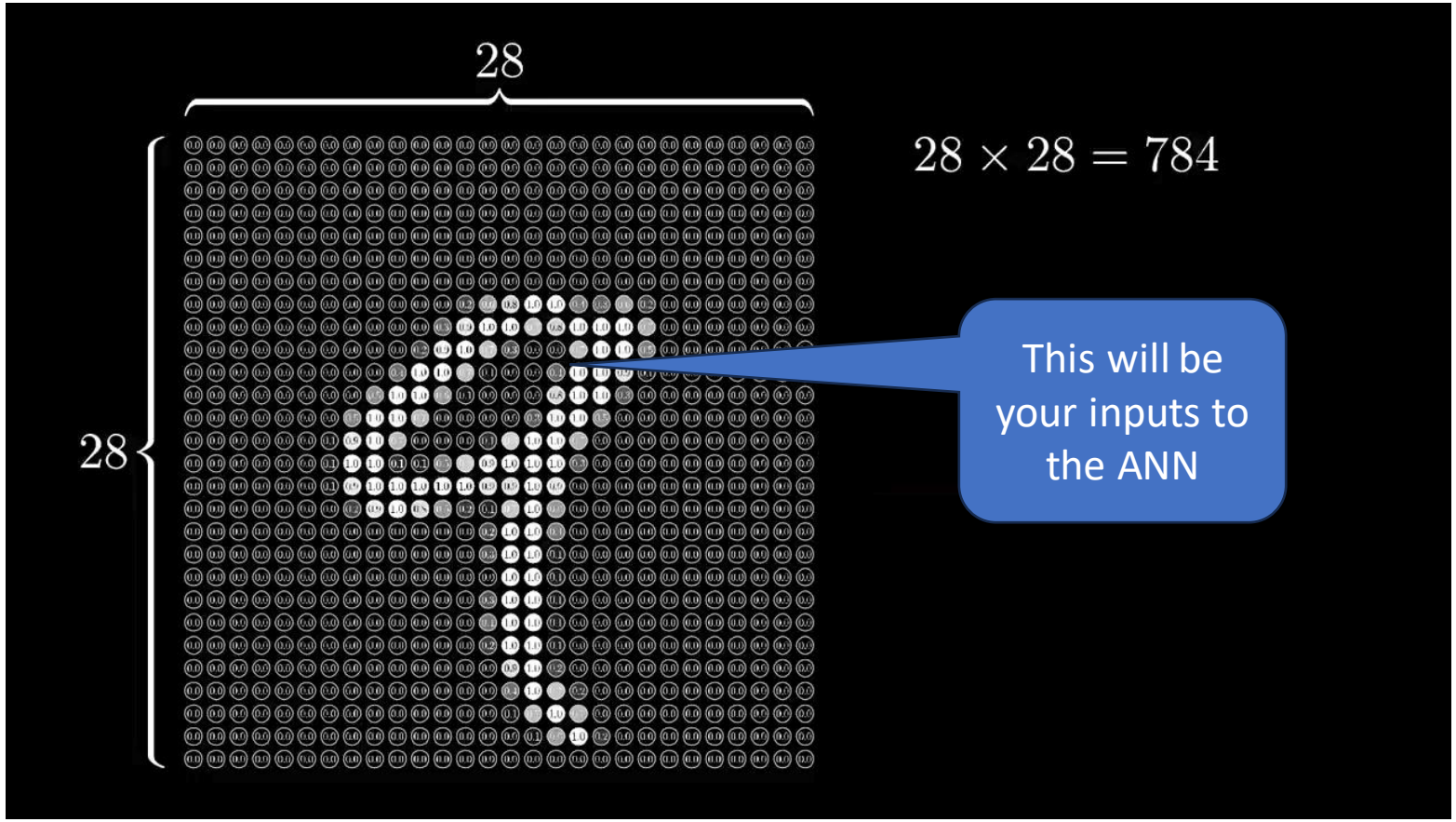
- Below is a general approach to tackling an image classification task:
    1. **Collect a Dataset** – Your dataset should contain images and corresponding labels.
    2. **Preprocess the Data** – This involves steps like resizing images to a uniform size and normalizing pixel values (usually to the range [0,1] or [-1,1]).
    3. **Split the Dataset** – Divide your dataset into at least two parts: training the model and testing its performance. A common split ratio is 80% for training and 20% for testing.
    4. **Choose a Model Architecture** – Start with a simple neural network for smaller datasets or leverage more complex architectures for more extensive datasets.
    4. **Train the Model** – Use your training dataset to train the model. This involves feeding the input images into the model, comparing the predicted output with the actual labels, and adjusting the model's weights through backpropagation.
    5. **Evaluate the Model** – Test the model's performance using the test dataset. Metrics such as accuracy, precision, recall, and F1 score are commonly used to evaluate classification models.
    6. **Fine-tune and Iterate** – Based on the model's performance, you may need to adjust the architecture, hyperparameters, or data preprocessing steps.
    7. **Deployment** – Once the model's performance is satisfied, you can deploy it to classify new images according to your application's needs.

Classifying Images From Pixels



Source: https://www.youtube.com/watch?v=aircAruvnKk

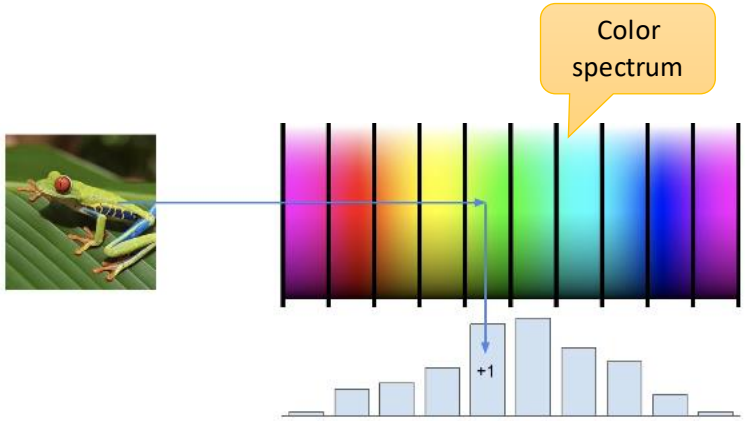
Classifying Images From Pixels (cont.)

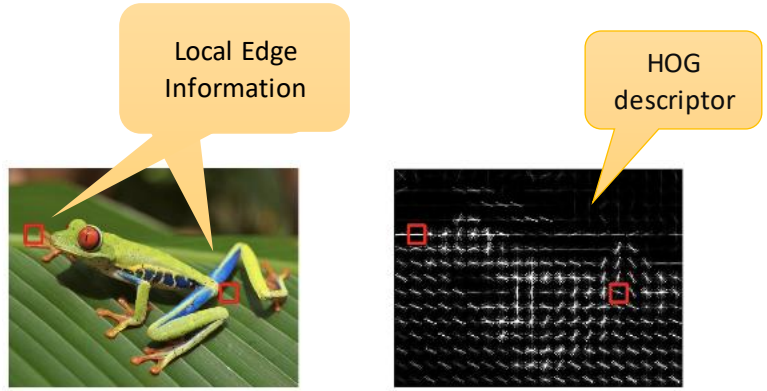


Source: https://www.youtube.com/watch?v=aircAruvnKk

Common Feature Extraction Approaches

- Color Histogram:
  - This spectrum illustrates how colors in the image are mapped to specific regions of the color space.
  - A color histogram represents the frequency of different colors within the image. Each bar in the histogram corresponds to a color in the spectrum.
- Histogram of Orientated Gradients (HOG):
  - It works by dividing the image into small connected regions called cells, and for each cell, accumulating a histogram of gradient directions or **edge orientations** for the pixels within the cell.
  - The combination of these histograms then represents the descriptor of the object.
  - The number of neurons in the **input layer** should match the number of features in your HOG descriptor.





