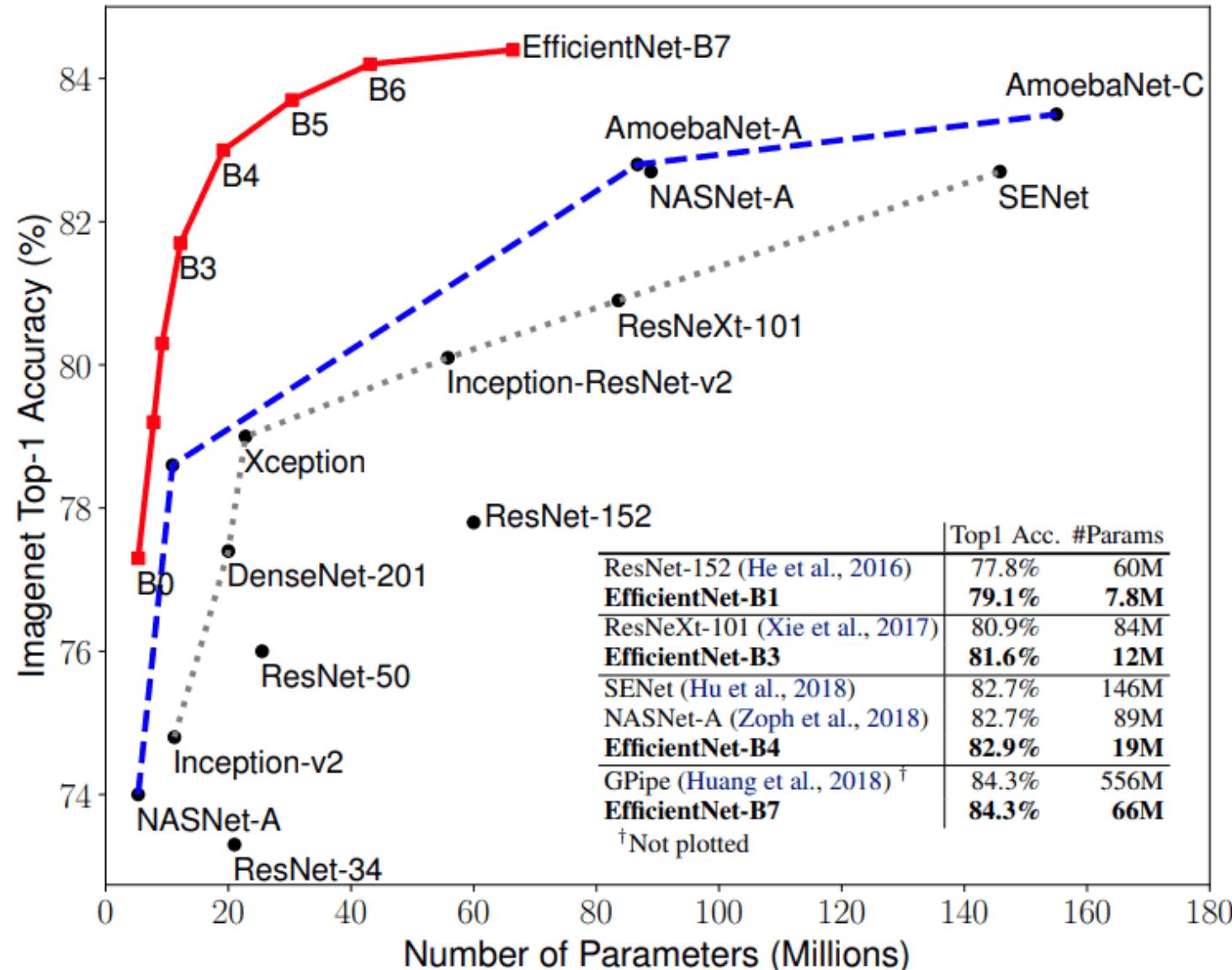


Day 4: Introduction to Neural Networks

Sergei V. Kalinin

What are Neural Nets?

- **Neural Networks are “glorified function fitters”**
 - Suppose, you have an input set (X) and the corresponding outputs (y)
 - The neural nets approximate the function that maps X to y , $f: X \rightarrow y$
- **How do they do that?**
 - They have trainable parameters ranging from 100s to billions (175 billion in ChatGPT-3)
 - Given X and y , we tune the parameters such that the function approximation gets better
- **Terms we need to know to understand how neural net works:**
 - Input-output pairs and how they are constructed
 - Neural network and it's architecture
 - Loss function to evaluate our predictions
 - Optimization algorithm (tuning) and its hyper-parameters
 - Explainability



Applications: Image classification



Dog



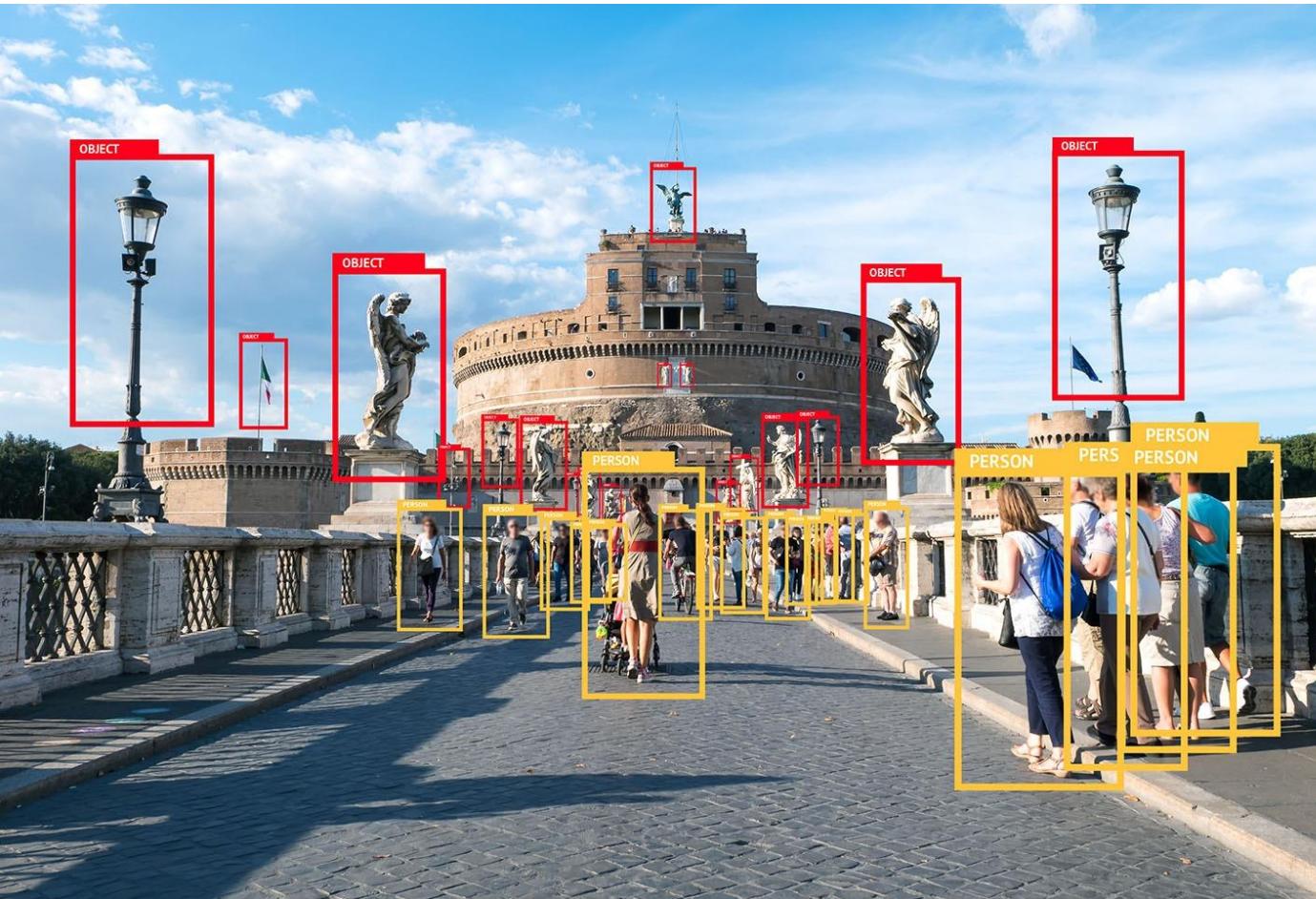
Dog



Cat



Applications: Object Detection



<https://www.augmentedstartups.com/blog/how-to-implement-object-detection-using-deep-learning-a-step-by-step-guide>

Applications: Semantic segmentation



<https://youtu.be/e9bHTIYFwhg?t=9>

Semantic segmentation continued...

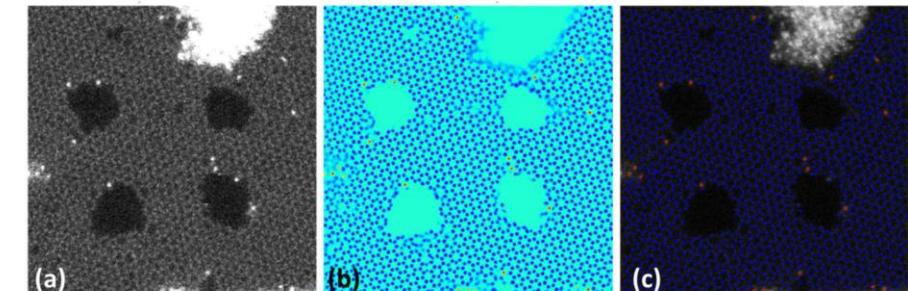
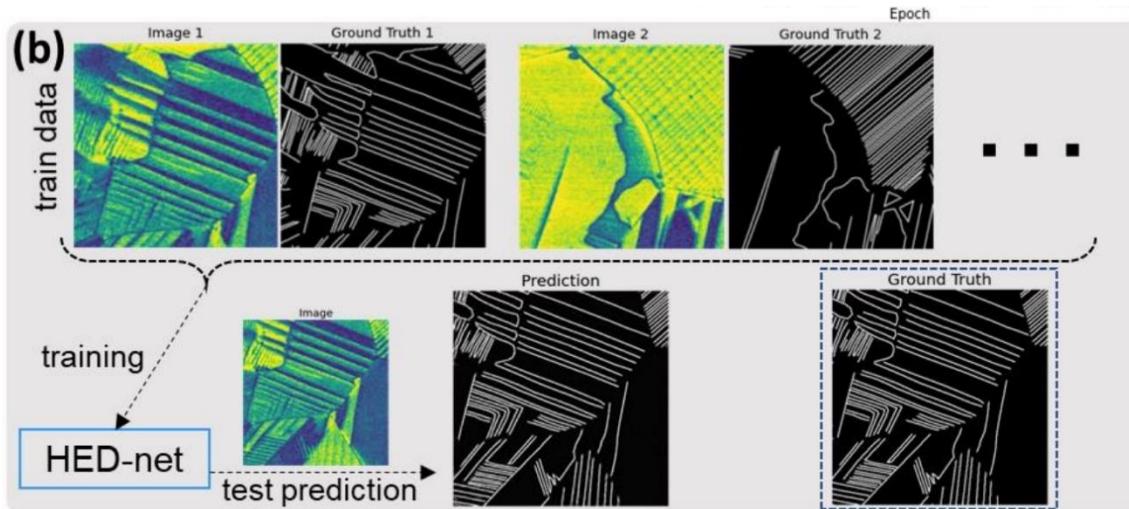


Figure 2. Experimental and deep learning-processed data. (a) Single image from dynamic STEM dataset corresponding to 10th frame. (b) DCNN analysis of image in (a) illustrating pixel probability density corresponding to carbon (blue) and silicon (red) atoms. (c) Atomic coordinates of C and Si. Note that amorphous region (same as holes) remain unrecognized by the DCNN. Image size is 16 nm.

Domain wall prediction from PFM images using ResHedNet, doi:

<https://doi.org/10.1002/adma.202103680>

Getting atomic coordinates from STEM images. Doi:

<https://doi.org/10.1126/sciadv.abd5084>

From the eyes of a Tesla



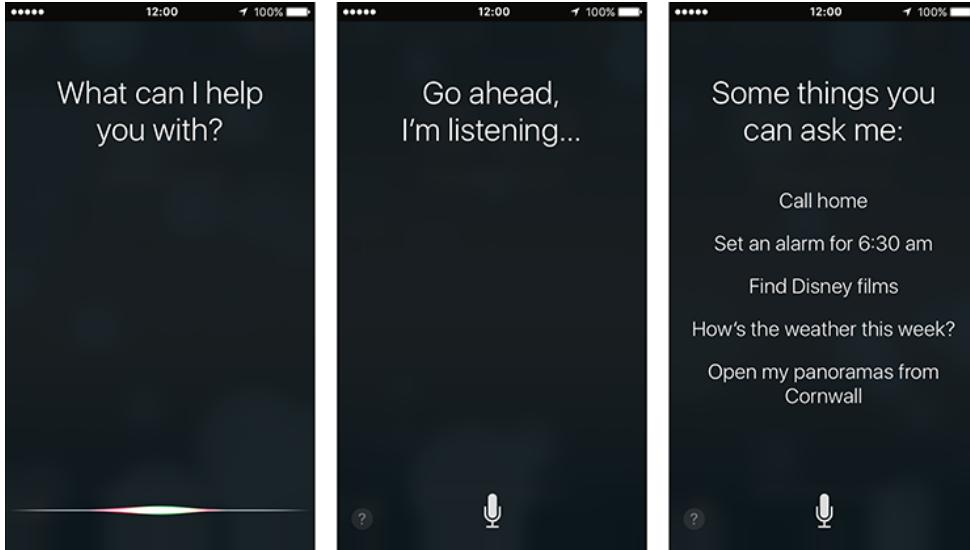
<https://www.youtube.com/watch?v=fKXztwtXaGo>

Applications: Activity recognition



https://youtu.be/qrzQ_AB1DZk?t=5

Applications: Language models



Hey Siri! Image from:
[https://accessibility.baesystems.com/
siri-iphoneipadipod-touch](https://accessibility.baesystems.com/siri-iphoneipadipod-touch)

You can find a plethora of other examples from various disciplines like science, robotics, art, medical imaging.

Advanced Data Analysis

VA Could you describe a bit about yourself inclunding the network architecture used and the number of learnable parameters that make you?

Certainly! I am based on the GPT (Generative Pre-trained Transformer) architecture, specifically a version known as GPT-4.

Network Architecture:

The GPT architecture is a type of transformer, a deep learning model that relies on self-attention mechanisms to process input data. Here's a broad overview of its components:

- **Input Embedding:** This layer converts input tokens (such as words in a sentence) into vectors of a fixed dimension.
- **Positional Encoding:** This is added to the input embedding to give the model information about the position of each token in the sequence.
- **Transformer Blocks:** Each block consists of multi-headed self-attention layers and feedforward neural networks. The self-attention mechanism allows the model to weigh input tokens differently, giving more importance to relevant tokens for a given task.
- **Normalization and Residual Connections:** These are used within each transformer block to help with training deep networks.
- **Output Head:** The final output of the transformer blocks is passed through a linear layer and a softmax function to produce a probability distribution over the possible output tokens.

Learnable Parameters:

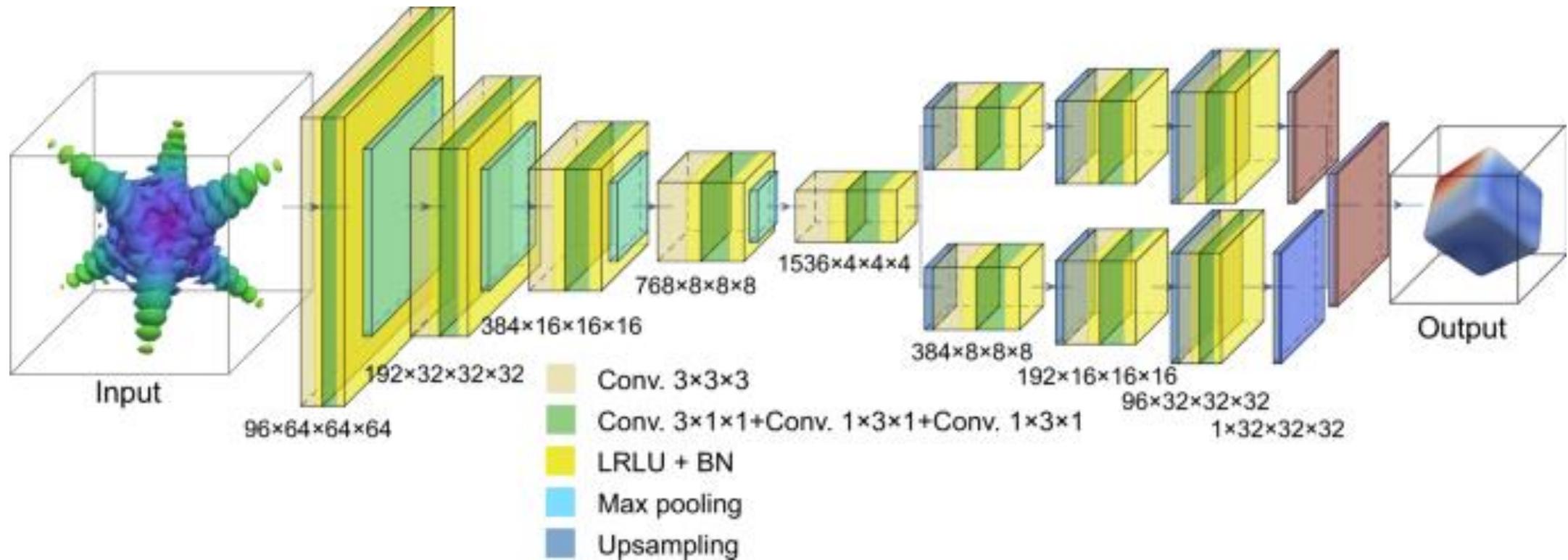
As for the number of learnable parameters, GPT-4 is reported to have around 175 billion parameters. These parameters are the weights and biases in the various layers of the neural network that are adjusted during the training process. The vast number of parameters allows GPT-4 to capture a wide array of patterns, nuances, and relationships within the data it has been trained on.

Regenerate

Send a message

ChatGPT can make mistakes. Consider checking important information.

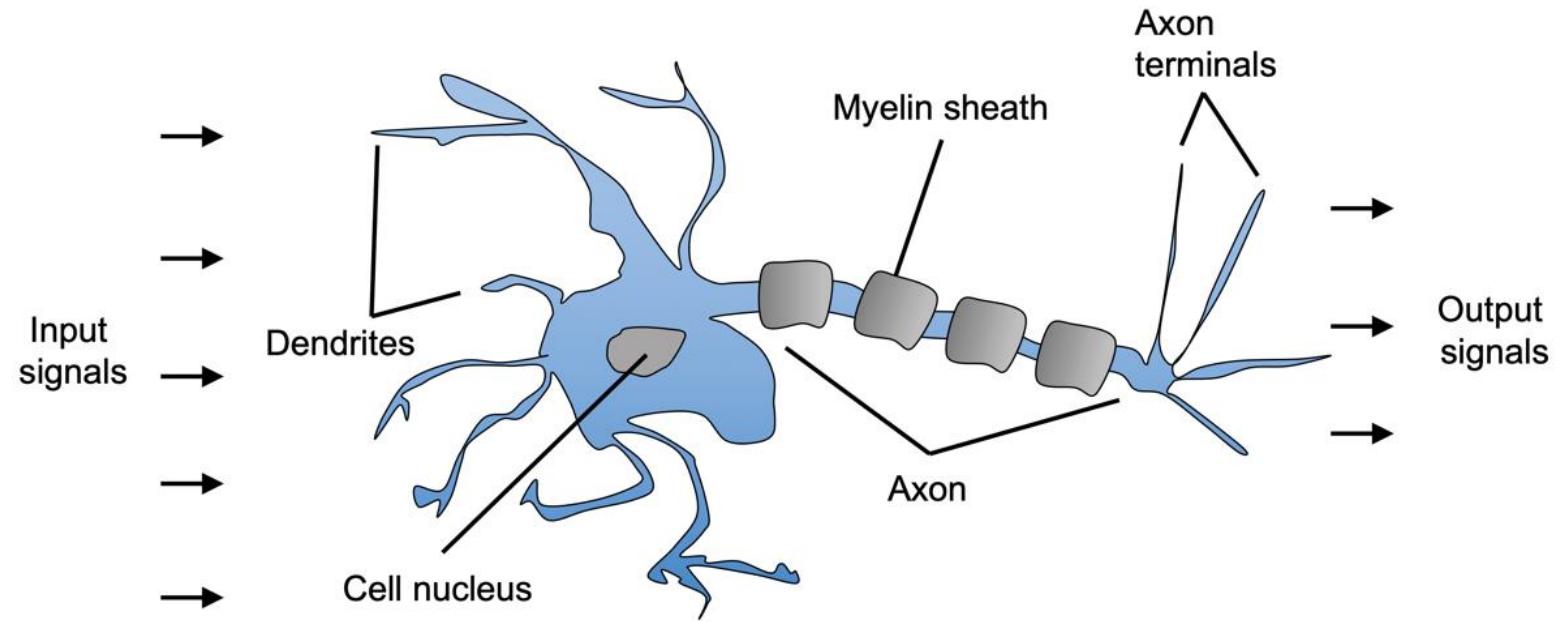
ChatGPT



Longlong Wu, Shinjae Yoo, Ana F. Suzana, Tadesse A. Assefa, Jiecheng Diao, Ross J. Harder, Wonsuk Cha & Ian K. Robinson, *Three-dimensional coherent X-ray diffraction imaging via deep convolutional neural networks*

<https://www.nature.com/articles/s41524-021-00644-z>

Brain structure and McCulloch-Pitts neuron



<https://www.kenhub.com/en/library/anatomy/histology-of-neurons>

A Logical Calculus of the Ideas Immanent in Nervous Activity by W. S. McCulloch and W. Pitts, *Bulletin of Mathematical Biophysics*, 5(4): 115-133, 1943).