Feature Learning For Image Classification

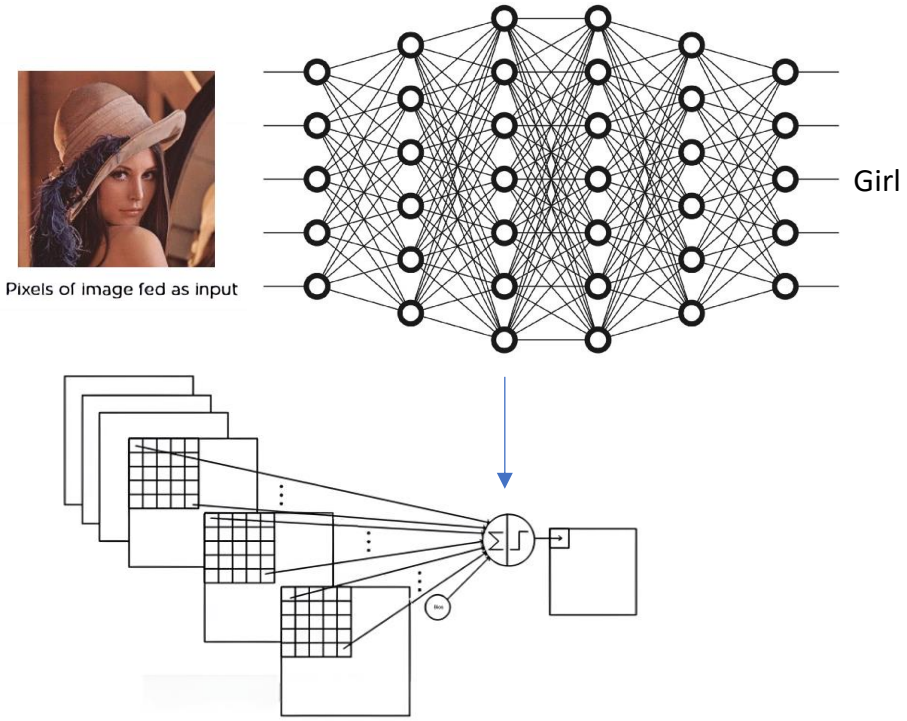
- Feature learning for image classification is a critical aspect of ML and computer vision, enabling algorithms to **automatically identify** and use the most relevant features in images for classification tasks.
- This approach differs from traditional methods that rely on manually designed and extracted features.
- **Deep Neural Networks** (especially) become a cornerstone in such a domain. They learn **hierarchical representations** of images automatically through the training process.

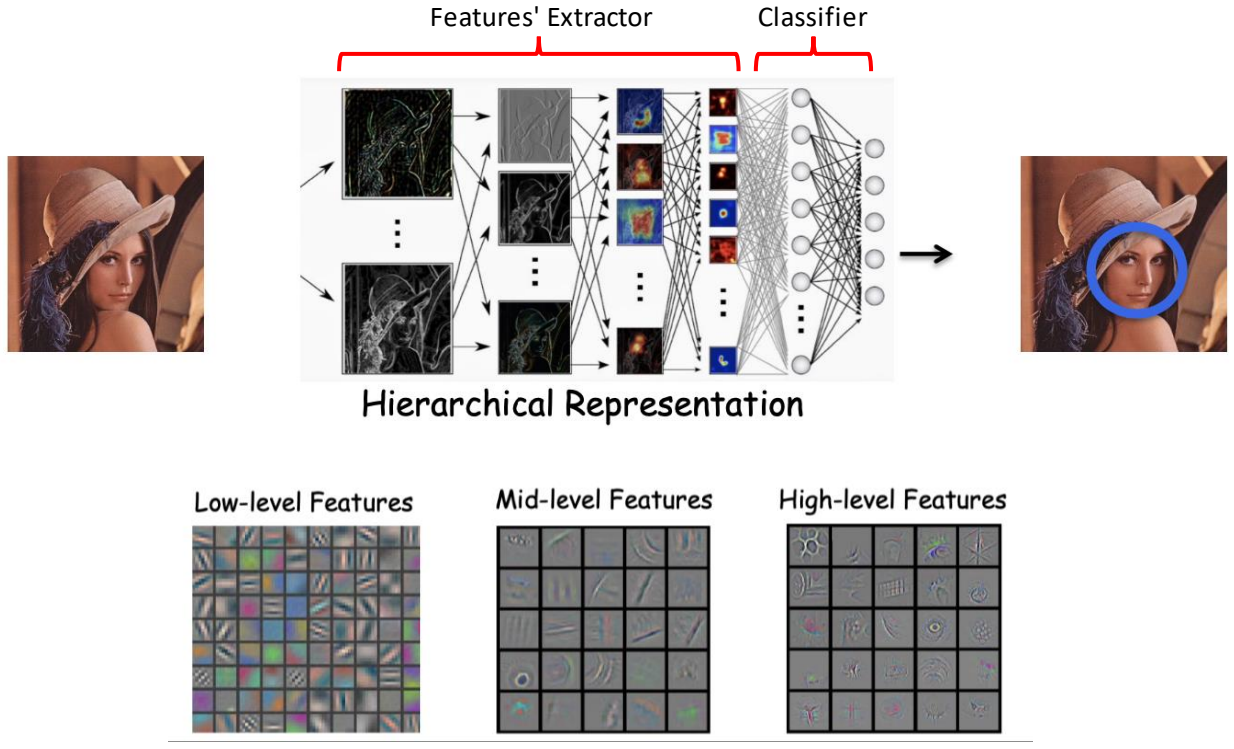
Attendance

You can use the provided link if you don't have a mobile phone or if your phone lacks a QR-Code reader – https://rebrand.ly/ECEMar5

Deep Learning

Deep Neural Network





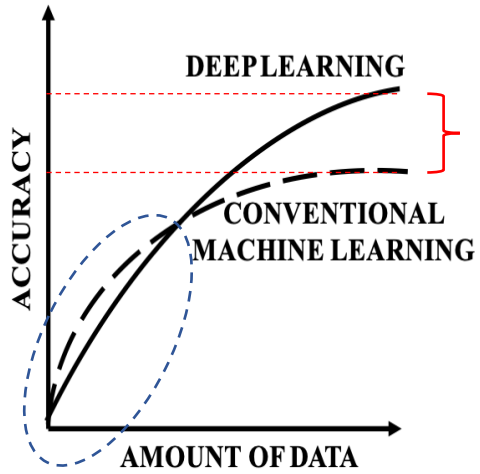
Hierarchical Representation of Features

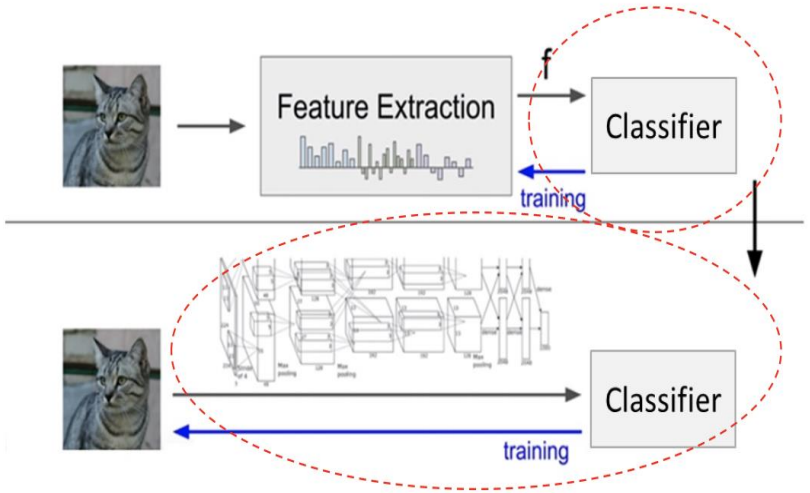
[Kamnitsas, 2017]

Convolutional Neural Networks (CNN)

- Each layer in a CNN learns to recognize increasingly complex patterns.
- The first few layers might learn to recognize **edges** and **textures**, while deeper layers might recognize more abstract concepts like **object parts**.
- CNN architectures like **AlexNet**, VGG, Inception, and **ResNet** have been particularly successful in image classification tasks.

Deep Learning vs Classical Machine Learning

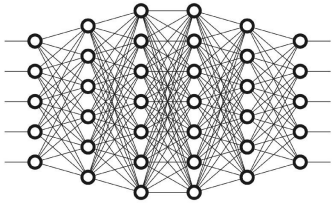




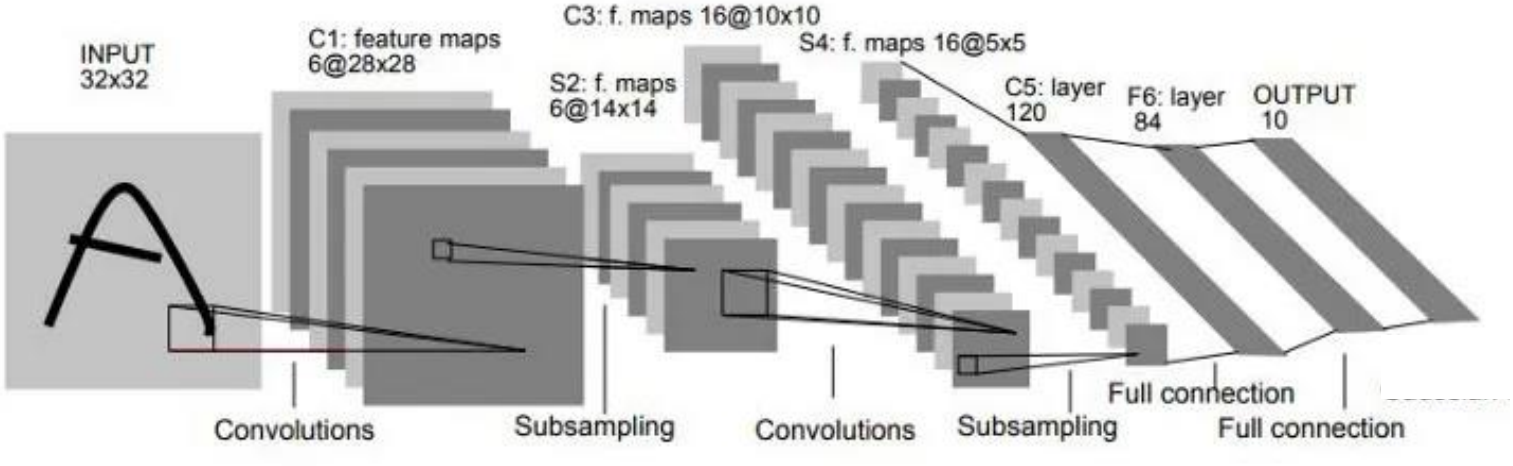
[Textbook: NNDL-1.1]

- For smaller datasets, conventional ML often provides slightly better performance.
- Conventional ML often provides more interpretable insights and ways to handcraft features.
- Deep learning (DL) methods (specifically CNN) tend to dominate for larger data sets.
- Progress in computational tools enables DL algorithms to process large sets of data in a short period of time.

LeNet Architecture



- The CNN applies a series of **convolutional layers**, where each layer consists of a set of filters (kernels) that perform convolution operations. These filters slide over the image to produce feature maps.
- After each convolution operation, a nonlinear activation function, typically ReLU is applied. ReLU allows us to learn more complex patterns.

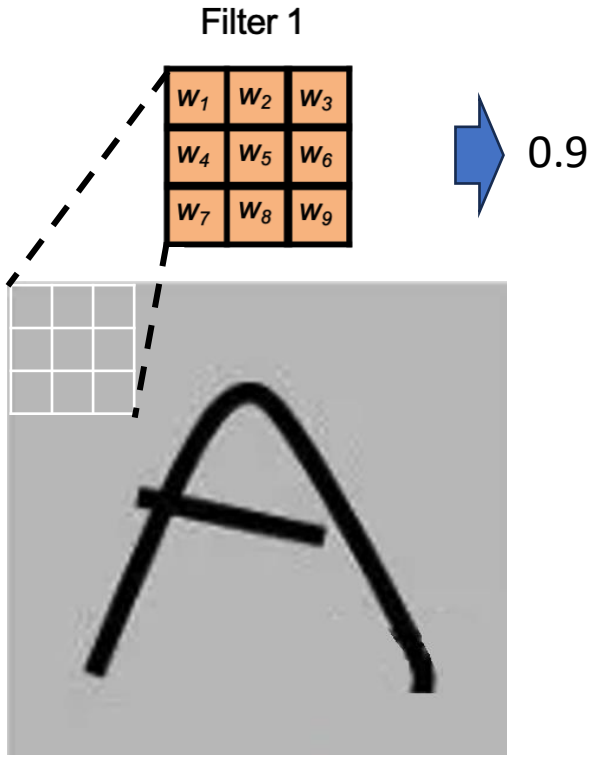


- Following ReLU, max pooling is often performed. Max pooling is a down-sampling strategy that reduces the input volume's spatial dimensions (width and height) for the next convolutional layer.
- Max pooling selects the maximum value from a group of pixels in a region defined by the pooling size (e.g., a 2x2 pixel area).

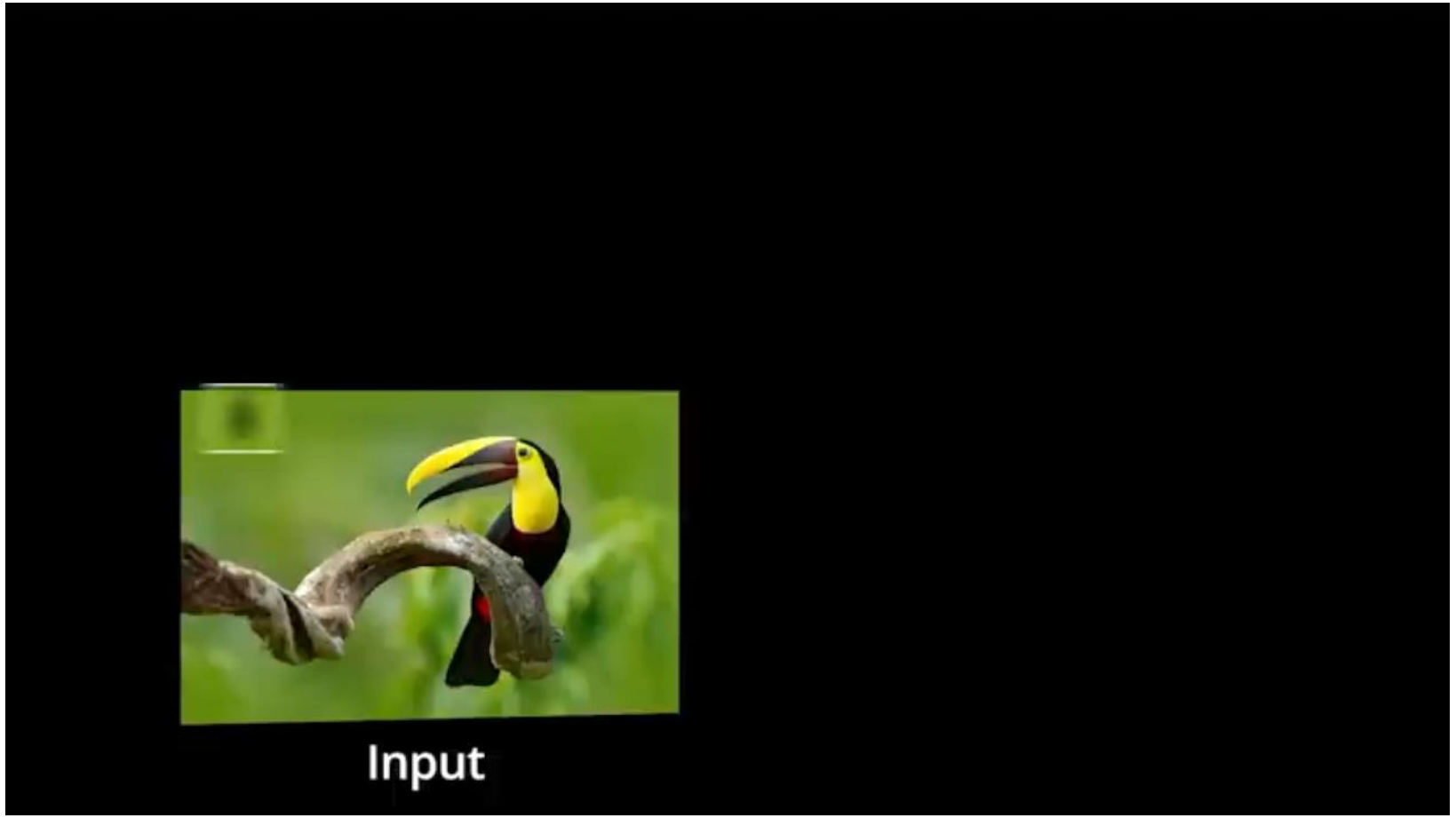
Source of image:
simplilearn

Feature Transformations For Images

- In image classification, individual pixels are not sufficient for effective discrimination between categories, as their values can be quite similar across various images, and they do not capture the necessary contextual details.
- **Convolution** involves applying a filter across the image, where an operation is conducted between the filter and the underlying pixel values. Each position yields a singular output that contributes to a **feature map**.
- This map highlights certain image attributes/features, such as edges, textures, or color transitions, influenced by the filter weights.
- For instance, a 3x3 filter could scan the image, resulting in an output value indicating the strength of a detected feature at that specific location on the feature map.