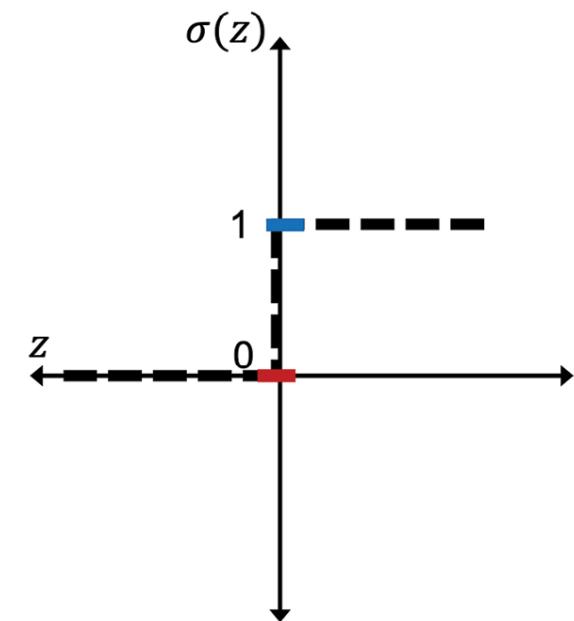
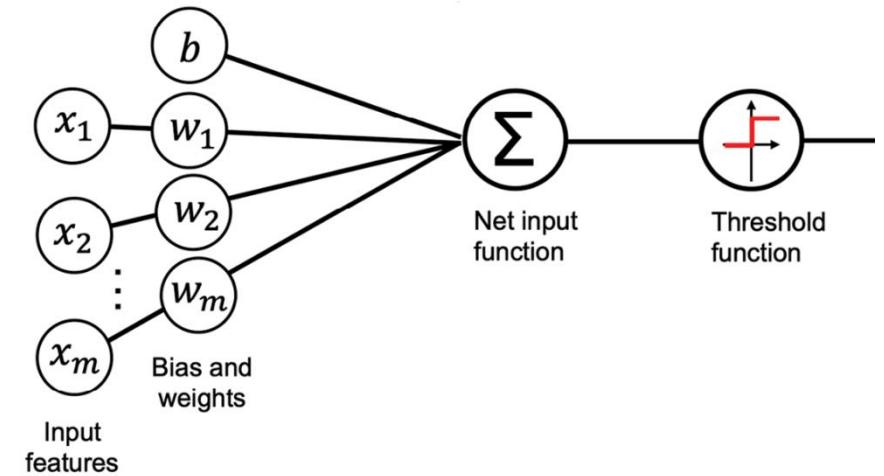
Building Linear Neuron

Input: $\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$

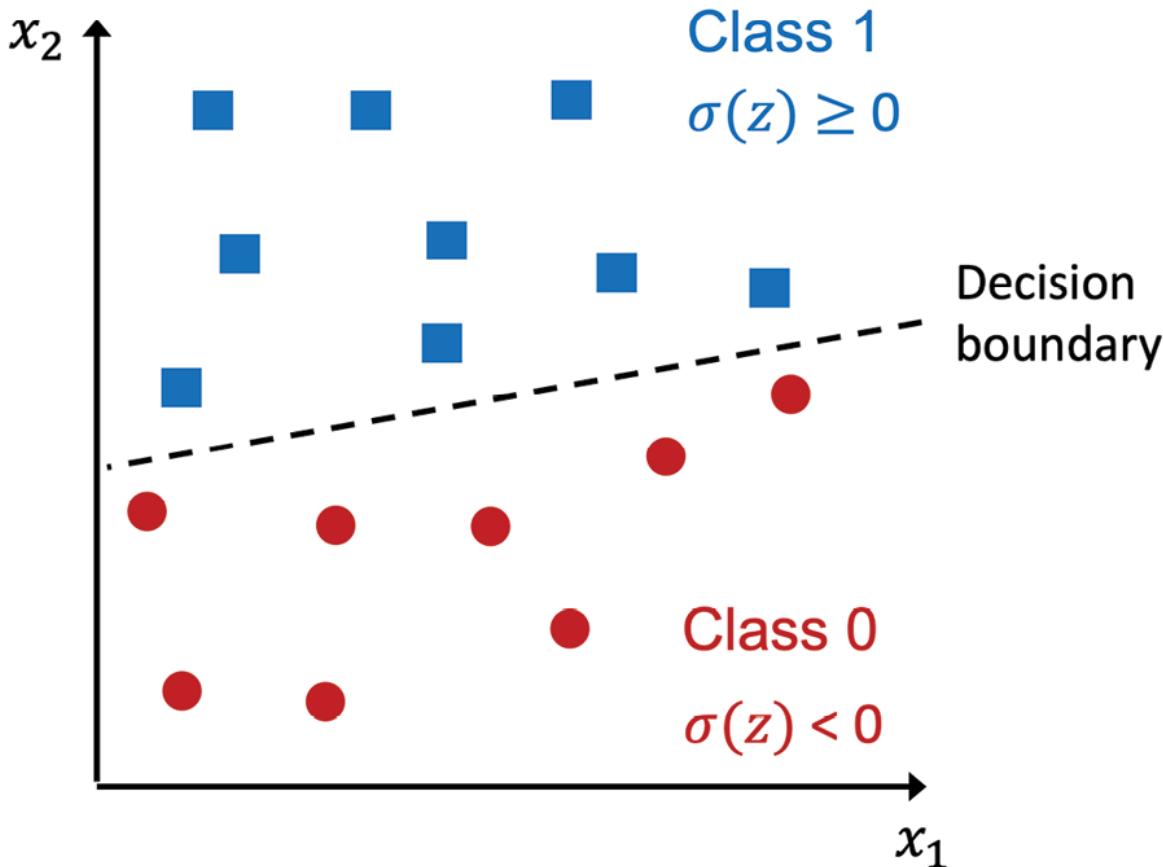
Weights: $\mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

Linear transform: $\mathbf{z} = w_1x_1 + \dots + w_mx_m + b = \mathbf{w}^T\mathbf{x} + b$

Output: $\sigma(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$



Linear Neuron in 2D



Linear transform:

$$z = w_1 x_1 + w_2 x_2 + b$$

Line:

$$x_2 = -w_1/w_2 x_1 - b/w_2$$

Training Linear Neuron

- Initialize the weights and bias unit to 0 or small random numbers
- For each training example, $\mathbf{x}(i)$:
- Compute the output value, $y(i) = \mathbf{w}^T \mathbf{x}(i) + b$
- Update the weights and bias unit: $w_j := w_j + \Delta w_j$ and $b := b + \Delta b$
- Where $\Delta w_j = \eta(y^{(i)} - \hat{y}^{(i)})x_j^{(i)}$ and $\Delta b = \eta(y^{(i)} - \hat{y}^{(i)})$

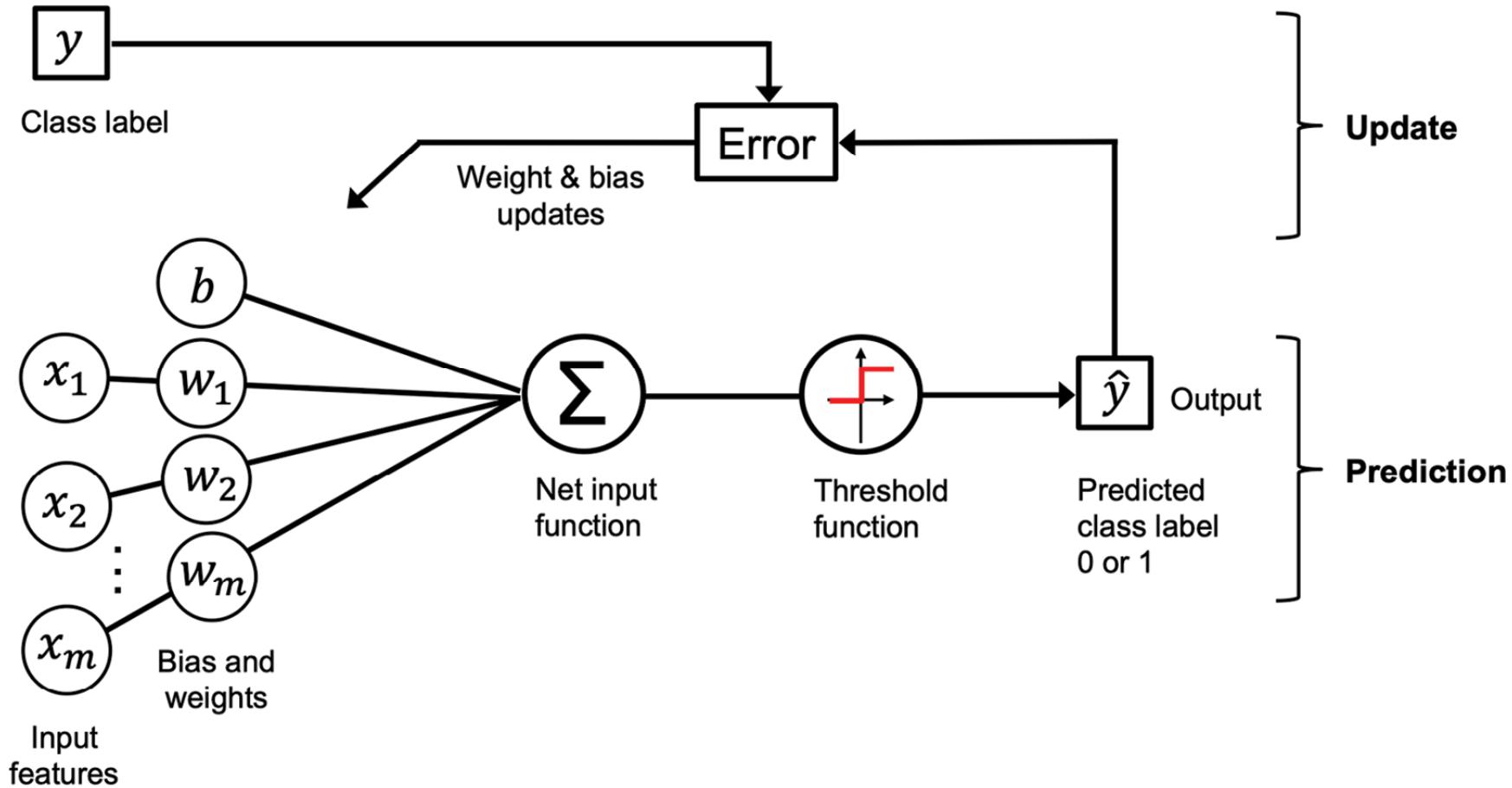
Each weight, w_j , corresponds to a feature, x_j , in the dataset,

η is the **learning rate** (typically a constant between 0.0 and 1.0),

$y^{(i)}$ is the **true class label** of the i th training example,

$\hat{y}^{(i)}$ is the **predicted class label**

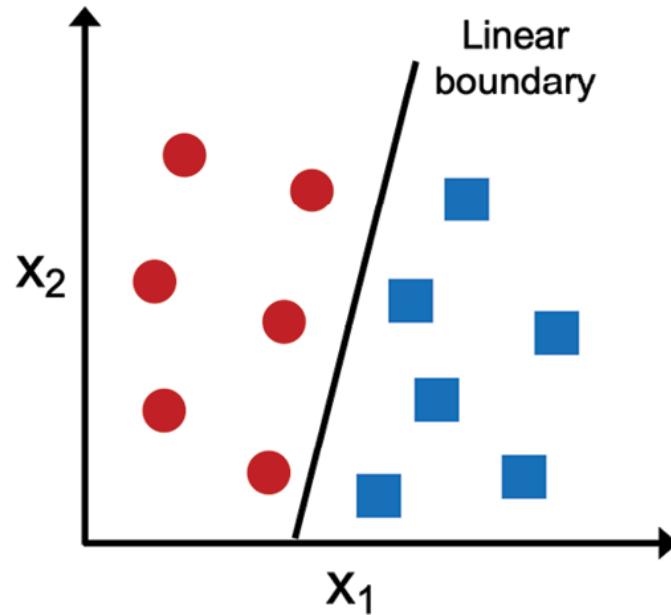
Training Linear Neuron



What problems can perceptron solve?

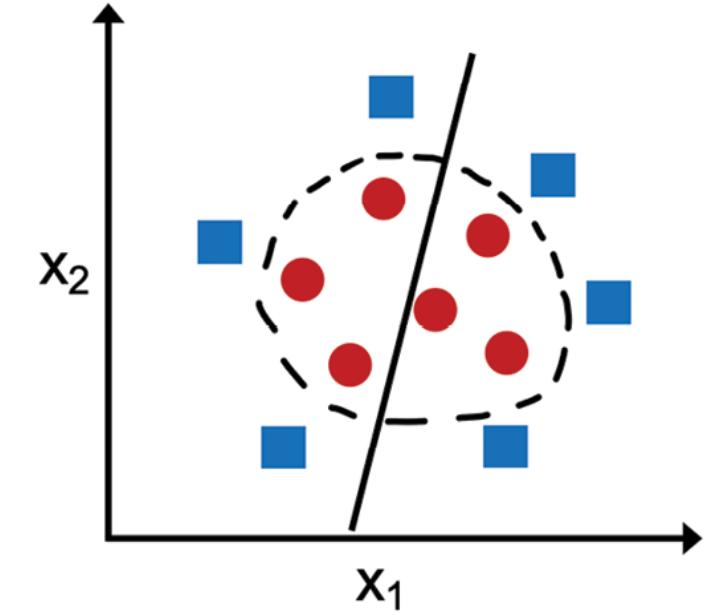
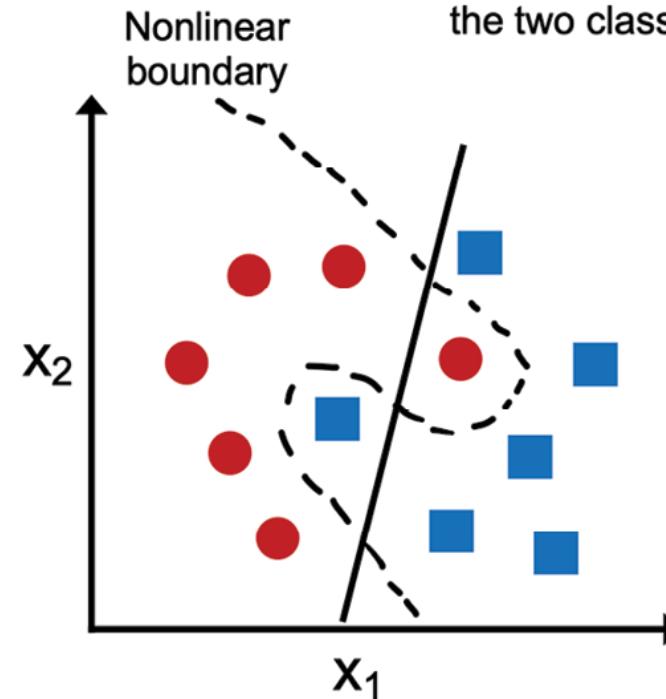
Linearly separable

A linear decision boundary that separates the two classes exists

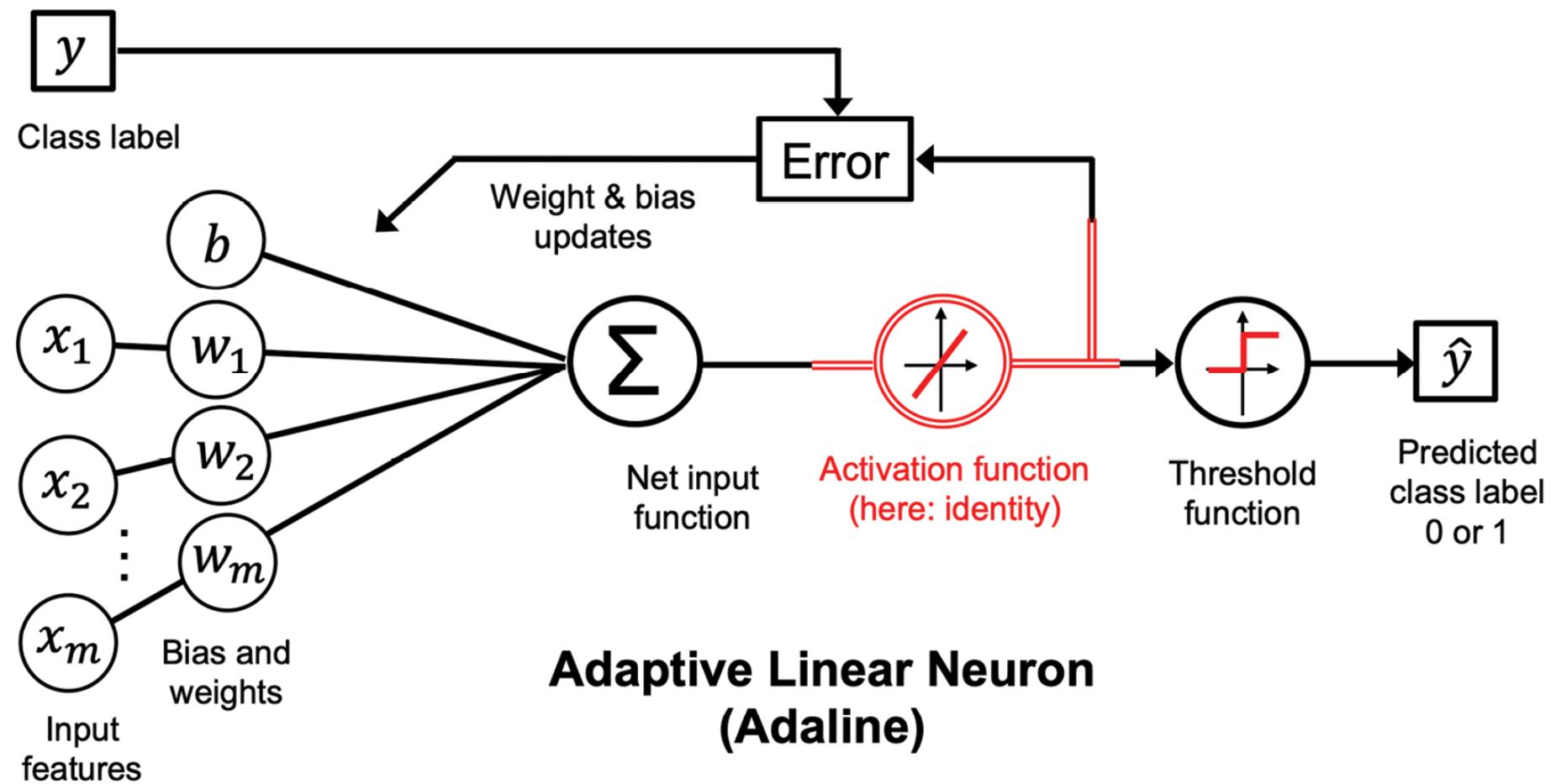


Not linearly separable

No linear decision boundary that separates the two classes perfectly exists



Adaline



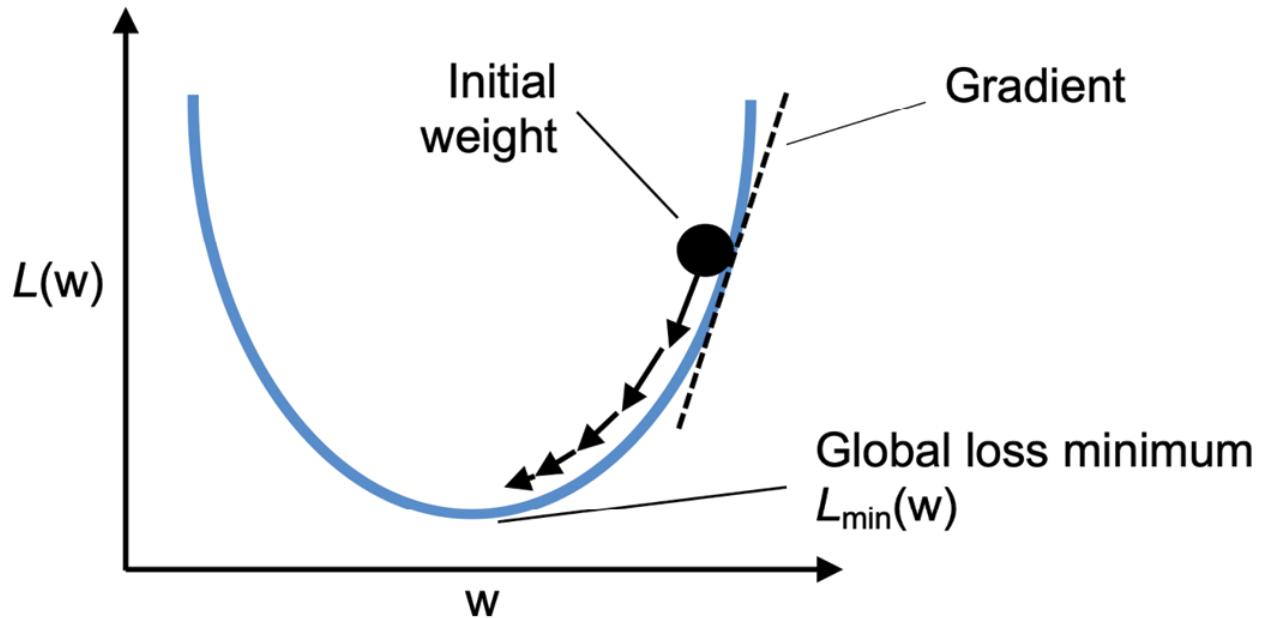
Adaline training

Loss function:

$$L(\mathbf{w}, b) = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \sigma(z^{(i)}))^2$$