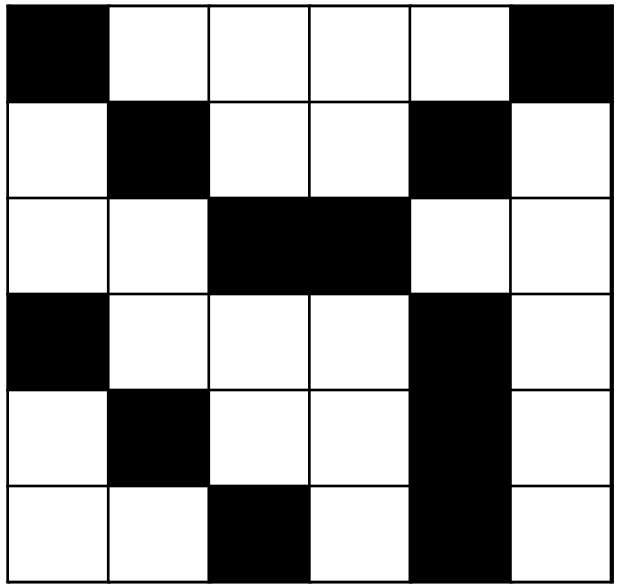


Feature Map



Source:
simplilearn

Applying Filters on an Image



6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

$Filter_1$

-1	1	-1
-1	1	-1
-1	1	-1

$Filter_2$

⋮ ⋮

Each filter detects a small pattern (3 x 3)

Applying 1st Filter

stride=1

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

6 x 6 image

Dot product

3 -1

Sum

1	0	0
0	1	0
0	0	1

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Applying 1st Filter (cont.)

stride=1

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

6 x 6 image

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

A Feature Map

1	-1	-1
-1	1	-1
-1	-1	1

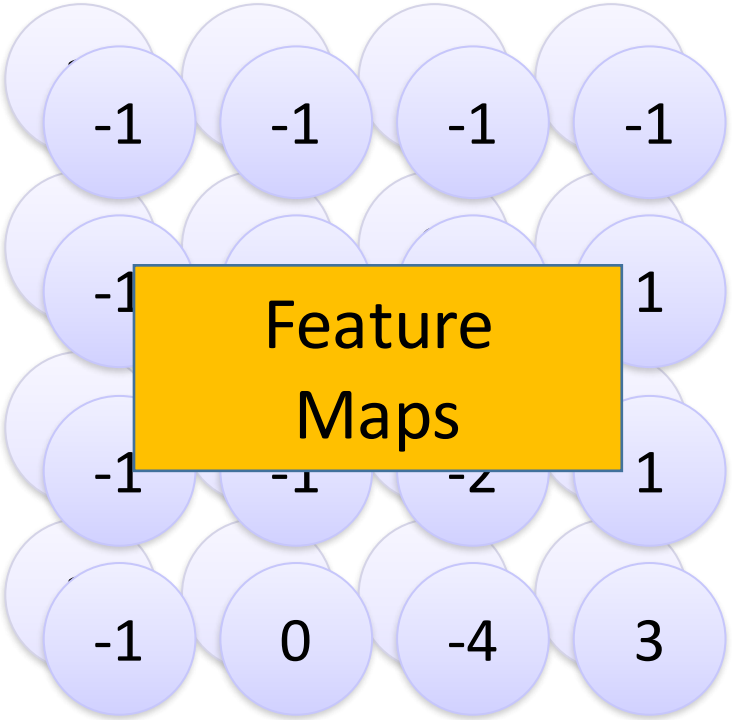
Filter 1

Applying 2nd Filter

stride=1

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

6 x 6 image



Repeat this for each filter

-1	1	-1
-1	1	-1
-1	1	-1

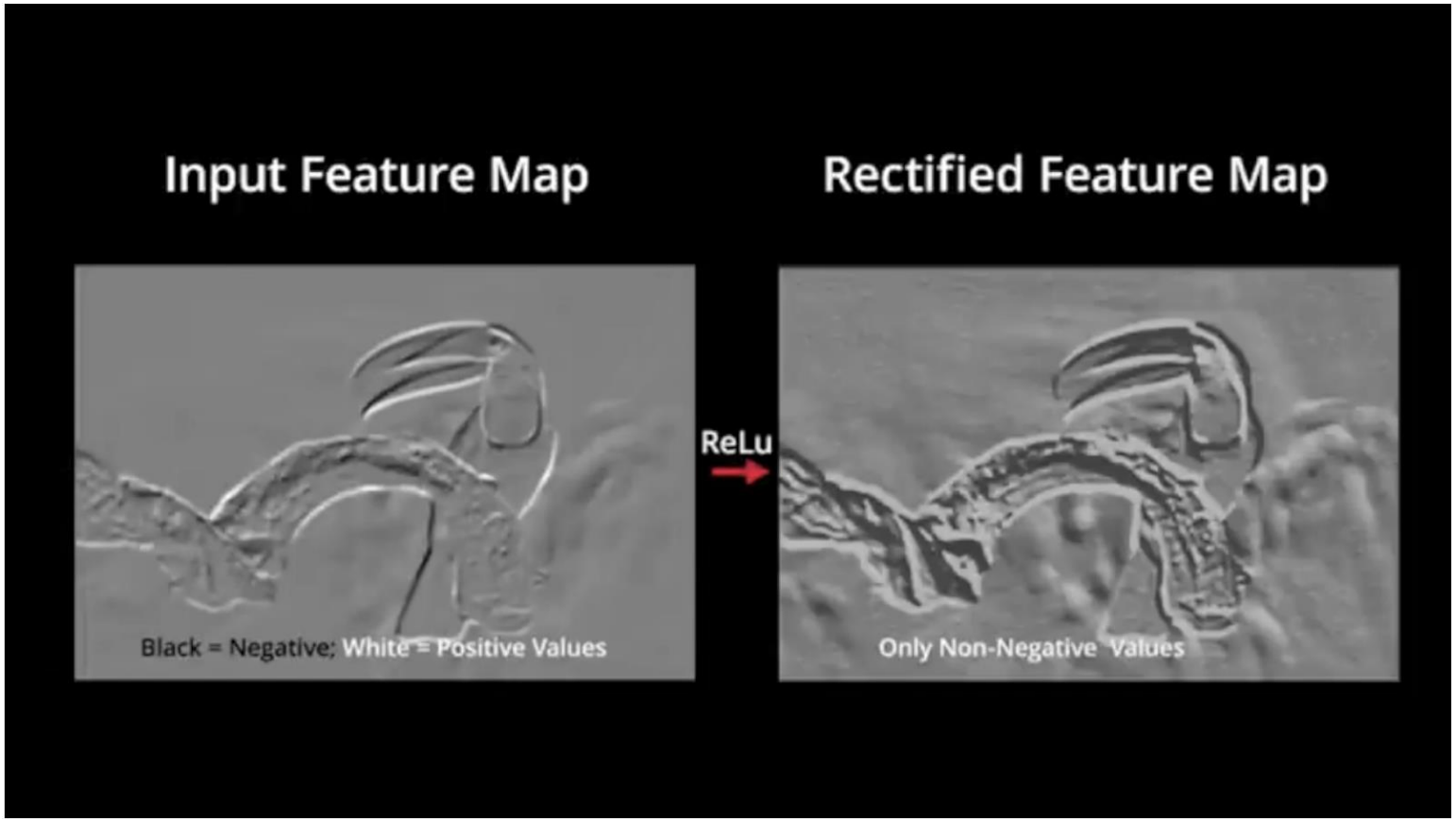
Filter 2

Common Types of Filters (FYI)

- **Edge Detection Filters** – These filters can highlight vertical, horizontal, or diagonal edges in images by responding to areas of high-intensity change. Common examples include Sobel, Prewitt, and Roberts' cross filters.
- **Sharpening Filters** – These enhance the edges, making the image look more defined. This is often done by accentuating the difference in color values between adjacent pixels.
- **Gaussian Blur Filters** – These use a Gaussian function to smooth the image, reducing detail and noise. This is useful for noise reduction and creating a hierarchy of features at multiple scales.
- **Low-Pass Filters (LPF)** – These filters allow low-frequency components from the image, such as general colors and shapes, while filtering out high-frequency components, such as noise and detailed textures.
- **High-Pass Filters (HPF)** – Conversely, these filters allow high-frequency components to pass through, which means they can be used to detect sharp changes in intensity, like fine details.