Weights update:

$$\mathbf{w} := \mathbf{w} + \Delta \mathbf{w}, \quad b := b + \Delta b$$



Learning rule:

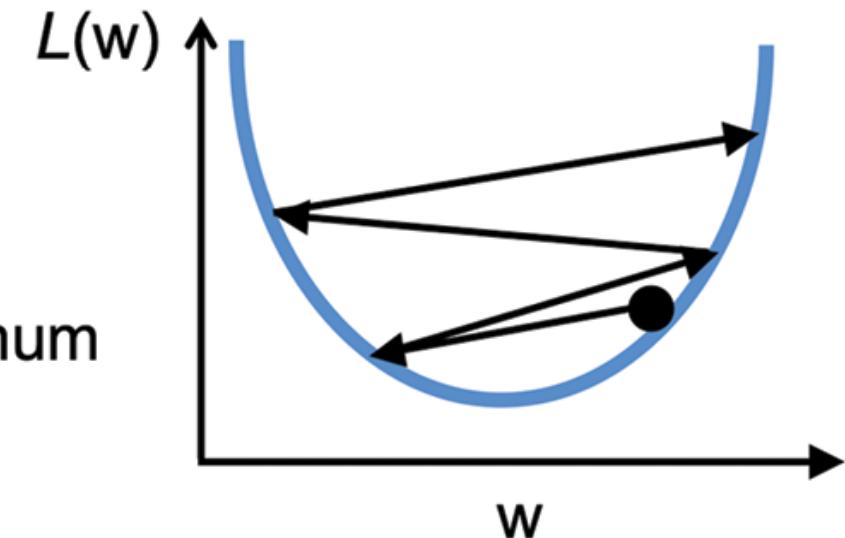
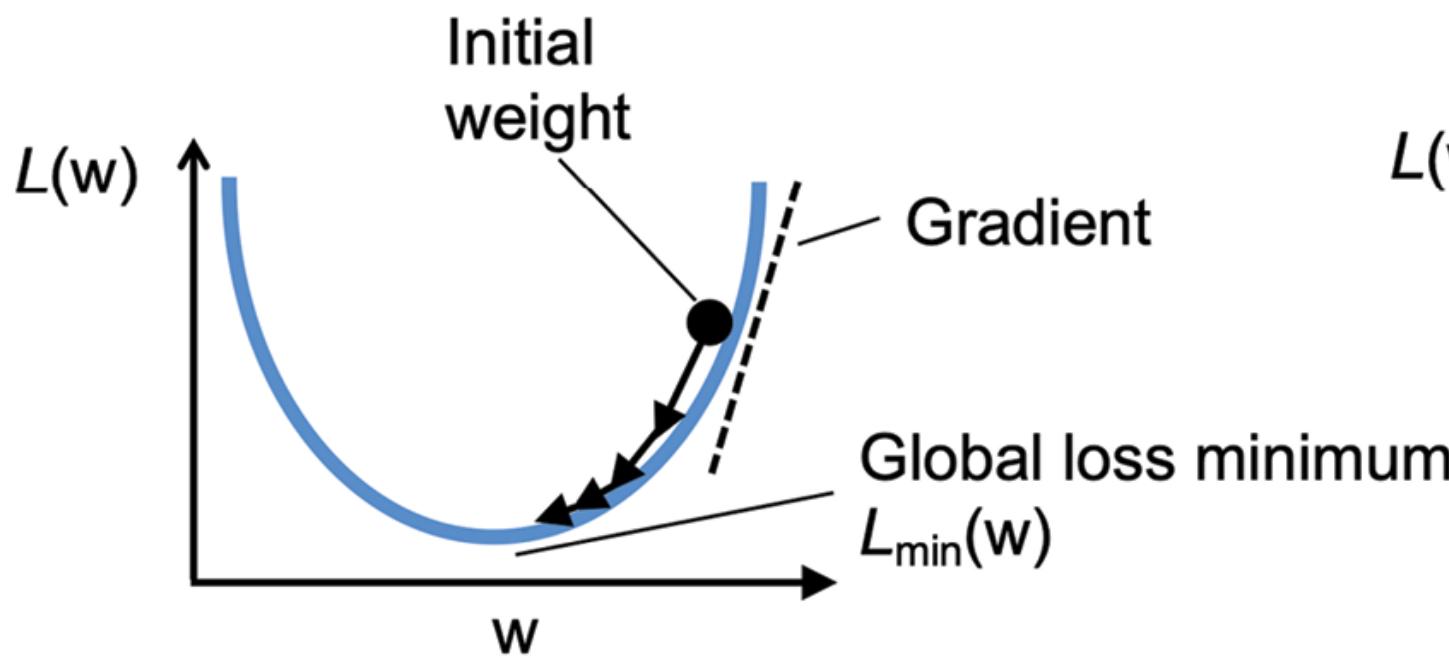
$$\Delta \mathbf{w} = -\eta \nabla_w L(\mathbf{w}, b), \quad \Delta b = -\eta \nabla_b L(\mathbf{w}, b)$$

How does it look like?

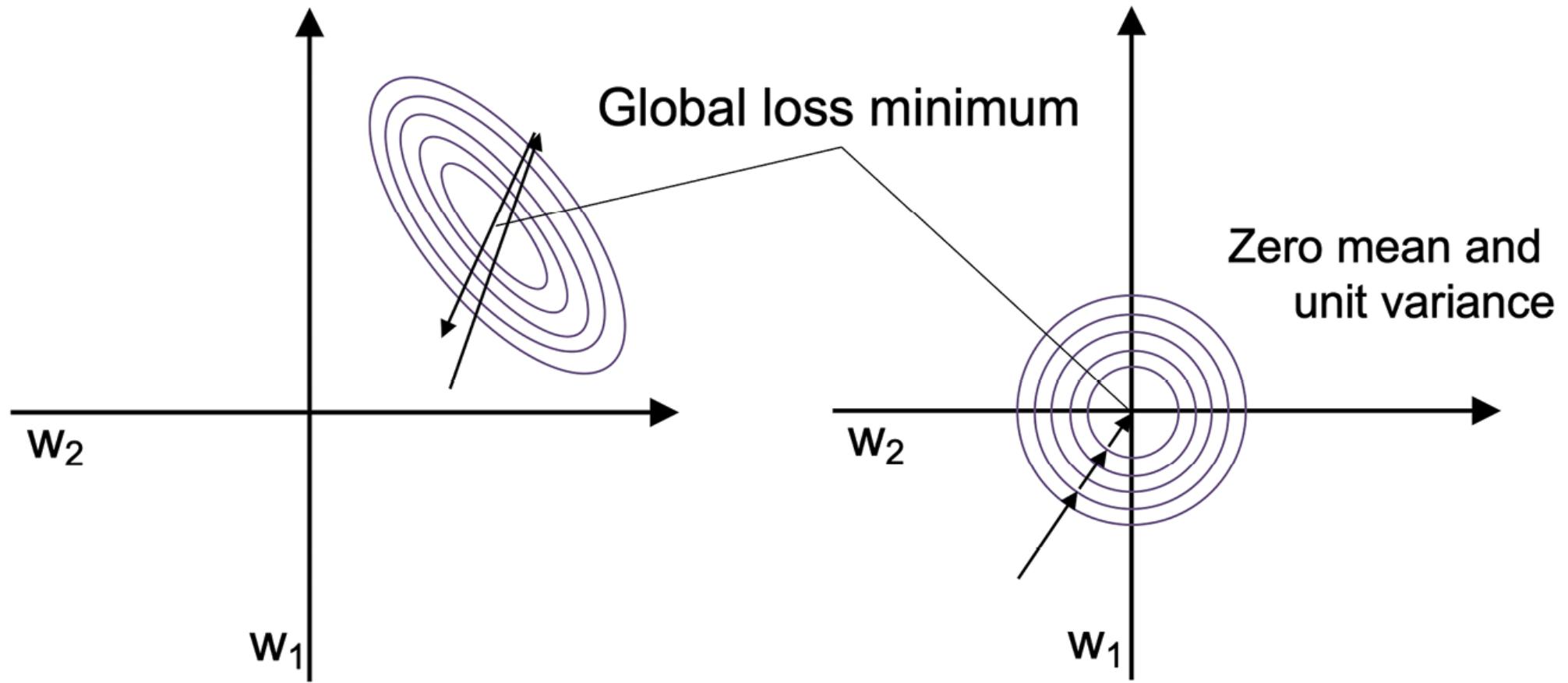
$$\begin{aligned}\frac{\partial L}{\partial w_j} &= \frac{\partial}{\partial w_j} \frac{1}{n} \sum_i (y^{(i)} - \sigma(z^{(i)}))^2 = \frac{1}{n} \frac{\partial}{\partial w_j} \sum_i (y^{(i)} - \sigma(z^{(i)}))^2 \\ &= \frac{2}{n} \sum_i (y^{(i)} - \sigma(z^{(i)})) \frac{\partial}{\partial w_j} (y^{(i)} - \sigma(z^{(i)})) \\ &= \frac{2}{n} \sum_i (y^{(i)} - \sigma(z^{(i)})) \frac{\partial}{\partial w_j} \left(y^{(i)} - \sum_j (w_j x_j^{(i)} + b) \right) \\ &= \frac{2}{n} \sum_i (y^{(i)} - \sigma(z^{(i)})) (-x_j^{(i)}) = -\frac{2}{n} \sum_i (y^{(i)} - \sigma(z^{(i)})) x_j^{(i)}\end{aligned}$$

- Adaline learning rule looks identical to the perceptron rule
- However, $\sigma(z^{(i)})$ where $z^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)} + b$ is a real number and not an integer class label.
- Also, weight update is calculated based on all examples in the training dataset (instead of updating the parameters incrementally after each training example): **batch gradient descent**.

Role of learning rate

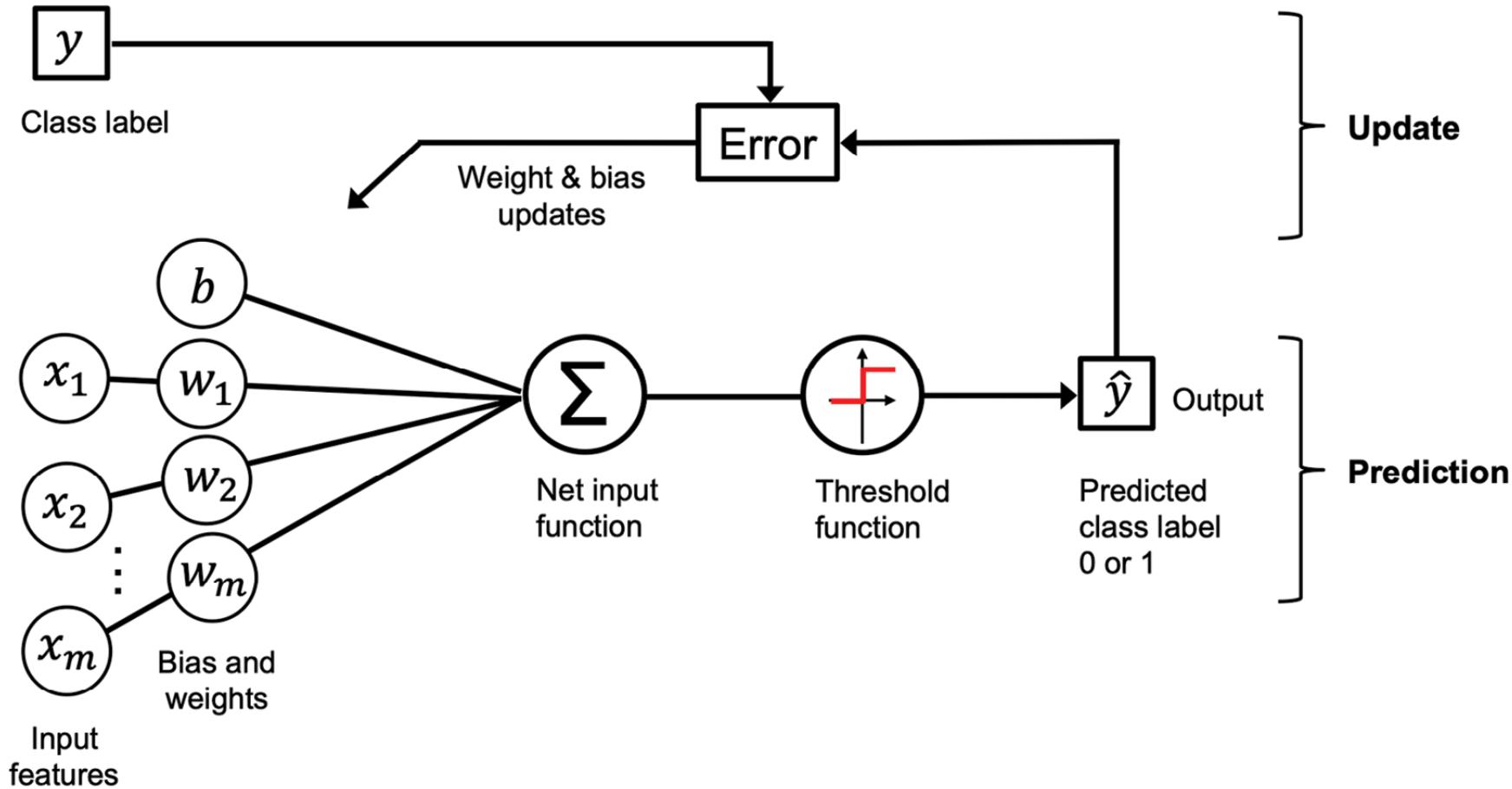


Why do we standardize input features?

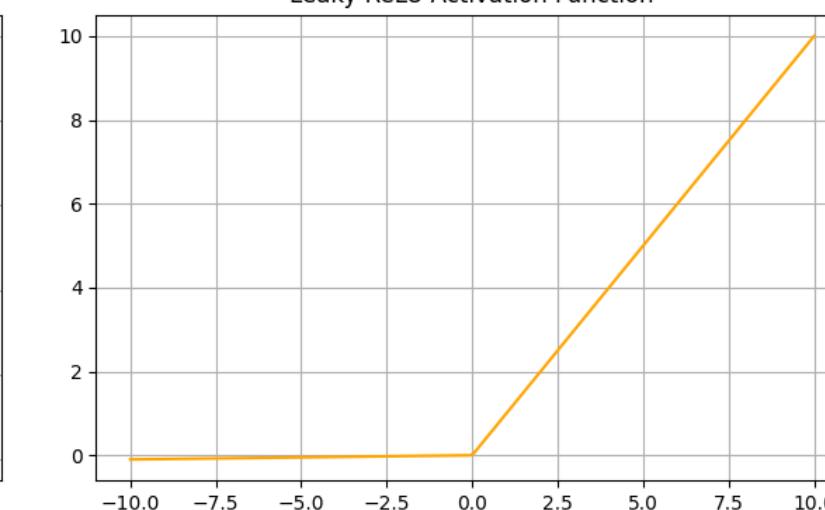
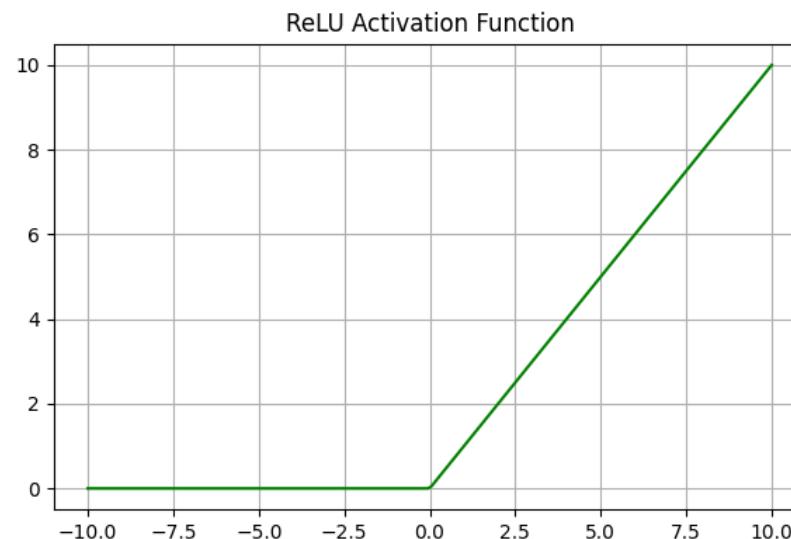
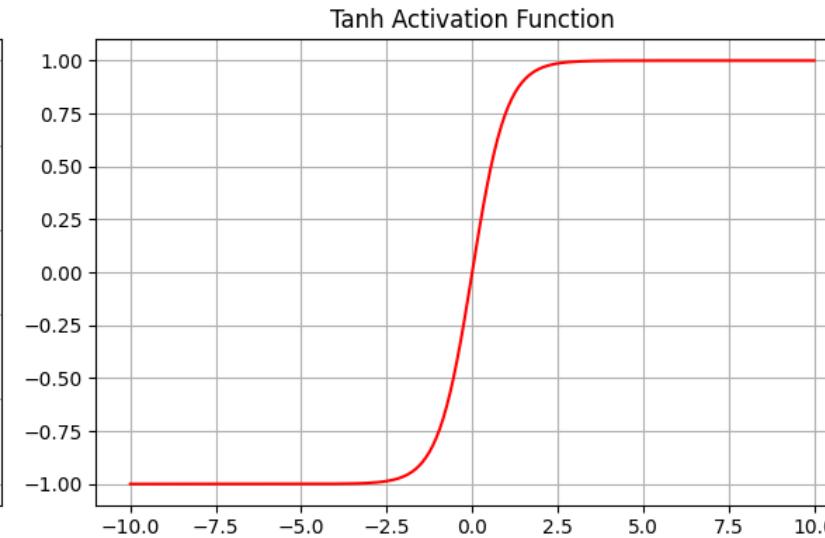
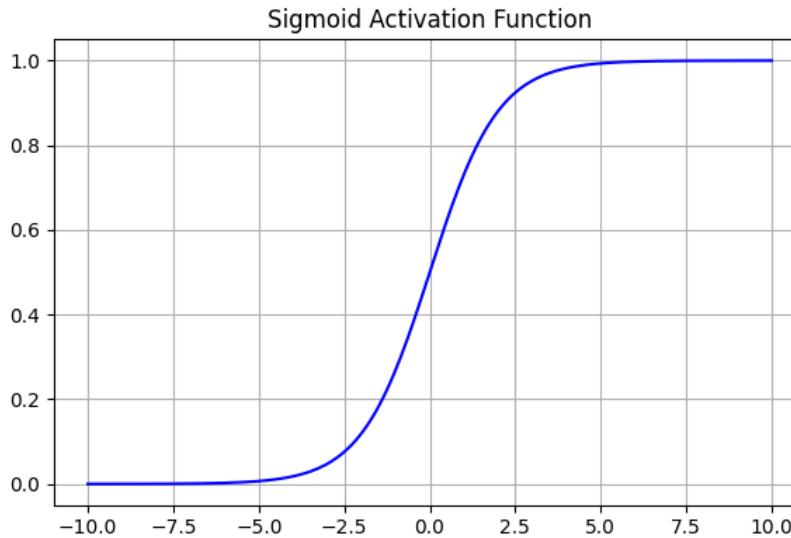


Colab: Perceptron

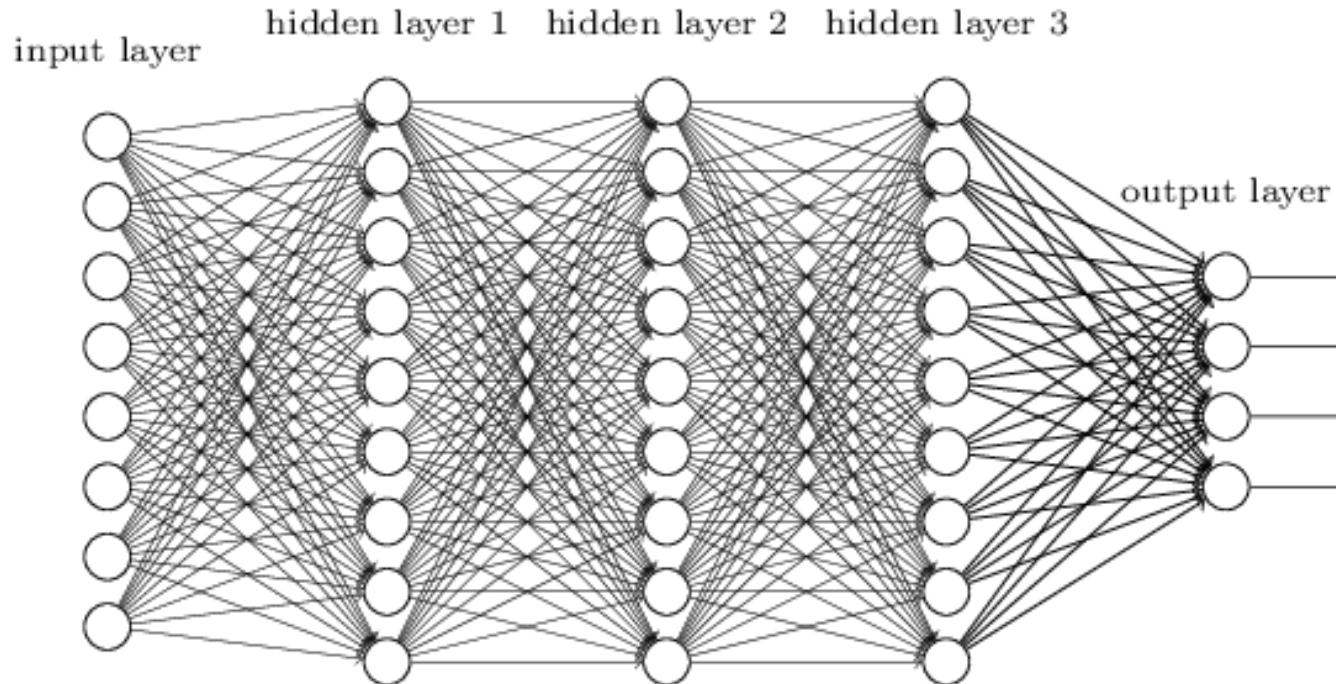
Training Linear Neuron



Activation functions



Putting Neurons Together



- Composed of multiple layers of artificial neurons.
- Each layer processes inputs received, applies a transformation (weights, biases, activation function), and passes the output to the next layer.
- Training a DNN involves adjusting weights and biases using backpropagation and a chosen optimization algorithm.
- The deep architecture enable the network to learn complex and abstract patterns in data.

Loss functions for supervised ML

A loss function, also known as a cost function, quantifies the difference between the predicted values and the actual target values. It guides the training of neural networks by providing a measure to minimize during optimization

Mean Squared Error (MSE): Used for regression problems.

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

Measures the average squared difference between actual and predicted values

Cross-Entropy Loss: Used for classification problems.

$$CE = - \sum_{i=1}^N y_i \log(\hat{y}_i)$$

Measures the performance of a classification model whose output is a probability value between 0 and 1

- Loss functions provide the primary feedback signal for learning.
- The choice of loss function can significantly affect the model's performance and convergence

Backpropagation

Backpropagation is a mechanism used to update the weights in a neural network efficiently, based on the error rate obtained in the previous epoch (i.e., iteration). It effectively distributes the error back through the network layers

- **Forward Pass:** Calculating the predicted output, moving the input data through the network layers
- **Loss Function:** Determining the error by comparing the predicted output to the actual output
- **Backward Pass:** Computing the gradient of the loss function with respect to each weight by the chain rule
- **Weight Update:** Adjusting the weights of the network in a direction that minimally reduces the loss (gradient descent)

Input Data → Forward Pass → Calculate Loss → Backward Pass → Update Weights

<https://medium.com/@14prakash/back-propagation-is-very-simple-who-made-it-complicated-97b794c97e5c>

Metrics

Quantitative measures used to evaluate and monitor the performance of the network, both during training and after deployment. Here are some common metrics used in DCNNs:

Accuracy: Defined as the ratio of correctly predicted observations to the total observations.

Precision: The ratio of correctly predicted positive observations to the total predicted positive observations. Important in scenarios where the cost of a false positive is high.

Recall (Sensitivity): The ratio of correctly predicted positive observations to all observations in the actual class. Crucial in situations where missing a positive instance is costly.

F1 Score: The harmonic mean of precision and recall. Trade-off between precision and recall.

Mean Squared Error (MSE): Commonly used in regression tasks.

Area Under the ROC Curve (AUC-ROC): Represents the degree or measure of separability achieved by the model.

Intersection over Union (IoU): Typically used in object detection and segmentation tasks. Measures the overlap between the predicted bounding box and the ground truth box.

How is metrics different from loss function?

- **Optimization vs. Evaluation:** Loss functions are used to optimize the model (i.e., to adjust the model parameters during training), whereas metrics are used to evaluate the model's performance.
- **Differentiability:** Loss functions need to be differentiable with respect to model parameters, but metrics do not have this requirement.
- **Interpretability:** Metrics are often more interpretable than loss functions and are chosen based on what makes the most sense for the specific application or domain.
- **Role in Model Training:** The loss function directly influences the training process, while metrics are more about assessing the outcome of that training.

How to not get lost?

- A vast array of network architectures ranging from Multilayer Perceptrons (MLPs) to Graph Neural Networks and Transformers
- Each architecture has unique characteristics suited for different types of data and tasks, ways to define the architecture, and so on
- Numerous methods to engineer loss functions
- Numerous ways to implement regularization

Problem-Centric Approach:

- Always start with the problem at hand: Analyze the nature of input data and desired output
- Choose a network architecture that aligns with the type and structure of your data
- Select a loss function that reflects the objective of the problem
- Metrics should be chosen based on what measures success for your specific task

Hyperparameter Tuning:

- Once the architecture and loss function are set, proceed to tune hyperparameters including network structure, optimizers, regularization, etc.
- Hyperparameter tuning should be guided by the chosen metrics and the nature of the problem

Some useful resources:

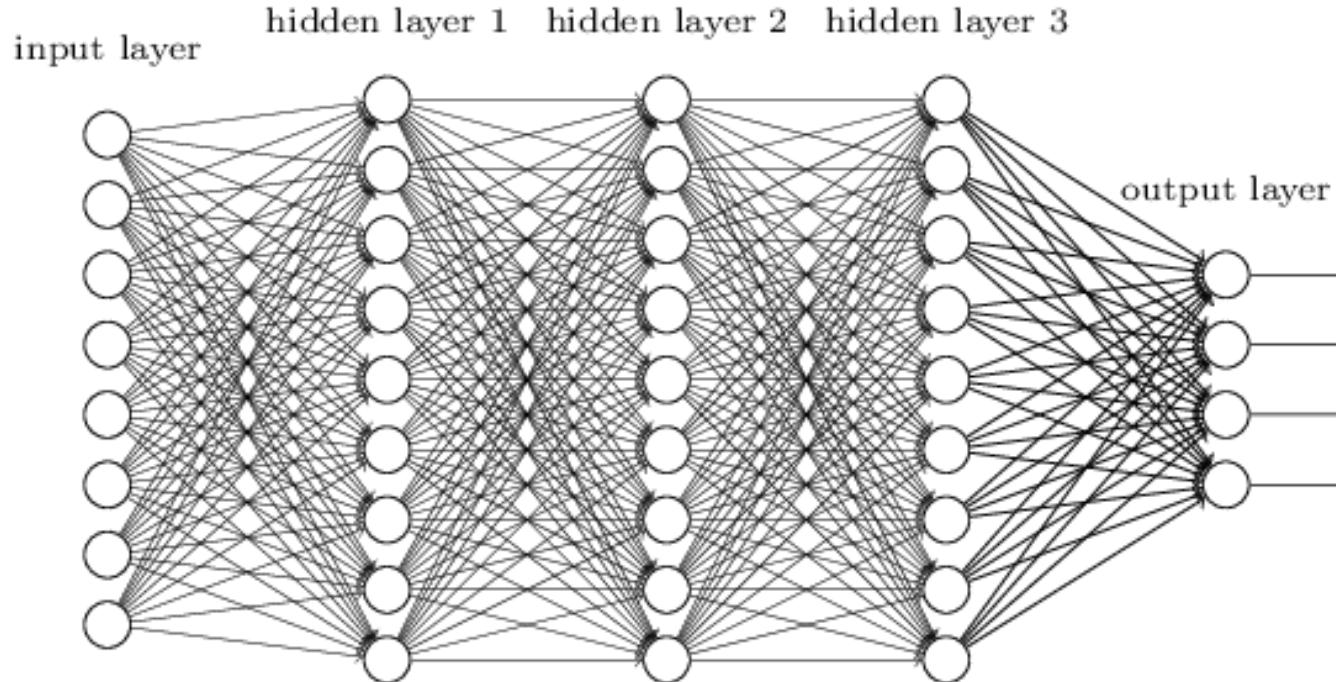
<https://udlbook.github.io/udlbook/>

<https://dmol.pub/ml/classification.html>

<https://keras.io/examples/>

Colab: DCNN 1

Putting Neurons Together



- Linear neurons and activation functions
- Input dimensionality
- Output dimensionality
- Special attention: activation function of the last layer
- Loss function – how different is what we get from what we want (supervised ML)
- Backpropagation – how we adapt the neuron settings given the loss
- Metrics – how we monitor the training/performance

Loss functions for classifiers

- 1. Binary Cross-Entropy:** Measures the performance of a classification model whose output is a probability between 0 and 1. It's suitable for models where each instance is expected to belong to one of two classes (**Sigmoid**)
- 2. Categorical Cross-Entropy:** Used when there are two or more label classes. Assumes a single-label, multi-class classification problem where each class is mutually exclusive (**Softmax**)
- 3. Sparse Categorical Cross-Entropy:** Similar to CCE but more efficient with a large number of classes. Used when the classes are mutually exclusive and the labels are integers (**Softmax**)
- 4. Hinge Loss:** Commonly used with Support Vector Machine (SVM) classifiers but applicable to neural networks for binary classification tasks. It's designed to create a margin between data points (**Linear or Tanh**)
- 5. Focal Loss:** An extension of cross-entropy loss, adding a focusing parameter to balance the importance of correctly classifying different classes, particularly useful in imbalanced datasets.
Sigmoid (Binary), **Softmax** (Multi-class)
- 6. Kullback-Leibler Divergence:** Measures how one probability distribution diverges from a second, expected probability distribution. Used in scenarios like learning a complex multi-class classification problem or training a model to output a probability distribution (**Softmax**)

Loss functions for regressors

1. Mean Squared Error (MSE): Standard Regression

Calculates the average squared difference between the estimated values and the actual value. Commonly used in regression problems and is sensitive to outliers (Linear).

2. Mean Absolute Error (MAE): Standard Regression

Computes the average of the absolute differences between predictions and actual values. Less sensitive to outliers compared to MSE (Linear)

3. Huber Loss: Robust Regression

Less sensitive to outliers than the MSE. It's quadratic for small errors and linear for large errors, combining the best properties of MSE and MAE (Linear)

4. Quantile Loss: Quantile Regression

Used in scenarios where the goal is to predict an interval instead of only point estimates. It allows the model to learn to predict a range that contains the true value with a certain probability (Linear).

7. Poisson Loss: Count Data Regression

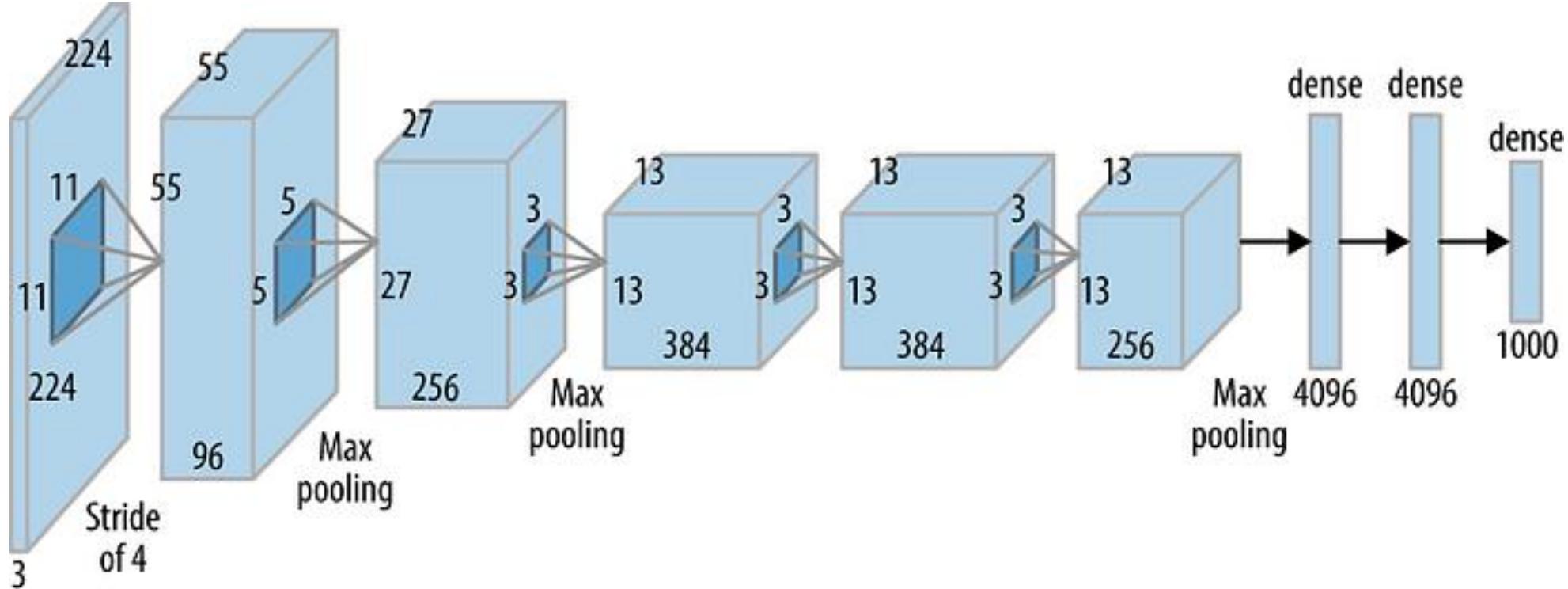
Suitable for modeling count data and intensity rates where the output values are non-negative integers (Exponential)

Loss functions for image segmentation

- 1. Binary Cross-Entropy:** Measures the pixel-wise binary classification loss. Commonly used in binary segmentation tasks (e.g., object vs. background). Ideal for problems where each pixel is classified independently into two classes.
- 2. Categorical Cross-Entropy:** An extension of binary cross-entropy for multi-class segmentation tasks. It calculates the loss for each pixel and classifies it into one of the multiple classes. Used in scenarios with multiple classes (e.g., different types of objects in an image).
- 3. Dice Loss:** Based on the Dice coefficient, which is a measure of overlap between two samples. Particularly effective for data with imbalanced classes, such as medical image segmentation.
- 4. Jaccard/Intersection Over Union (IoU) Loss.** Based on Jaccard index. Useful in evaluating the similarity between the predicted segmentation and the ground truth, especially when the classes are imbalanced.
- 5. Weighted Cross-Entropy:** A variant of cross-entropy loss where each class is given a specific weight, usually inversely proportional to the class frequency. Useful in cases of class imbalance where certain classes need more focus.

Deep Convolutional Neural Networks

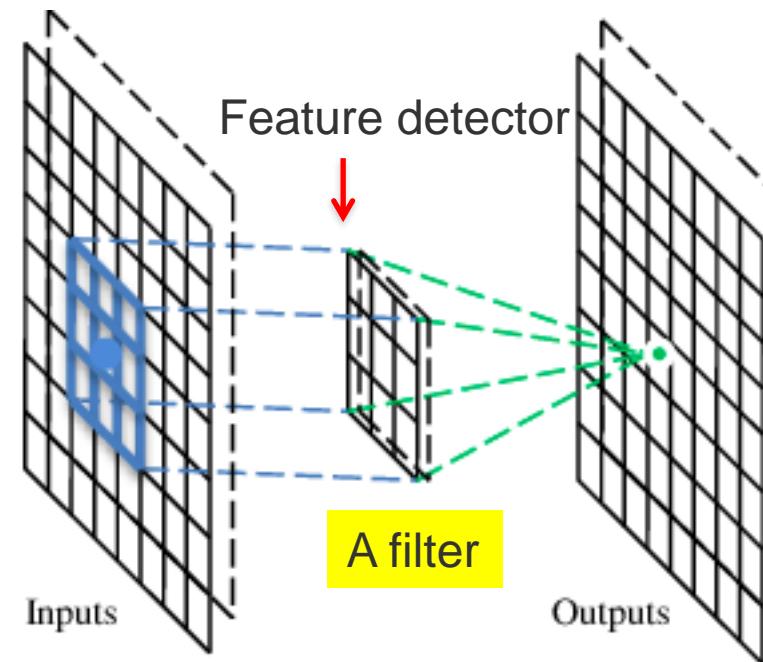
Structure of AlexNet



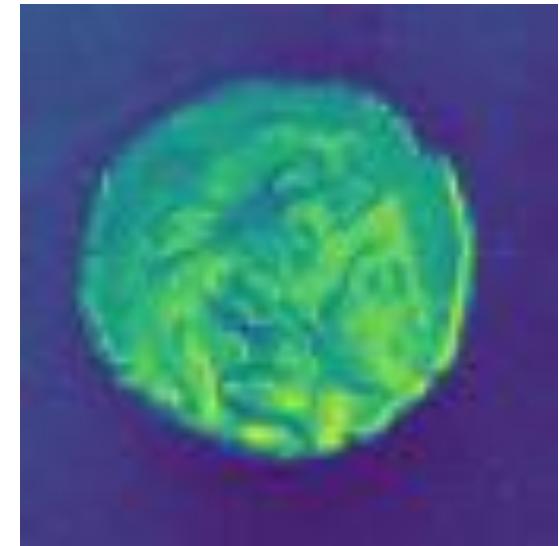
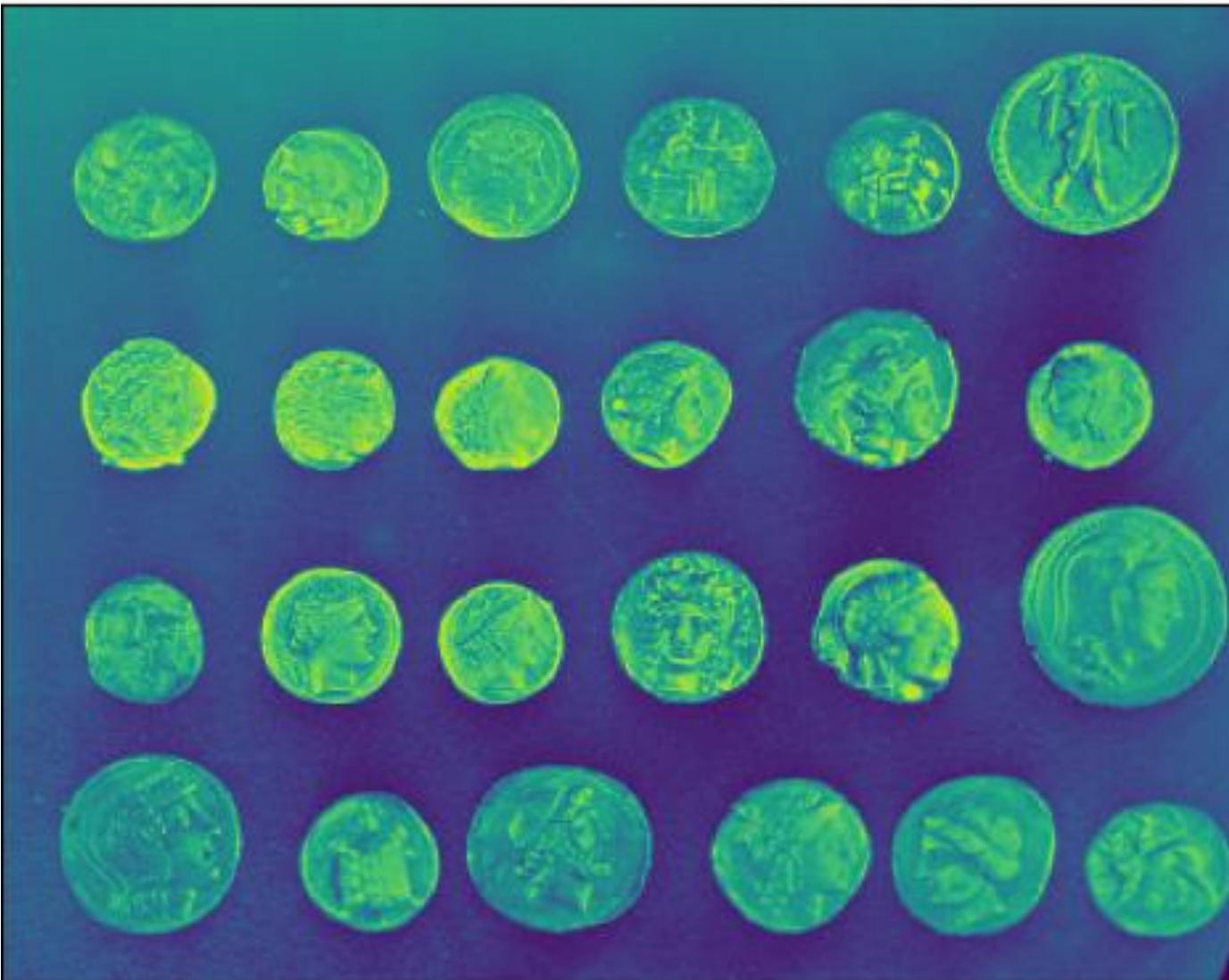
<https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecc96>

Finding features in images

A CNN is a neural network with some convolutional layers (and some other layers). A convolutional layer has a number of filters that does convolutional operation.