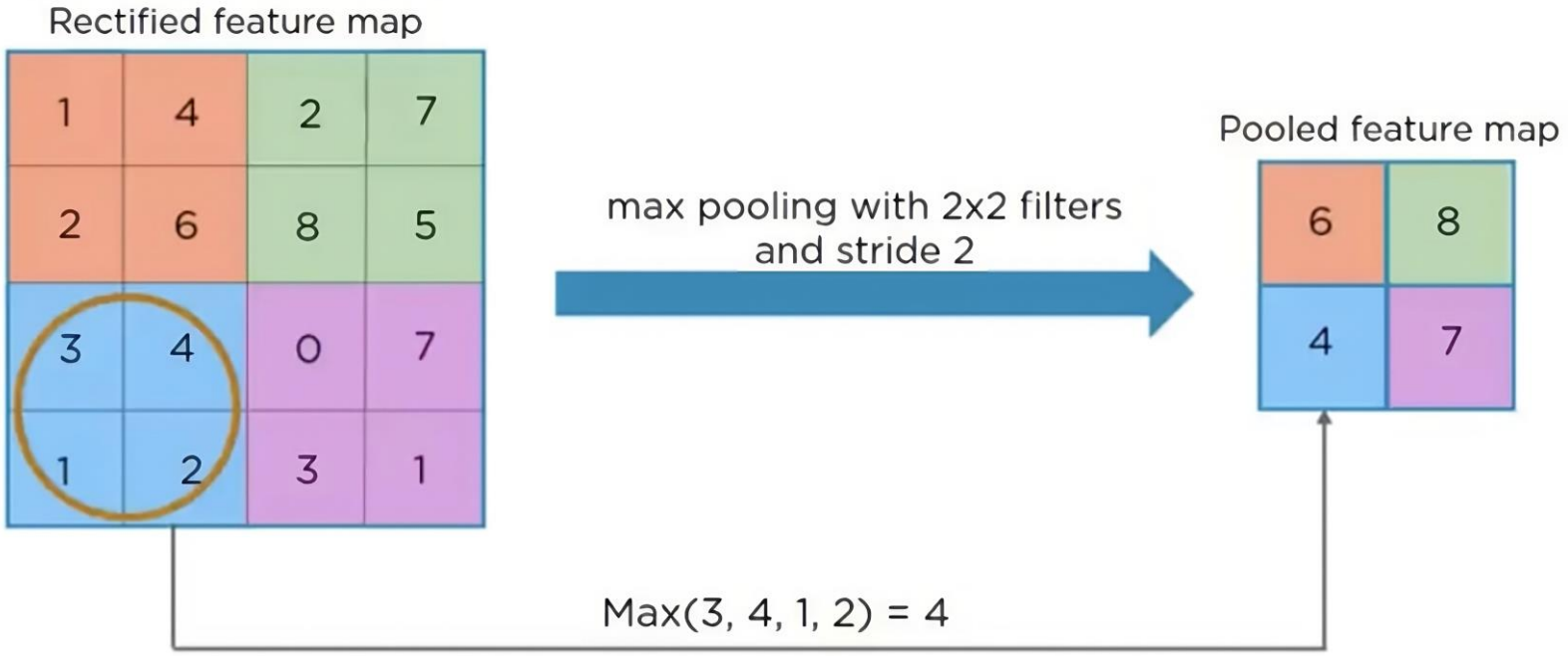
The ReLU Activation Function

- The ReLU (Rectified Linear Unit) activation function is a piecewise linear function that will output the input directly if it is positive; otherwise, it will output zero.
- It has become the default activation function for many types of neural networks because it is easy to compute and often results in faster training.



Source:
simplilearn

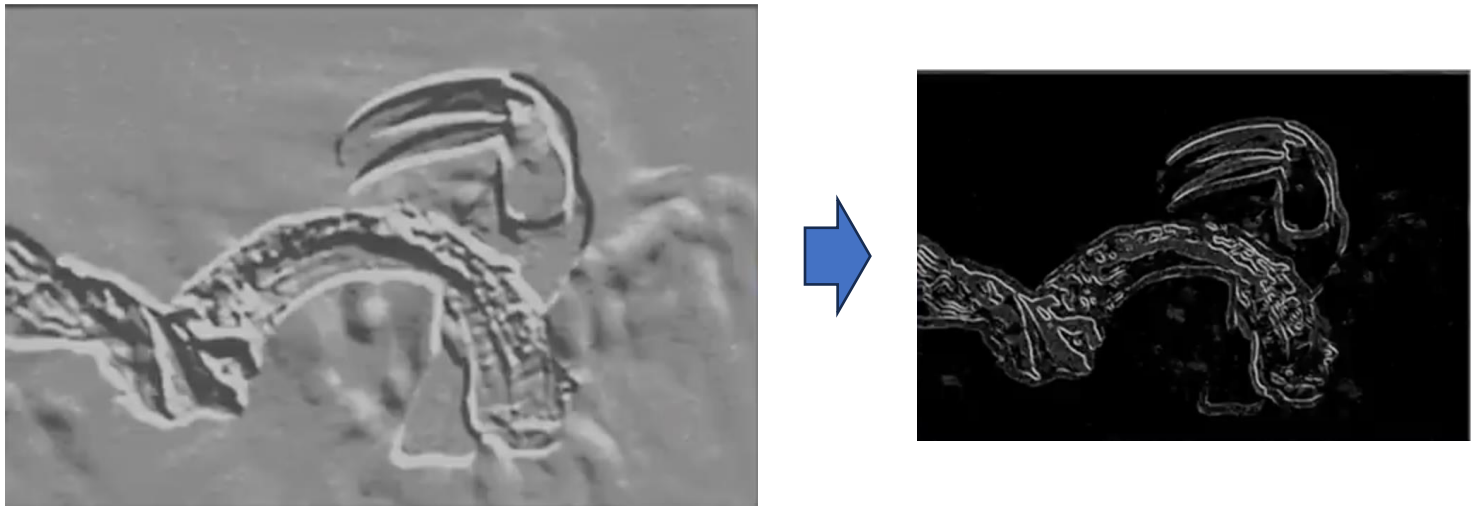
Pooling Layer



- A pooling layer then processes the rectified feature map(s). Pooling serves as a down-sampling technique that reduces the size of the feature map.

Source:
simplilearn

Pooling Layer (cont.)

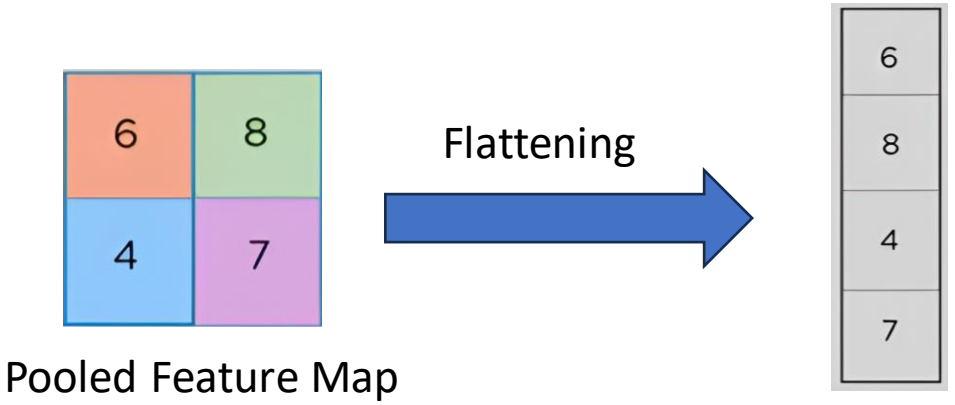


Detects the bird's outlines, angles, and various characteristics

- The pooling layer uses different filters to recognize parts of the image, like edges, corners, and body parts such as eyes and beaks.