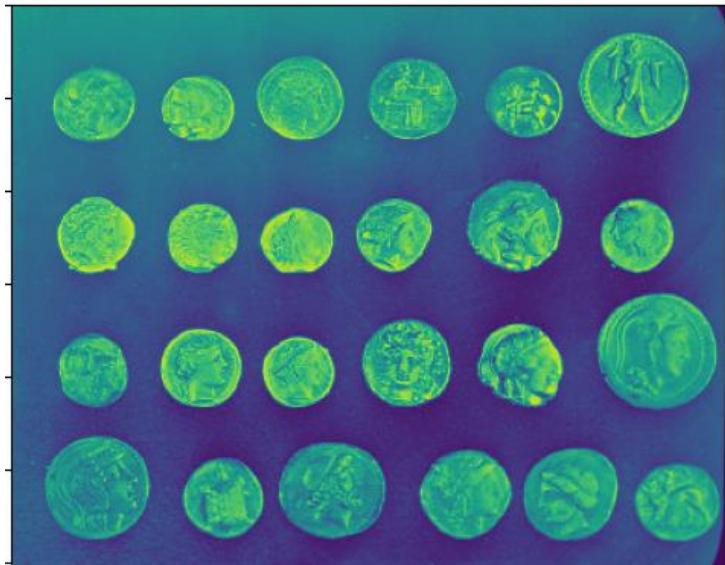


Coin example (scikit-image)

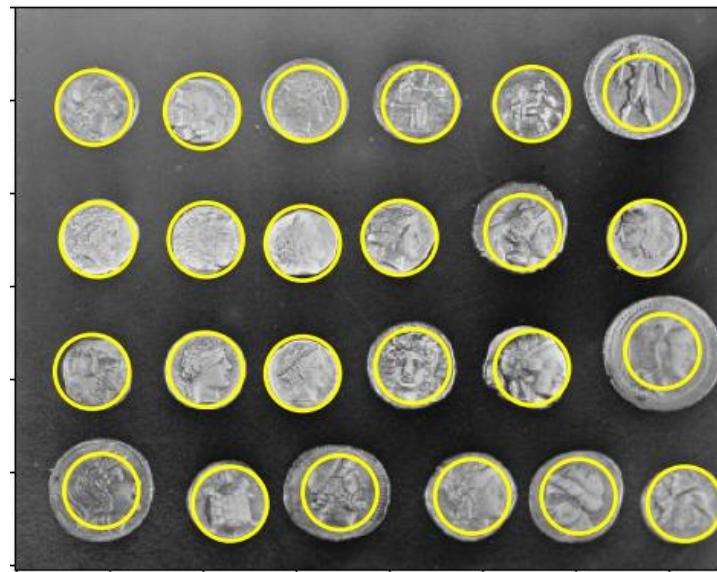


What are features in this image?

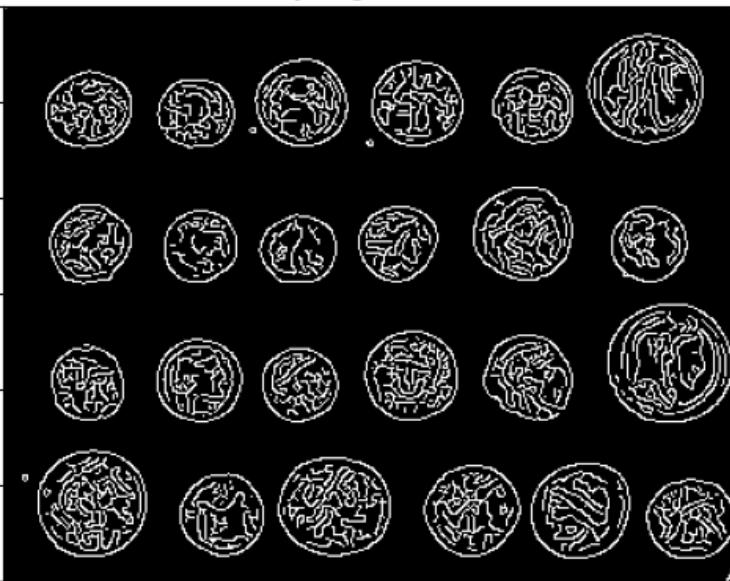
Feature finding in scikit-image



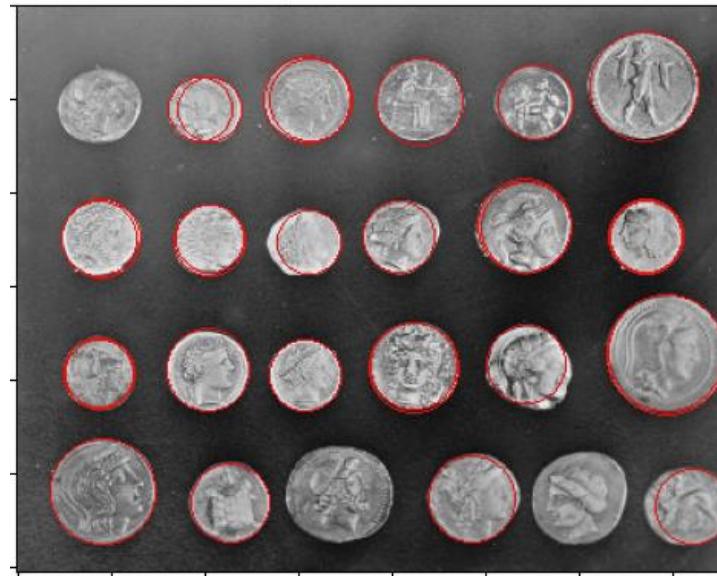
LoG Blob Detection



LoG Blob Detection



Canny Edge Detection



Hough Circle Transform

What is convolution?

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

These are the network parameters to be learned.

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).

What is convolution?

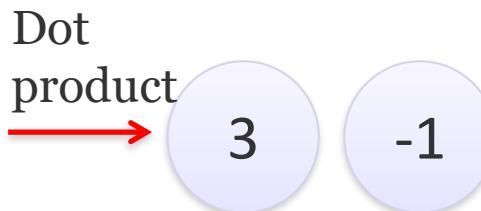
stride=1

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

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

Filter 1



What is convolution?

If stride=2

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

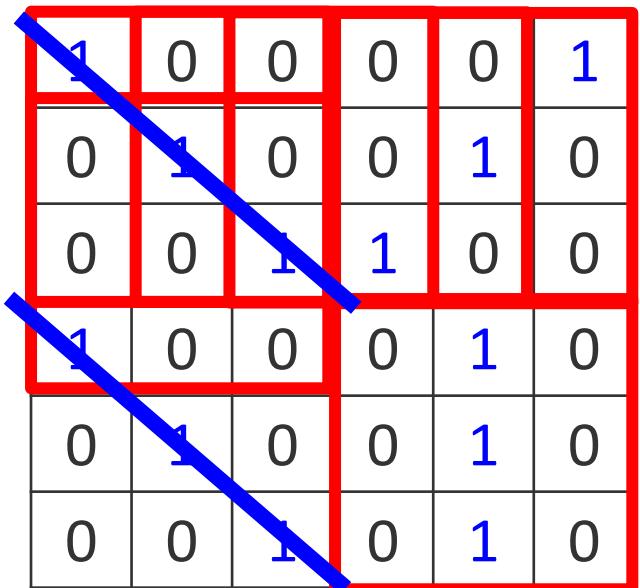
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

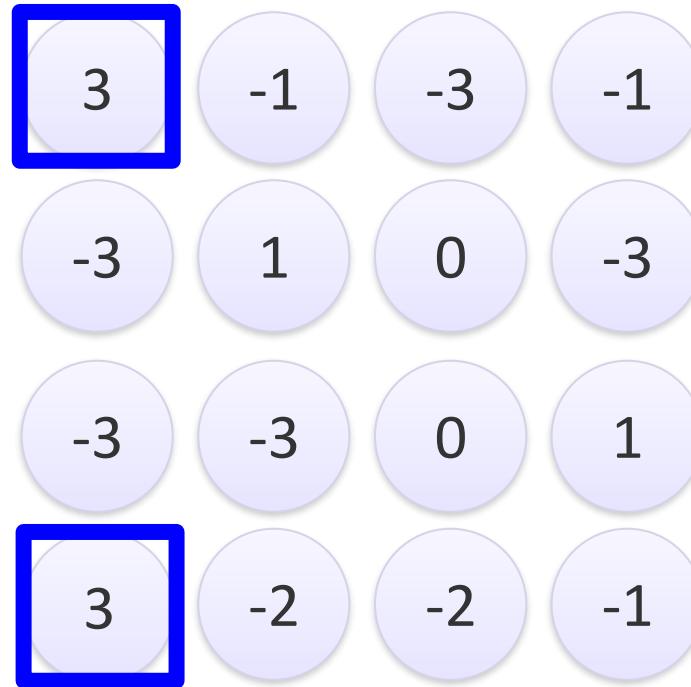
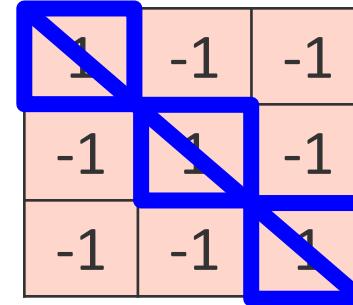


What is convolution?

stride=1



6 x 6 image



What is convolution?

stride=1

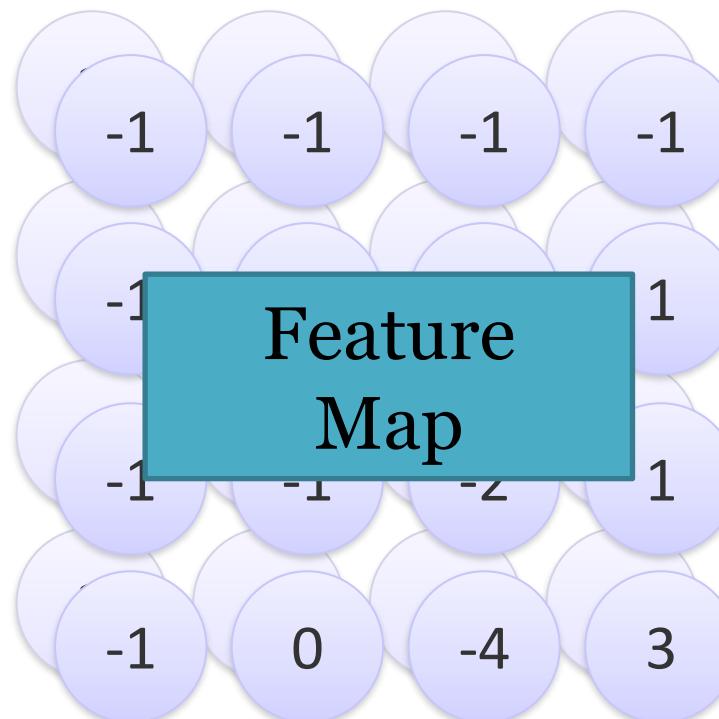
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

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

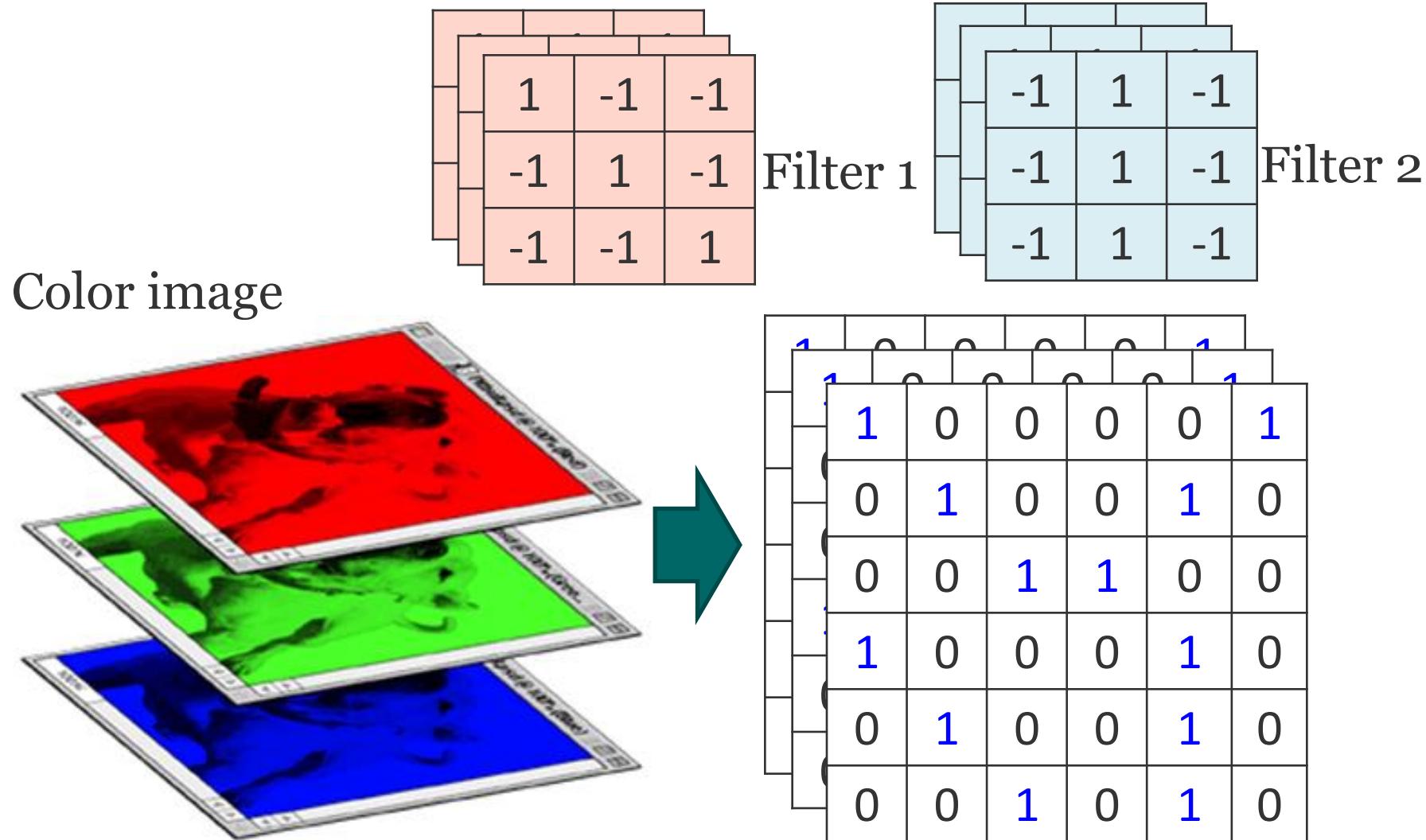
Filter 2

Repeat this for each filter

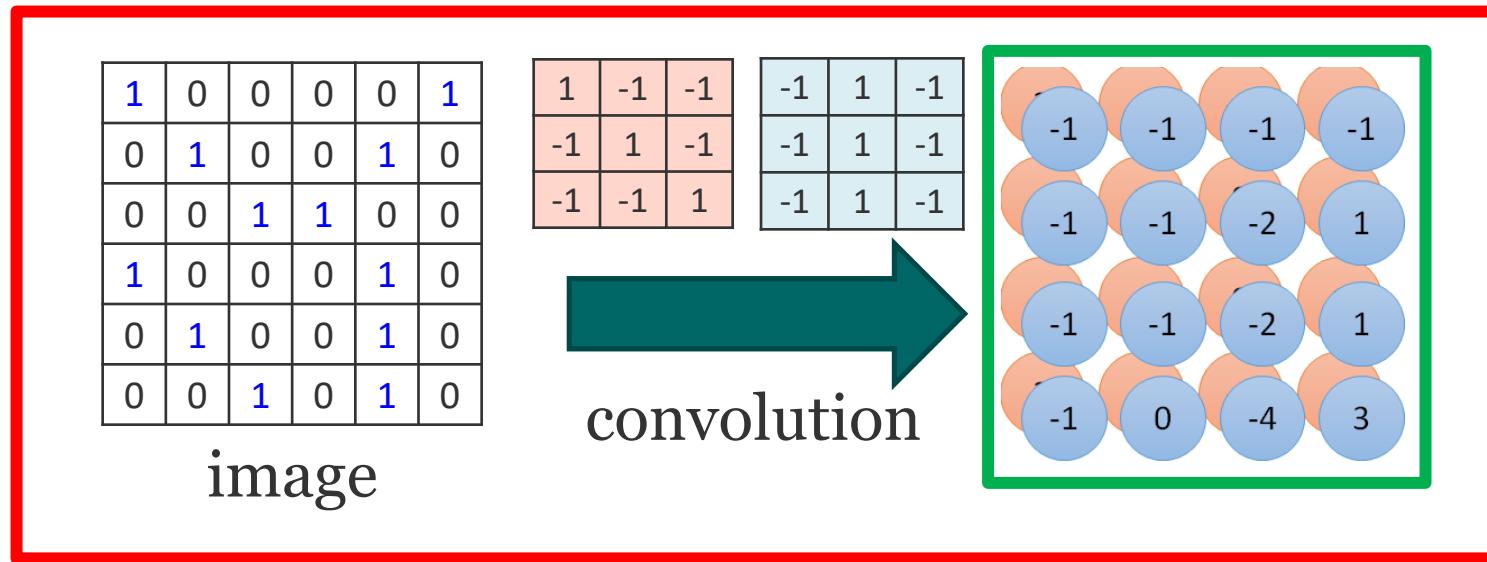


Two 4 x 4 images
Forming 2 x 4 x 4 matrix

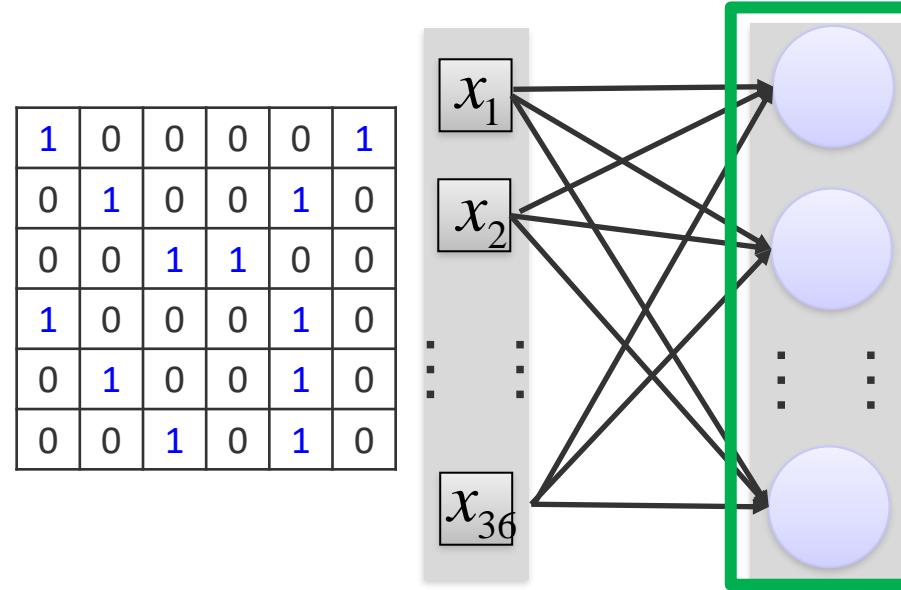
But what about color?

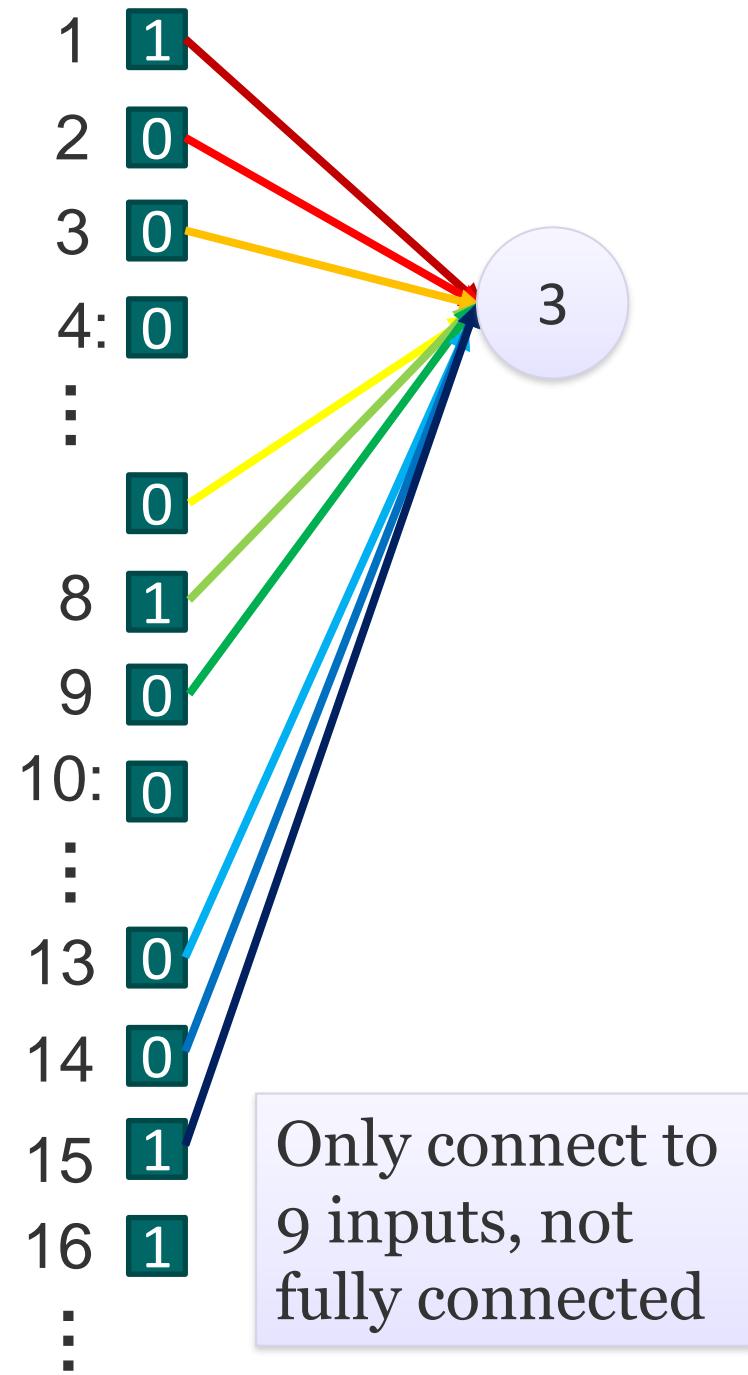
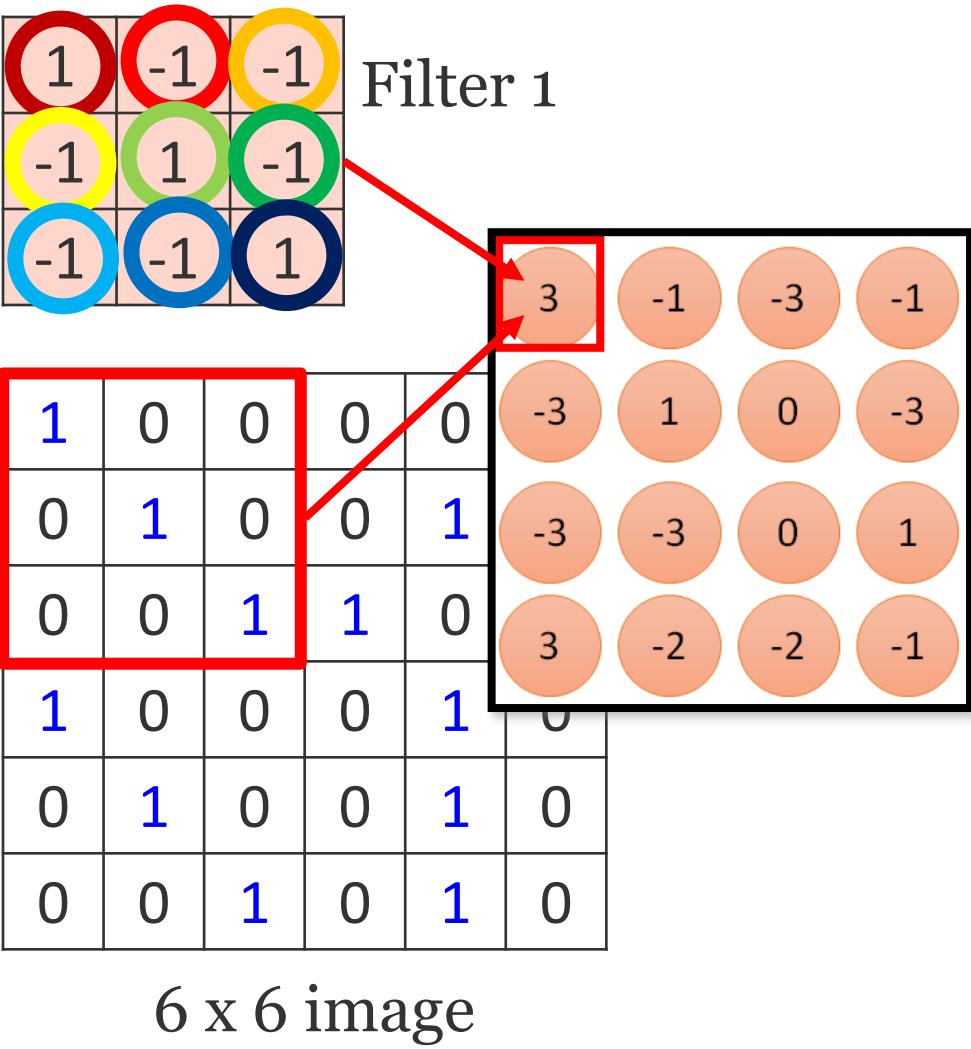


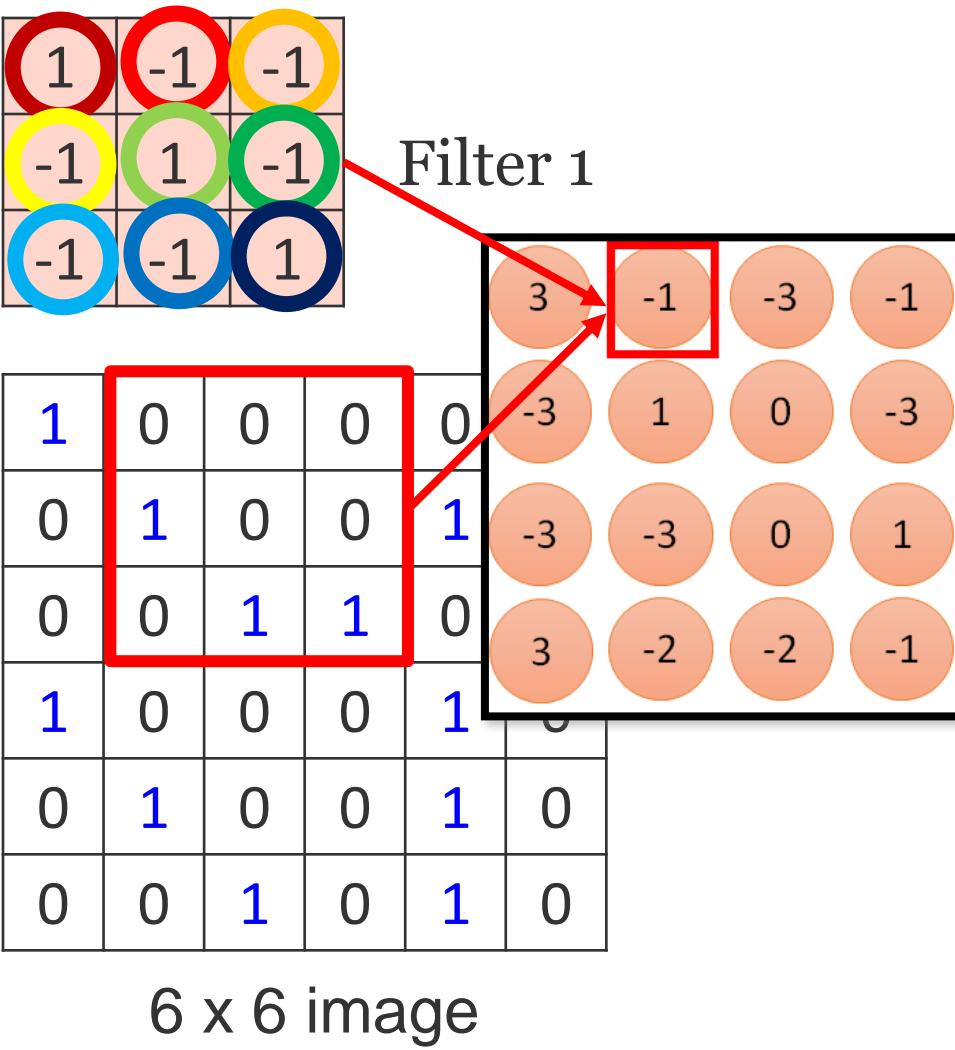
Convolution vs. fully connected



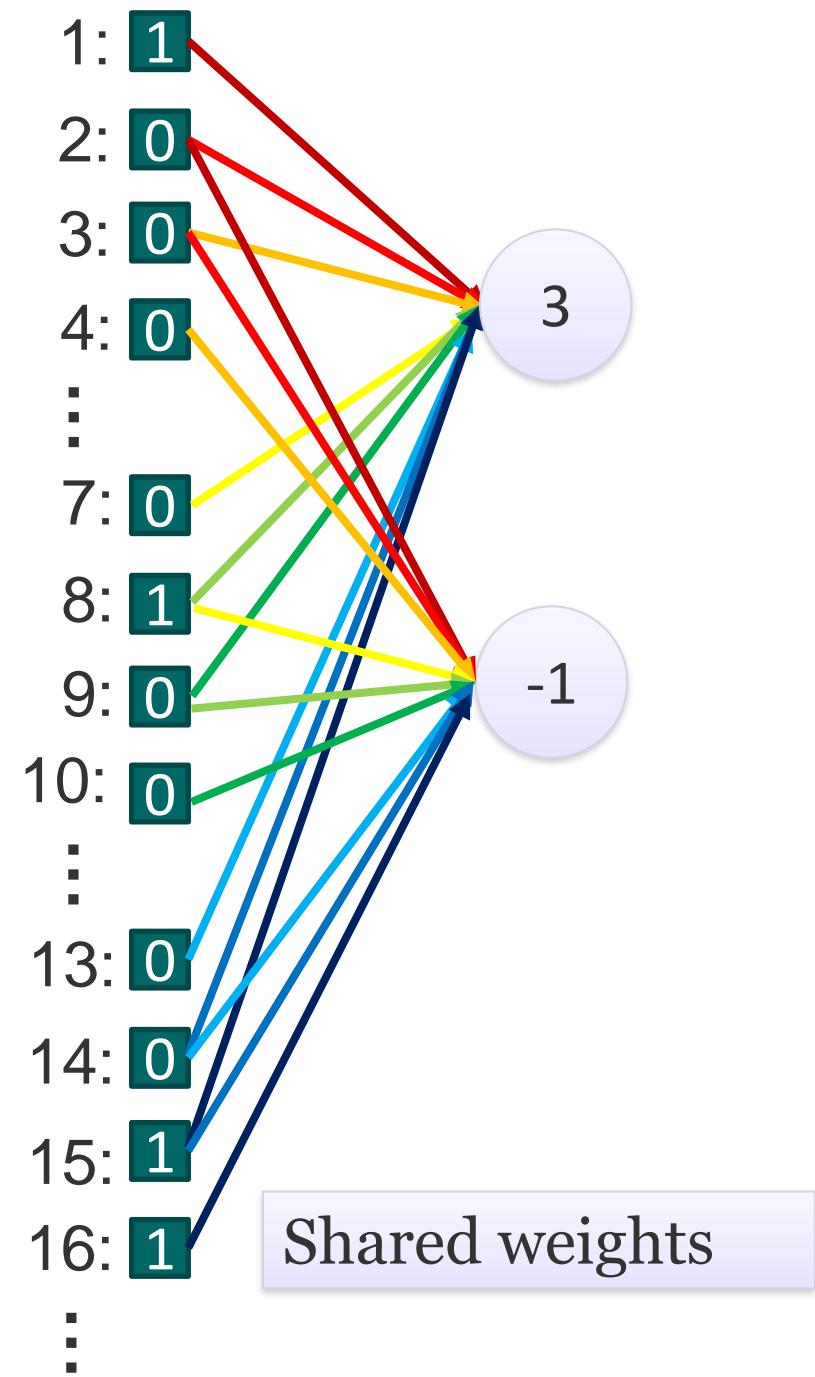
Fully-
connected







Fewer parameters



Scale invariance: pooling

- Subsampling pixels will not change the object
bird



Subsampling



We can subsample the pixels to make image smaller
 fewer parameters to characterize the image

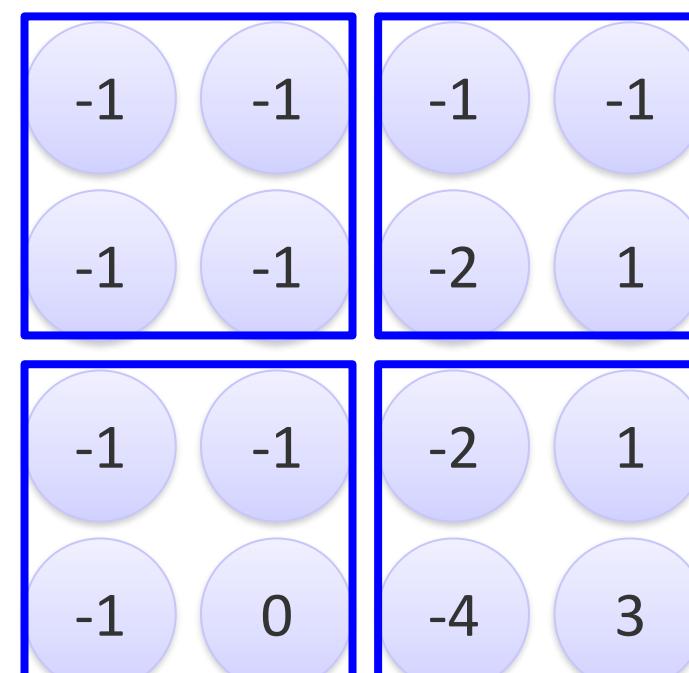
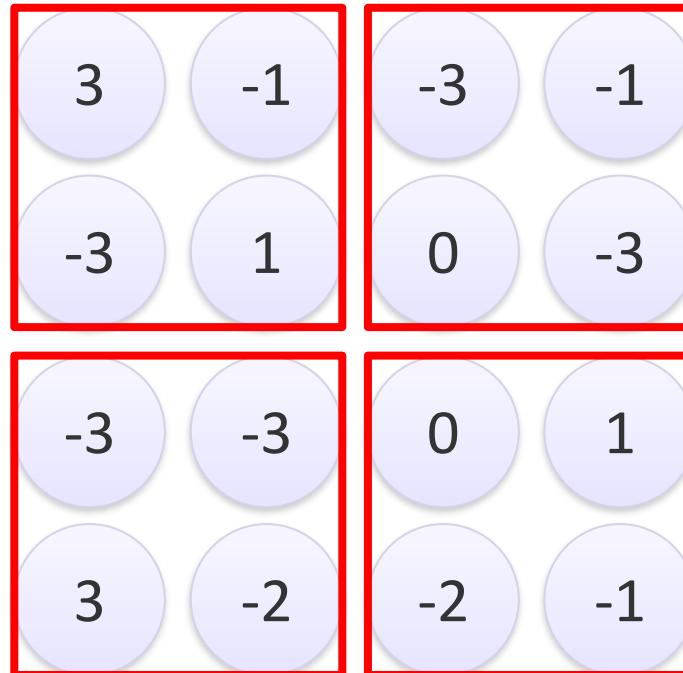
MaxPooling