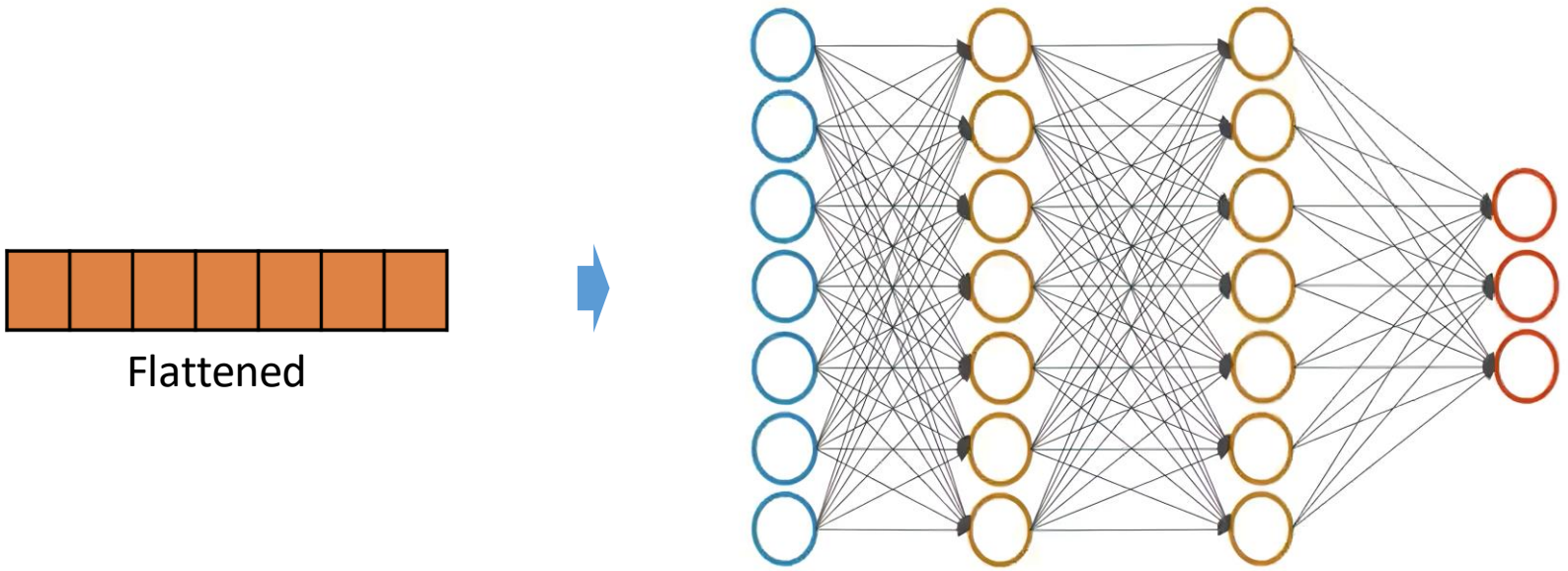
Source of image:
simplilearn

Flattening



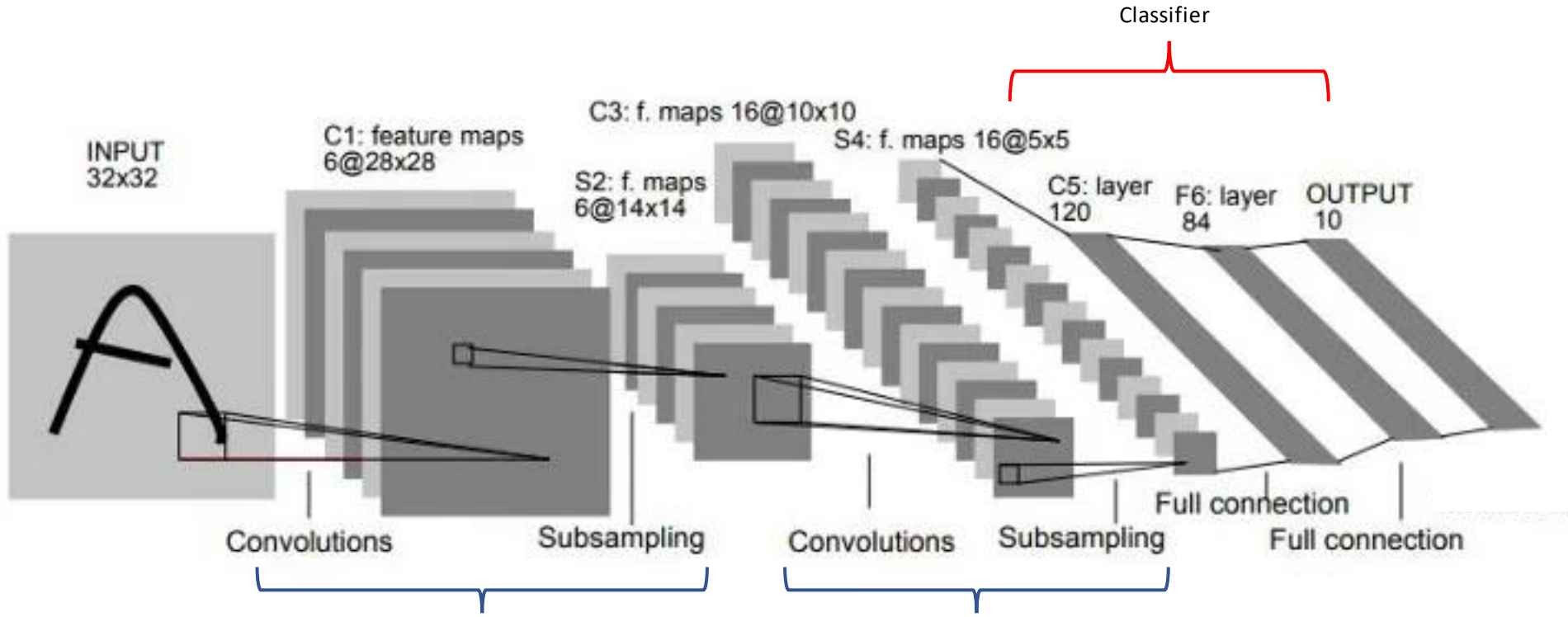
- This step is essential for transitioning from the spatially structured layers to the densely connected layers of the network. Essentially, it lines up all the features detected by the pooling process into a **single string** of values, preparing the data for the final classification part of the neural network.
- This transformation is critical as it allows the network to understand and process the image data in a format suitable for making predictions.

Fully Connected Layer

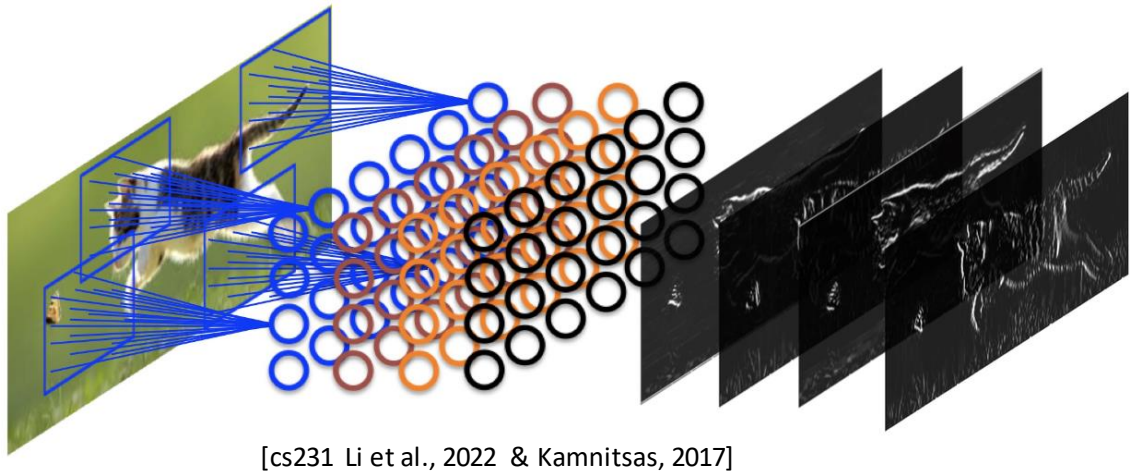


- The flattened output, derived from the (last) pooling layer, serves as the input for the fully connected layer. Here, the neural network performs the task of classifying the image.
- This layer integrates the features identified and flattened, such as **edges**, **textures**, and **patterns**, to decide what the image represents.

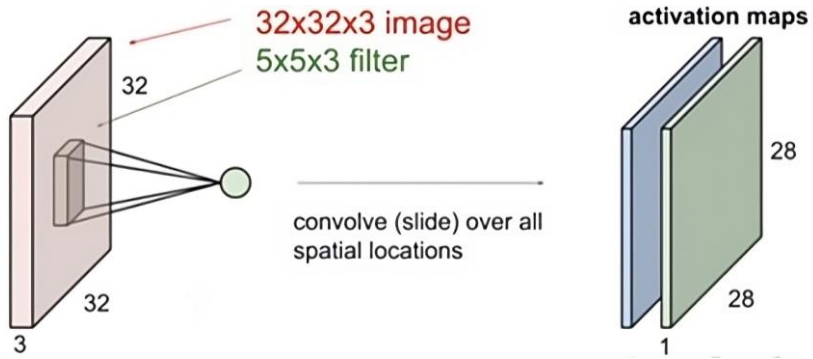
Recall: LeNet Architecture



Multi-channel (color) Images



[cs231 Li et al., 2022 & Kamnitsas, 2017]



A single-channel feature map in the output.

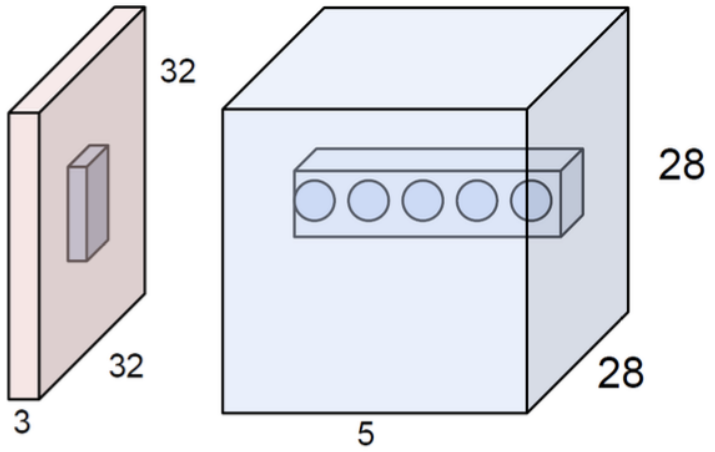
- **For color images**, each filter must have the same number of channels as the input image. So, filters also have a depth of 3 for an RGB image. The filter convolves with the image by performing element-wise multiplication with each channel and summing the results. This produces a single-channel feature map in the output.

$$\text{Output size} = \frac{\text{W}-\text{F}+2P}{S} + 1$$

$$\text{Output size} = \frac{32-5+2(0)}{1} + 1 = \frac{27}{1} + 1 = 28$$

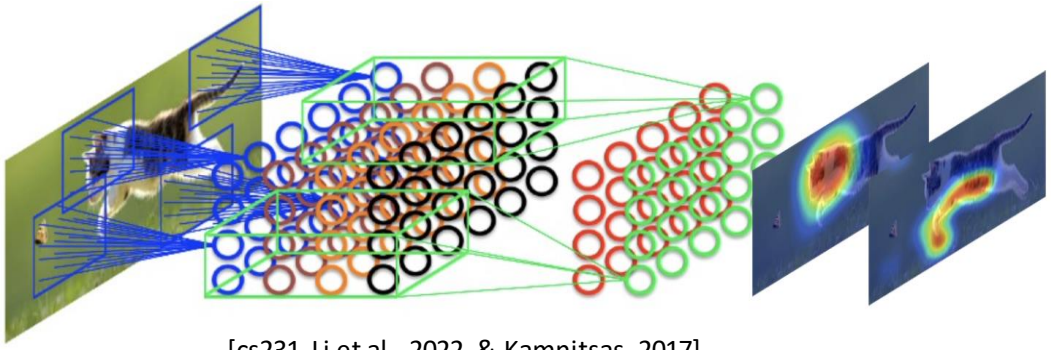
- **W** is the width (or height) of the input image,
- **F** is the filter size,
- **P** is the padding, and
- **S** is the stride.