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

Filter 1

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

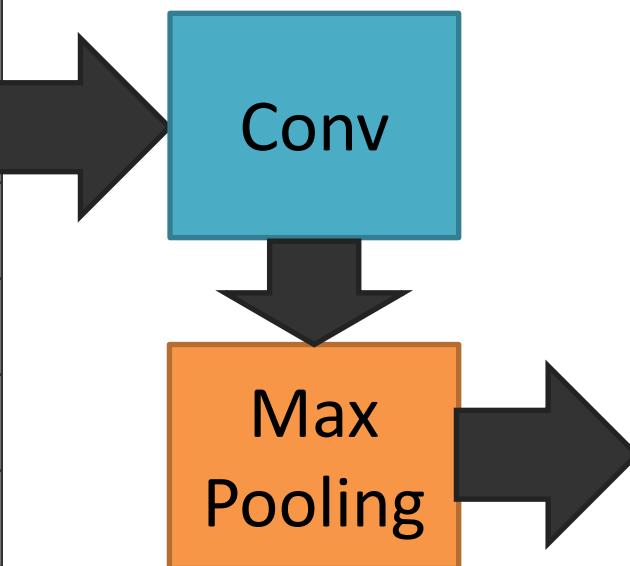
Filter 2



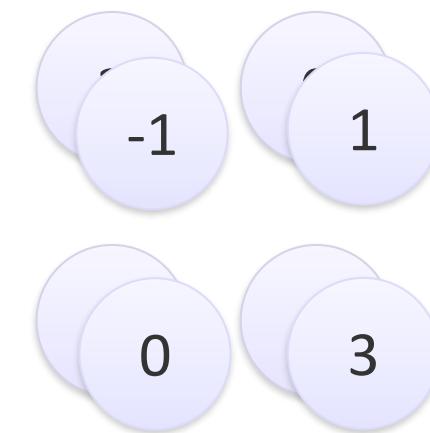
MaxPooling

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



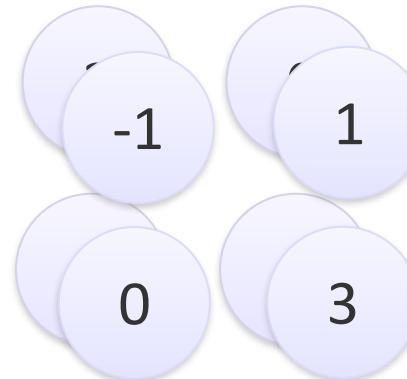
New image
but smaller



2 x 2 image

Each filter
is a channel

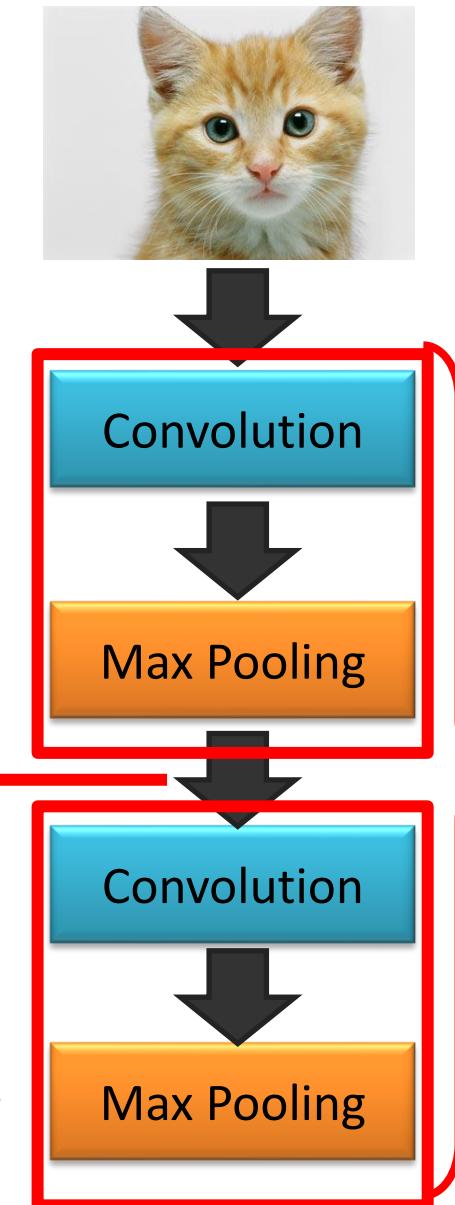
Making DCNN



A new image

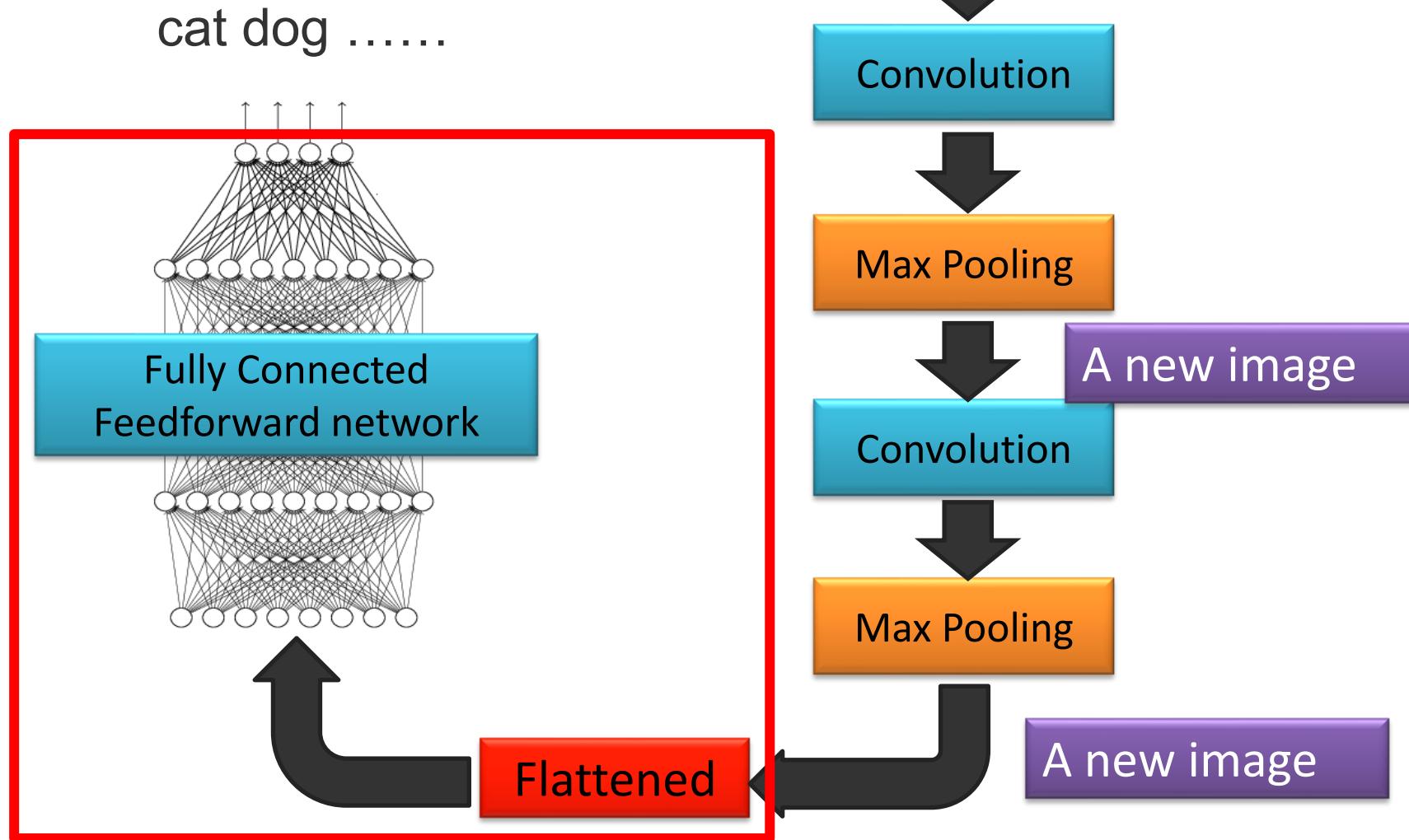
Smaller than the original image

The number of channels is the number
of filters

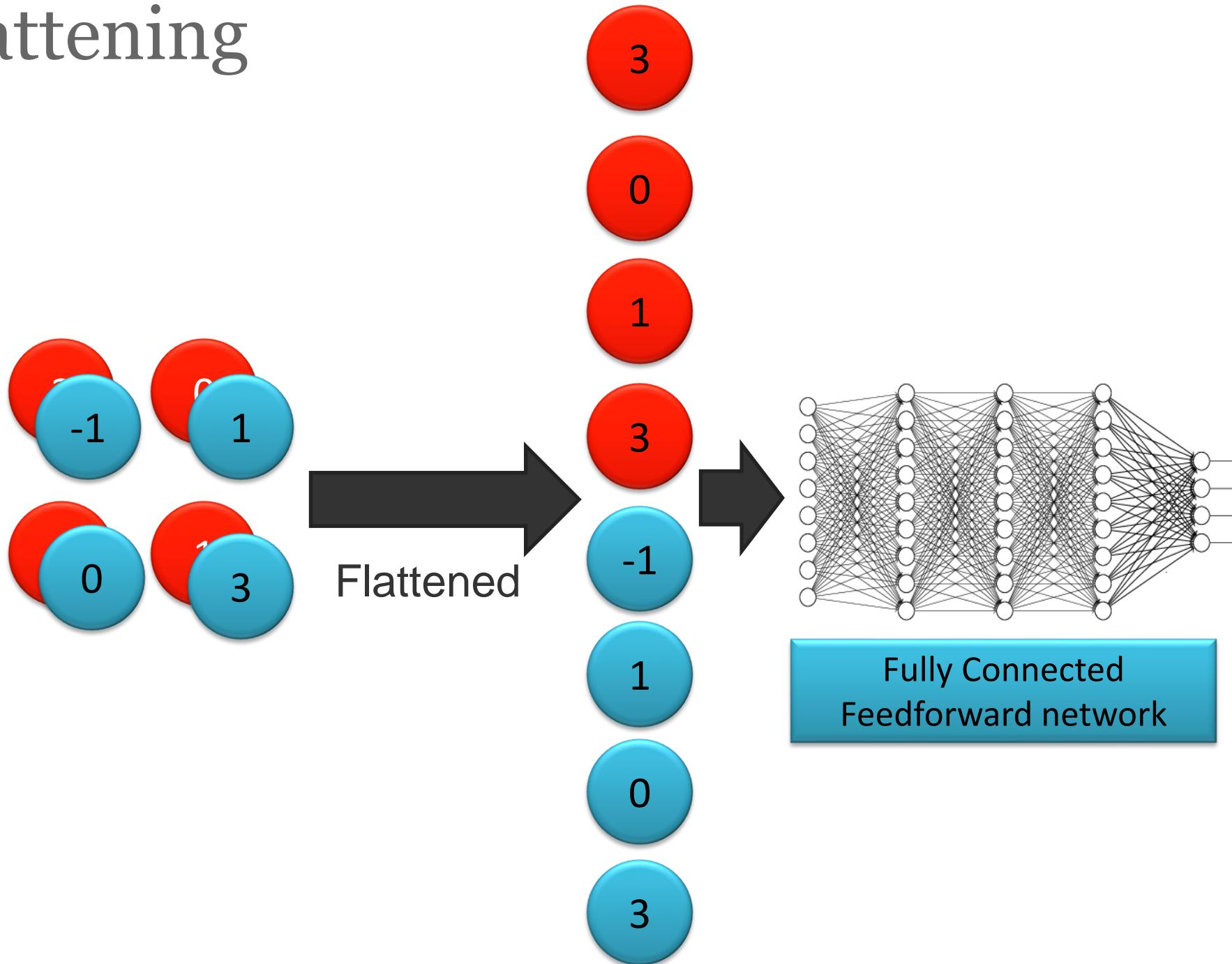


Can repeat
many times

Making DCNN



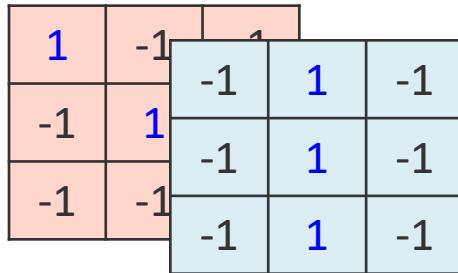
Flattening



Keras

Only modified the *network structure* and *input format (vector -> 3-D tensor)*

```
model2.add( Convolution2D( 25, 3, 3,  
                           input_shape=(28, 28, 1)) )
```



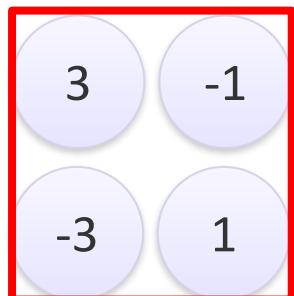
Input_shape = (28 , 28 , 1)

28 x 28 pixels

There are
25 3x3
filters.

1: black/white, 3: RGB

```
model2.add(MaxPooling2D( (2, 2) ))
```



input

Convolution

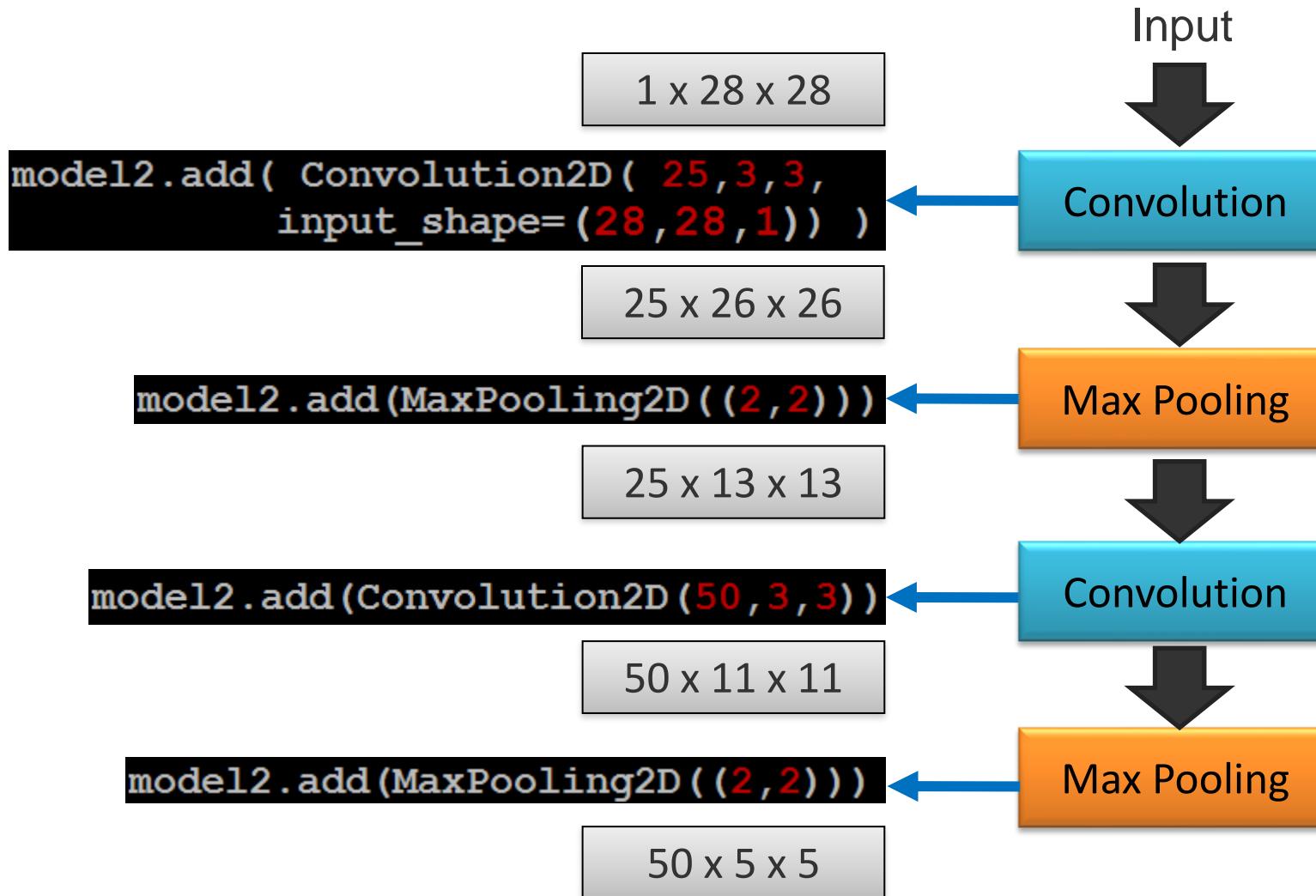
Max Pooling

Convolution

Max Pooling

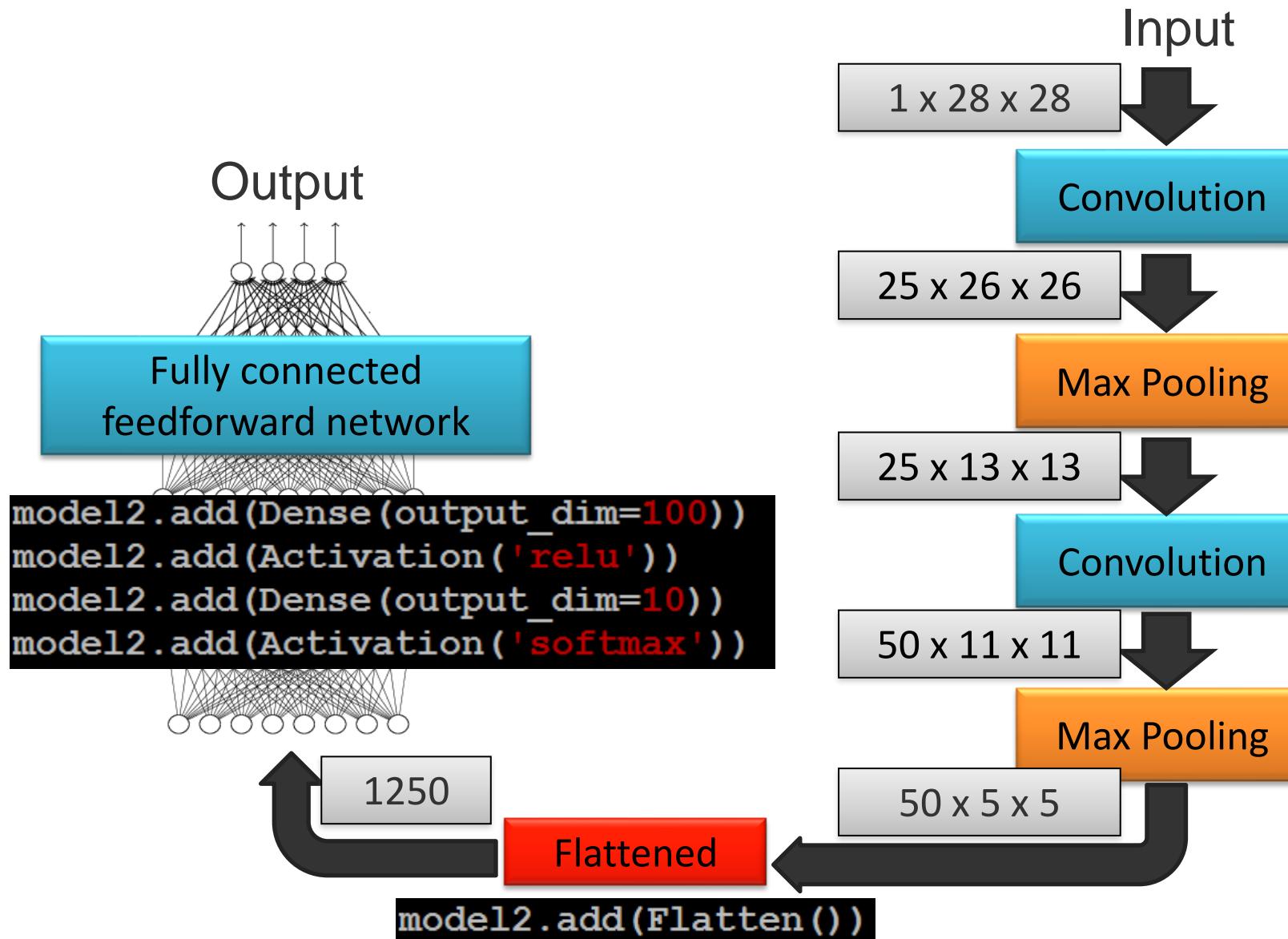
Keras

Only modified the *network structure* and *input format (vector -> 3-D array)*



Keras

Only modified the *network structure* and *input format (vector -> 3-D array)*

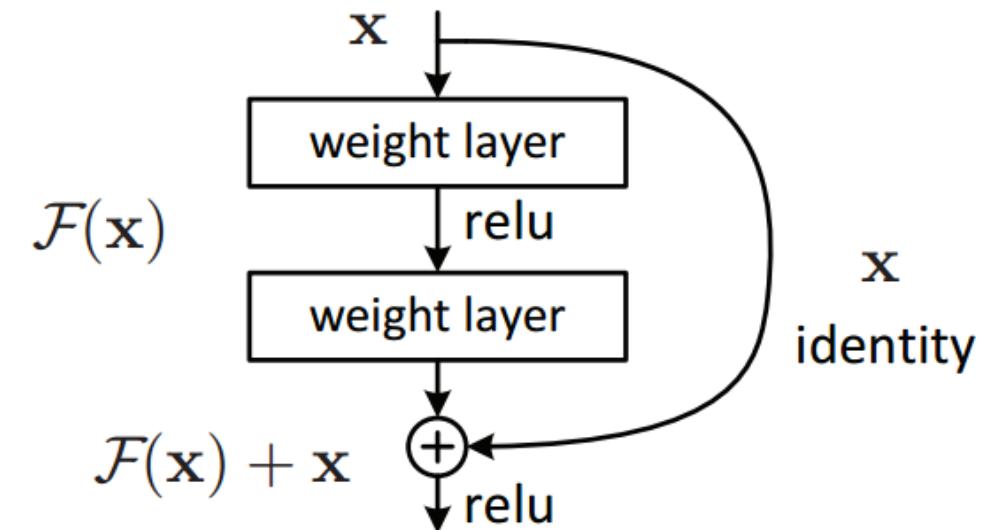


Layer types

1. **Convolutional Layer:** Extracts features from the input image through convolution operations. Key Parameters: Number of filters, kernel size, strides, padding.
2. **Activation Layer:** Introduces non-linearity to the model, allowing it to learn complex patterns. Common Types: ReLU (Rectified Linear Unit), Sigmoid, Tanh.
3. **Pooling Layer:** Reduces the spatial dimensions (height and width) of the input volume, making the model more computationally efficient and less sensitive to the exact location of features. Types: Max Pooling, Average Pooling.
4. **Batch Normalization Layer:** Normalizes the output of a previous layer by subtracting the batch mean and dividing by the batch standard deviation. Helps in speeding up training and reducing the sensitivity to network initialization.
5. **Dropout Layer:** Randomly sets a fraction of input units to zero at each update during training, helping to prevent overfitting.

Layer types

6. **Fully Connected (Dense) Layer:** Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular neural networks. Used to classify or regress based on features extracted by the convolutional layers.
7. **Flatten Layer:** Flattens the input from a multi-dimensional tensor to a 1D tensor, making it possible to connect convolutional and pooling layers with dense layers.
8. **Residual Connections** (as in ResNet): Allows the output of one layer to bypass one or more layers and be added to the output of a later layer, improving gradient flow through the network.
9. **Transposed Convolutional Layer (Deconvolution):** Used in generative models and some segmentation tasks; it upscales the input feature map.