Multiple Filters

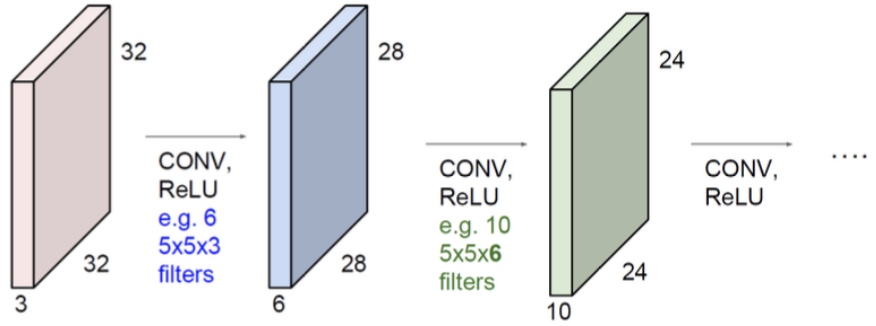


- **Multiple Filters** – CNNs don't just apply one filter to the input image but multiple filters. Each filter can look for different features in the image, such as **edges in different orientations**, **textures**, or **specific color patterns**.
- **Stacking Feature Maps** – When multiple filters are applied to a multi-channel image, the results are stacked along the depth dimension to form the **output volume**. If 5 different filters are used, there will be 5 feature maps stacked together.

Multiple Convolutional Layers



[cs231 Li et al., 2022 & Kamnitsas, 2017]



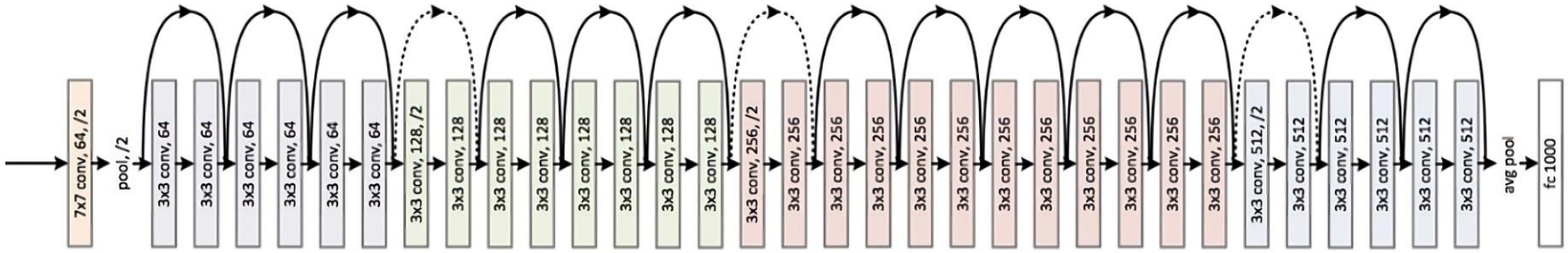
- **Subsequent Layers** – As this output volume is passed through more layers in the CNN, the network continues to combine and recombine these features, detecting increasingly complex patterns. Later layers might work with many channels, far more than the original three channels in the input image.
- **Depth of Layers** – The depth of the layers increases as we move deeper into the network. This depth corresponds to the number of filters applied at each layer. Each layer's depth becomes the "channel" dimension for the next layer.

Deeper Architectures (Schematic Representation)

AlexNet 2012

VGG16 2014

- The numbers beside the layers (e.g., 96, 256, 384, 4096) indicate the number of filters or neurons in that layer.

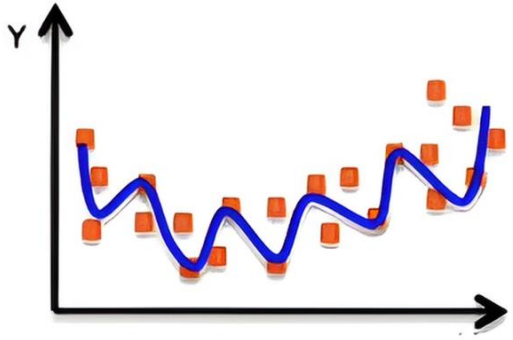


ResNet 2015

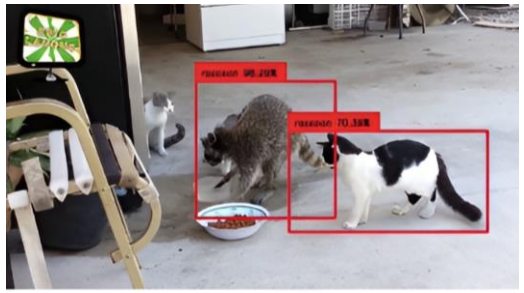
Common variants of the ResNet architecture include ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152. The number after "ResNet" typically denotes the number of layers in the network.

[cs231 Li et al., 2022]

Deep Learning Challenges



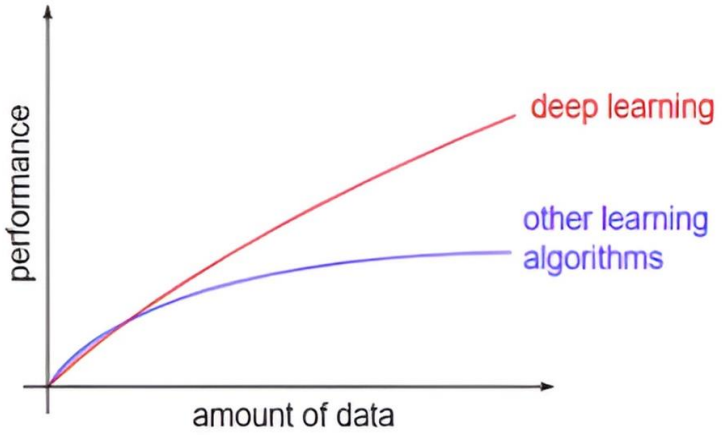
Overfitting
Too complex, extra parameters, does not generalize well



Evaluation satisfactory (e.g. insufficient accuracy)

Deep Learning Challenges

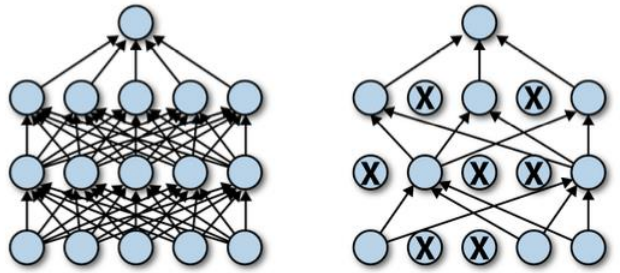
Computational resources (e.g. need GPU)

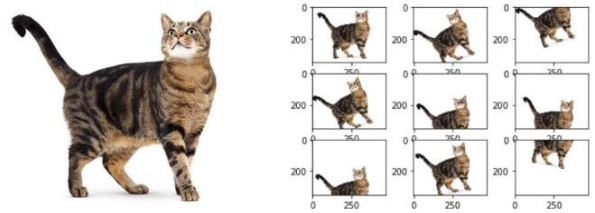


Data not sufficient (difficult to add more labeled data)

Resolving Overfitting Issue

- A model learns the detail and noise in the training data as concepts and generalizes that to new data predictions.
- Solutions:
  - **Drop-out** – In each forward pass, randomly set some neurons to zero
  - **Data augmentation** – Random translation, rotations, distortion, and crops to enlarge the dataset
  - **Early stopping** – Stop training before we have a chance to overfit
  - **Regularization** – add the regularization term to the loss

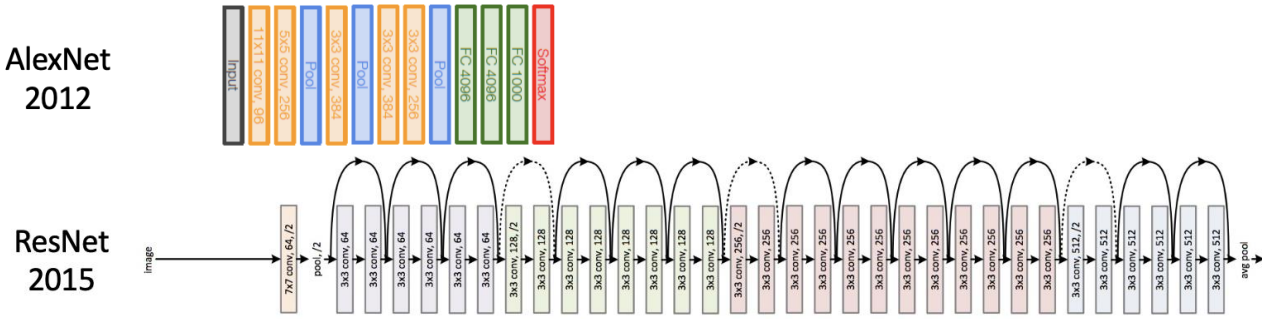


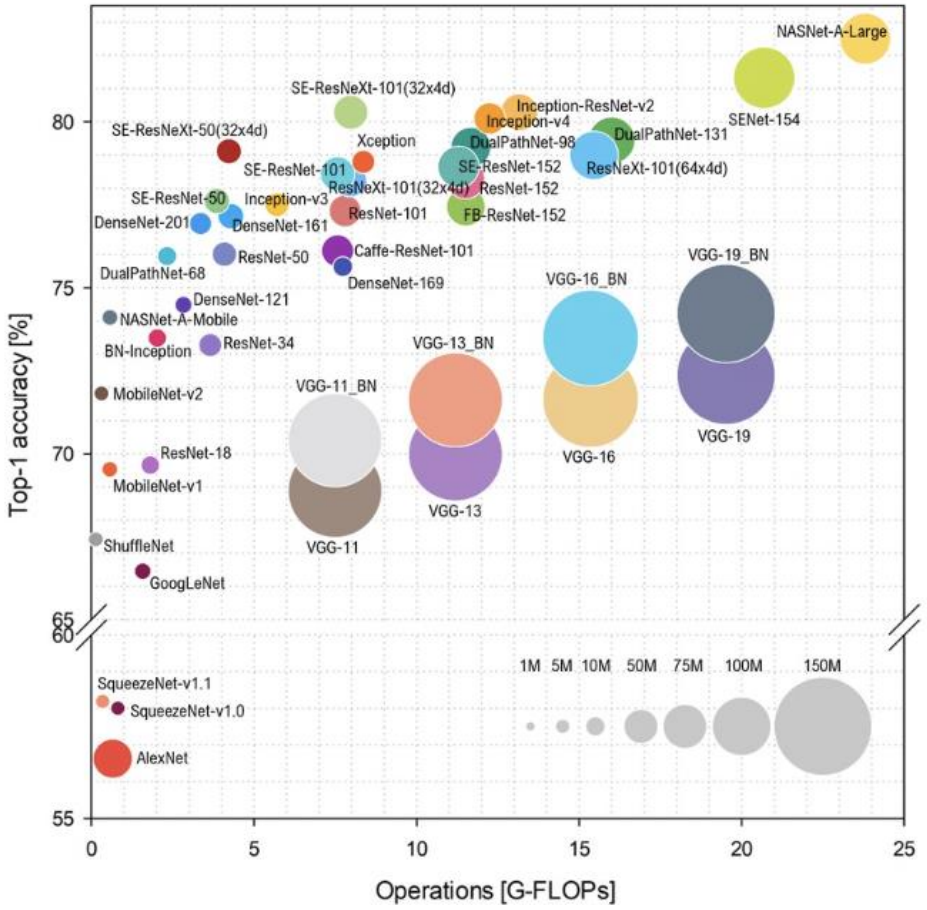


[cs231 Li et al., 2022]

Take advantage of Popular Models

- EX. **ImageNet** Benchmark – It is a standard test used to evaluate the performance of image recognition algorithms.
- ImageNet have more than 14 million images been hand-annotated by the project to indicate what objects are pictured and in at least one million of the images, bounding boxes are also provided.

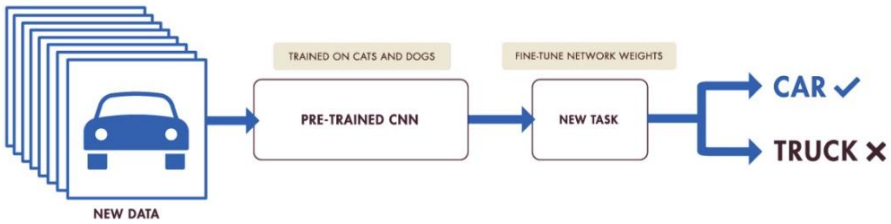




[cs231 Li et al., 2022]

Transfer Learning

- Transfer learning is commonly employed to **conserve time** and **resources** by avoiding the need to train several machine learning models from the ground up for tasks that are alike.
- Establishing a DNN from the beginning requires substantial data and computational power. If there is a moderate amount of data available, one can utilize a model that has already been trained on a vast dataset for a task that is akin to the desired application.

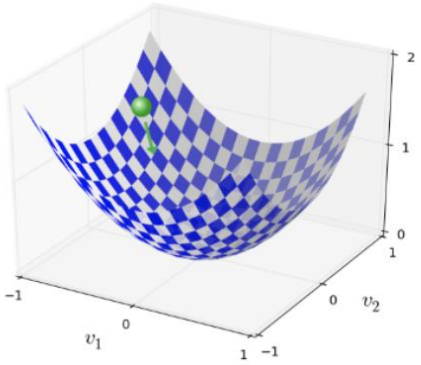


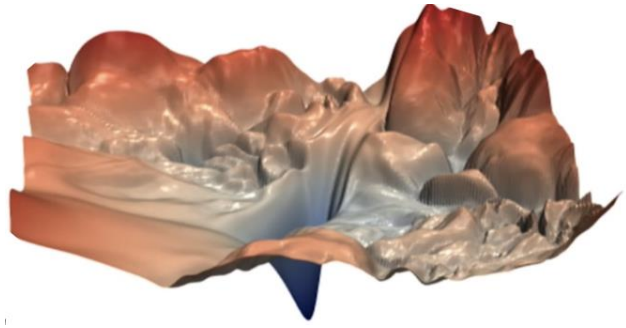
- Adjust and adapt the model to new data by employing one of the following methods:
  - **Full Model Training** – Utilize the structure of the pre-existing model and retrain it using your extensive dataset, which necessitates significant computational resources.
  - **Partial Layer Training** – Recognize that the initial layers capture universal characteristics while the deeper layers enhance specific details.
  - **Layer Freezing and Fine-Tuning** – Maintain the convolutional layers of the original model unchanged to serve as a constant mechanism for feature extraction and modify only the classifier layers. This approach is beneficial when computational resources or data are limited.

[Pingel, Nehemiah, 2017]

Convergence Difficulties

- Goal: find a set of parameters/weights w, to maximize the likelihood of correct output, given the input
- Gradient Decent (using the backpropagation algorithm to update weights)
  - What is the best learning rate?
    - Large: overshoots, is unstable and diverges
    - small: converges very slowly or gets stuck in false local minima
  - What if the loss function is complex?
- Solutions: hyperparameter optimization, adaptive learning (**SGD**, **ADAM**, Adagrad, etc.)



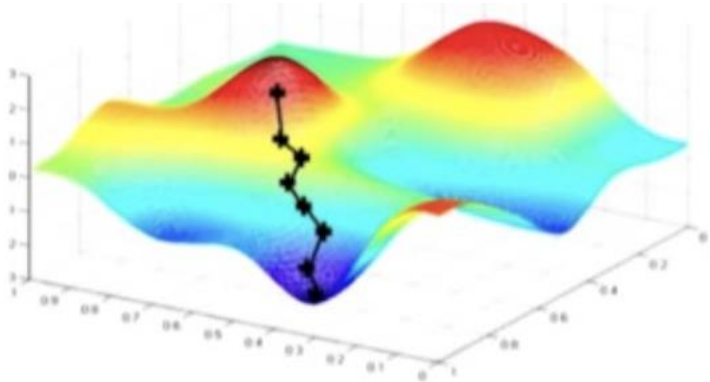


[Amini, "MIT 6.S191", 2023]

Stochastic Gradient Decent and Mini Batch

- Instead of computing for all training samples, randomly pick a small subset (mini-batch) of training samples at each step of gradient descent.
- SGD algorithm:

1. Initialize weights randomly $\sim \mathbb{N}(0, \sigma^2)$
2. Loop until convergence:
   - Pick batch of B data points
   - Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B}\sum_{k=1}^{B}\frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$
   - Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
3. Return weights



Benefits:

- More accurate estimation of the gradient
- Smoother convergence
  Allows larger learning rates
  Fast training (parallelize computation on GPUs)

- Compared to gradient descent, **SGD** takes more steps to converge, but each step is much faster

Computing Resources

- Graphic processing units (GPUs) $$
- Server or Cloud-based compute solutions (HPC, Google Colab, AWS)

Small scale

Graphics processing unit (GPU)

Large scale

High Performance Computing (HPC)

Next

- More about DNN & Autoencoders


thank you