DCNNs vs. MLP

1. Parameter Efficiency

- Fewer Parameters: CNNs require significantly fewer parameters than FCNs. They use shared weights and convolutional filters, reducing the total number of trainable parameters. This makes CNNs more efficient and less memory-intensive.
- Reduces Overfitting: With fewer parameters, CNNs are less prone to overfitting, especially with image data.

2. Exploitation of Spatial Structure

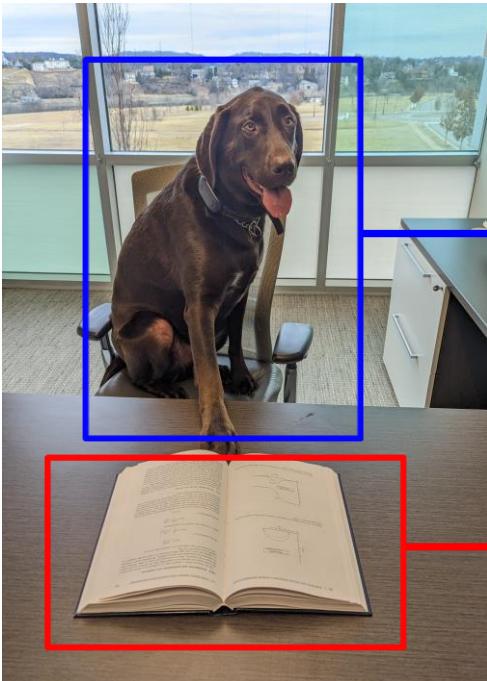
- Local Connectivity: CNNs exploit the spatial structure of the data by applying convolutional filters that capture local features (like edges, textures) in early layers and more complex features (like patterns or object parts) in deeper layers.
- Preservation of Spatial Relationships: Unlike FCNs that lose spatial relationships by flattening the input, CNNs maintain the spatial hierarchy and relationships between different parts of the input.

3. Translation Invariance

- Robust to Translation: Due to pooling layers and the nature of convolution operations, CNNs are inherently more robust to the translation of input data. This means that if an object shifts in an image, a CNN can still detect it effectively.

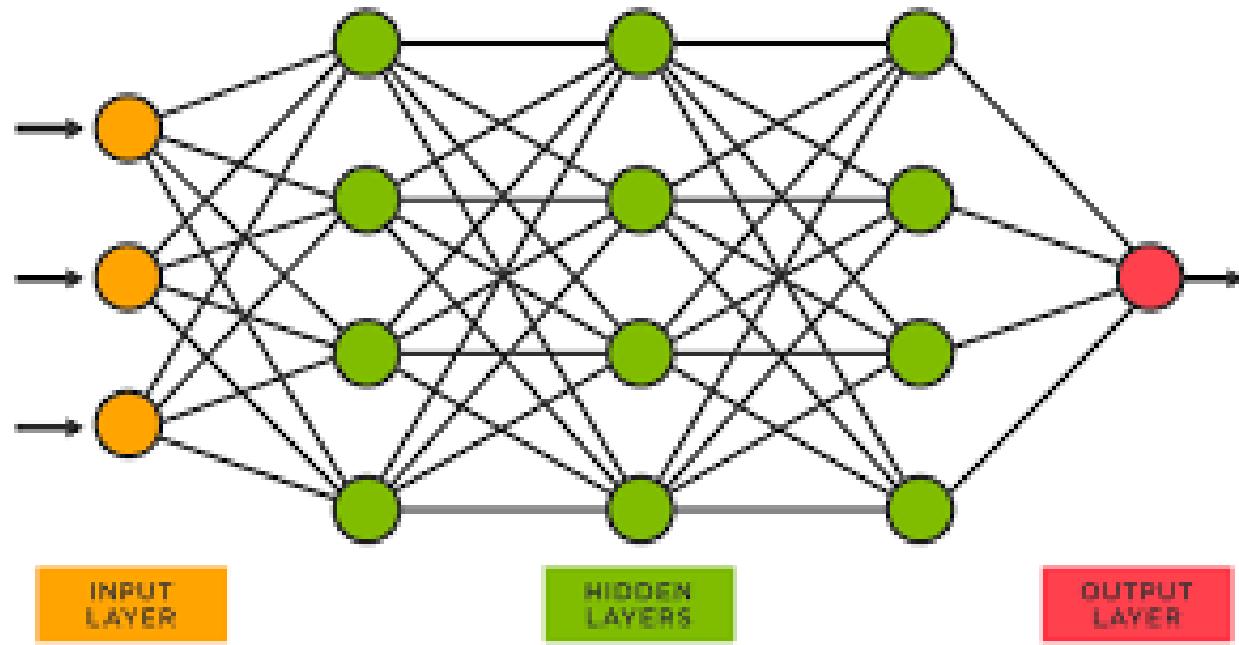
Supervised Machine Learning

- Regression
- Classification
- Semantic segmentation
- Instance segmentation
- ...



Dog

Book

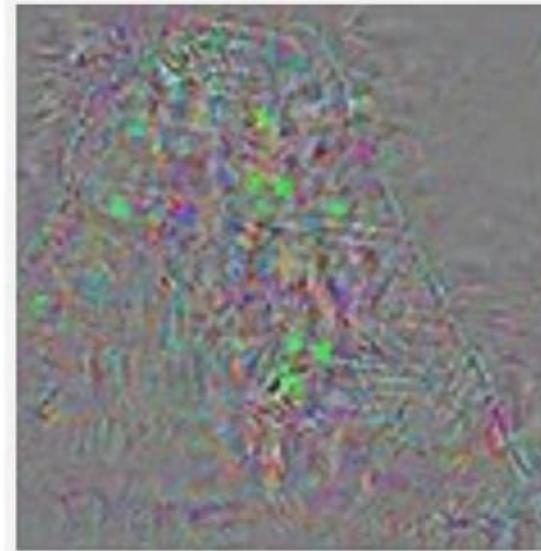


Adversarial attacks

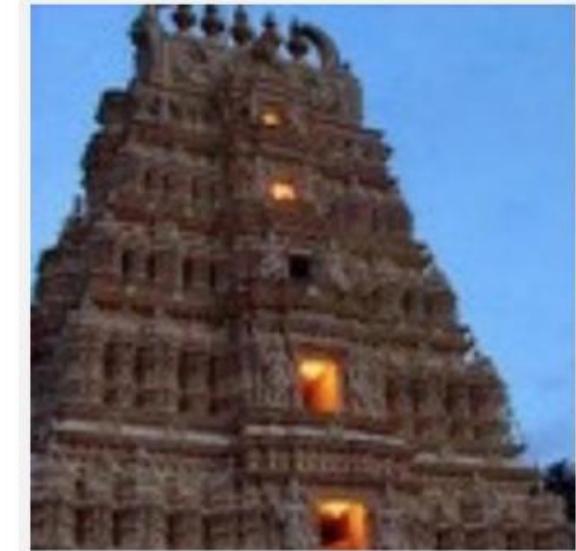


Original image

Temple (97%)



Perturbations



Adversarial example

Ostrich (98%)

What are the problems with ML models?

- We don't trust the models
- We don't know what happens in extreme cases
- Mistakes can be expensive / harmful
- Does the model make similar mistakes as humans ?
- How to change model when things go wrong ?

What do we want to get?

- Interactive feedback - can model learn from human actions in online setting ?
(Can you tell a model to not repeat a specific mistake ?)
- Recourse – Can a model tell us what actions we can take to change its output ?
(For example, what can you do to improve your credit score?)

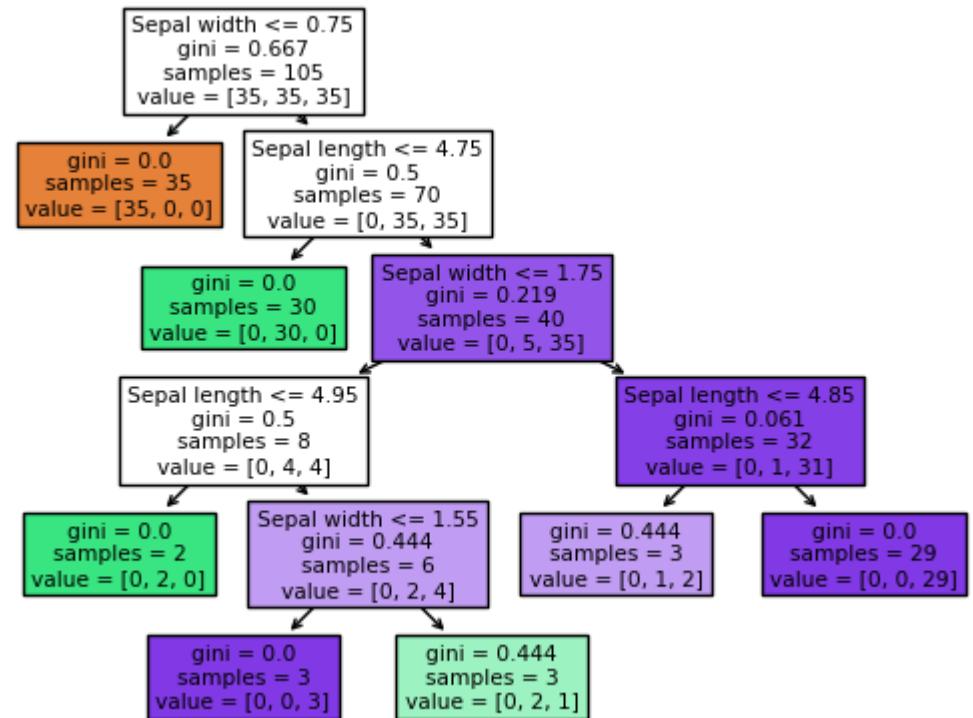
Models: Explainable and Not

Some models are explainable:

1. Linear or physics-defined function
2. Decision trees

But what about:

1. Image segmentation
2. Natural language processing
3. Classification
4. ...



What is explainability?

- **Faithfulness:** how to provide explanations that accurately represent the true reasoning behind the model's final decision.
- **Plausibility:** Is the explanation correct or something we can believe is true, given our current knowledge of the problem
- **Understandable:** Can I put it in terms that end user without in-depth knowledge of the system can understand ?
- **Stability:** Does similar instances have similar interpretations ?

What do we expect from explainer?

- 1. Interpretable:** It should provide a qualitative understanding between the input variables and the response. It should be easy to understand.
- 2. Local Fidelity:** It might not be possible for an explanation to be completely faithful unless it is the complete description of the model itself. Having said that it should be at least locally faithful, i.e it must replicate the model's behavior in the vicinity of the instance being predicted.
- 3. Model Agnostic:** The explainer should be able to explain any model and should not make any assumptions about the model while providing explanations.
- 4. Global perspective:** The explainer should explain a representative set to the user so that the user has a global intuition of the model

Ways to explain ML methods

Global vs local:

- Do we explain individual prediction (Heatmaps, Rationales)?
- Do we explain entire model (Linear Regression, Decision Trees)?

Inherent or post-hoc:

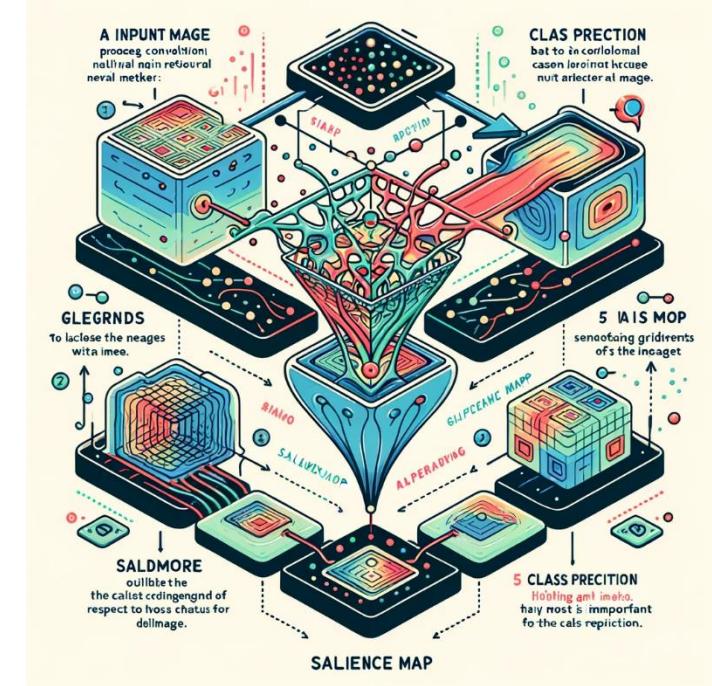
- Is the explainability built into the model (Linear Regression, Decision Trees, Natural Language Explanations)
- Is the model black-box and we use external method to try to understand it (Heatmaps)?

Model based vs Model Agnostic

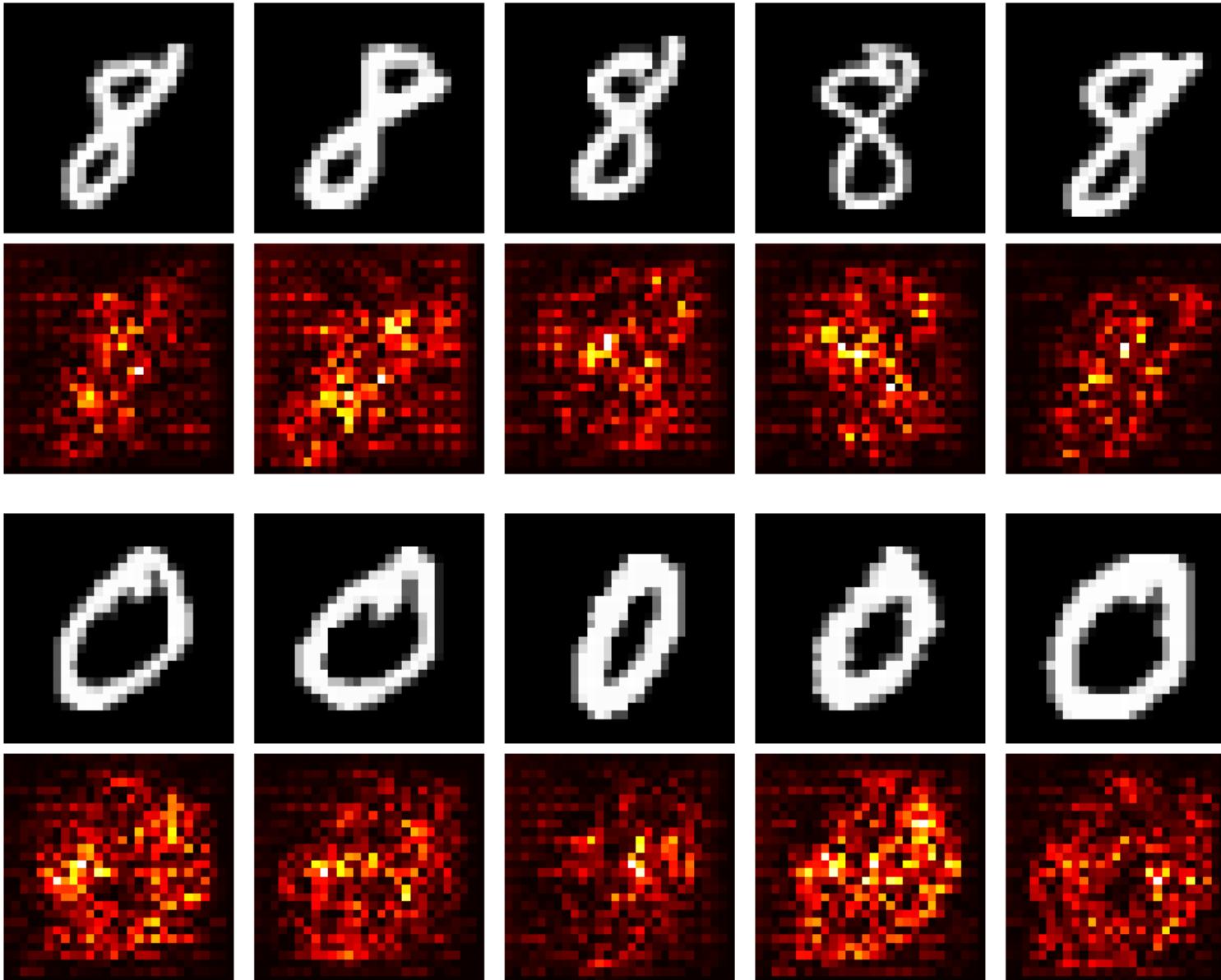
- Can it explain only few classes of models (attention gradients – differentiable models only)
- Can it explain any model (LIME, SHAP)?

Saliency maps for differentiable models

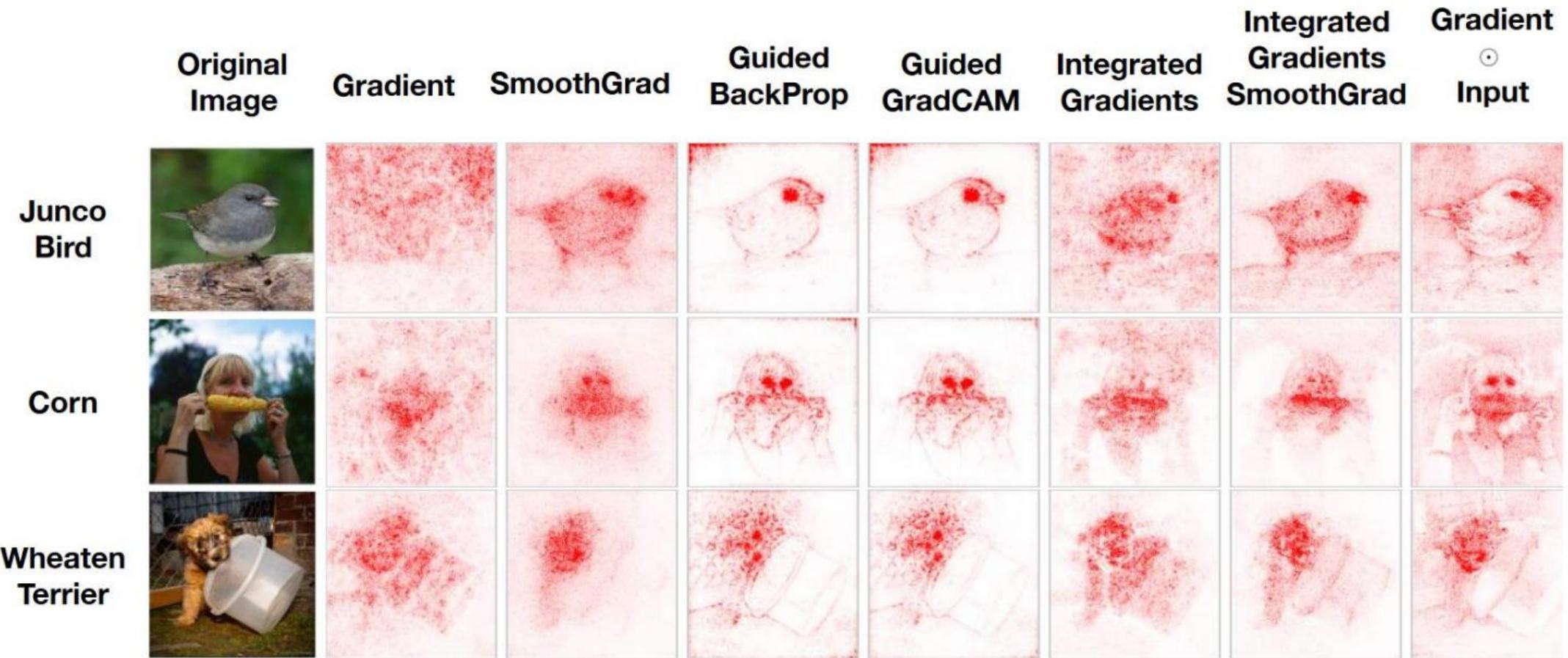
- Choose the target class for which you want to compute the saliency map. This could be the class predicted by the network or any other class of interest.
- Pass the image through the model to get the output predictions. In the case of classification, this output is typically a probability distribution over classes.
- Extract the model's output (e.g., the probability or the logit) corresponding to the target class.
- Calculate the gradient of the output for the target class with respect to the input image. It highlights how much each pixel in the input image contributes to the output value for the chosen class.
- Post-process the gradients to create a saliency map:
 - Taking the absolute value of the gradient.
 - Collapsing the gradient across the color channels, often by taking the maximum or the average across channels.



Example of saliency maps for MNIST



There are many ways to get salience maps



[Adebayo et al 2018]

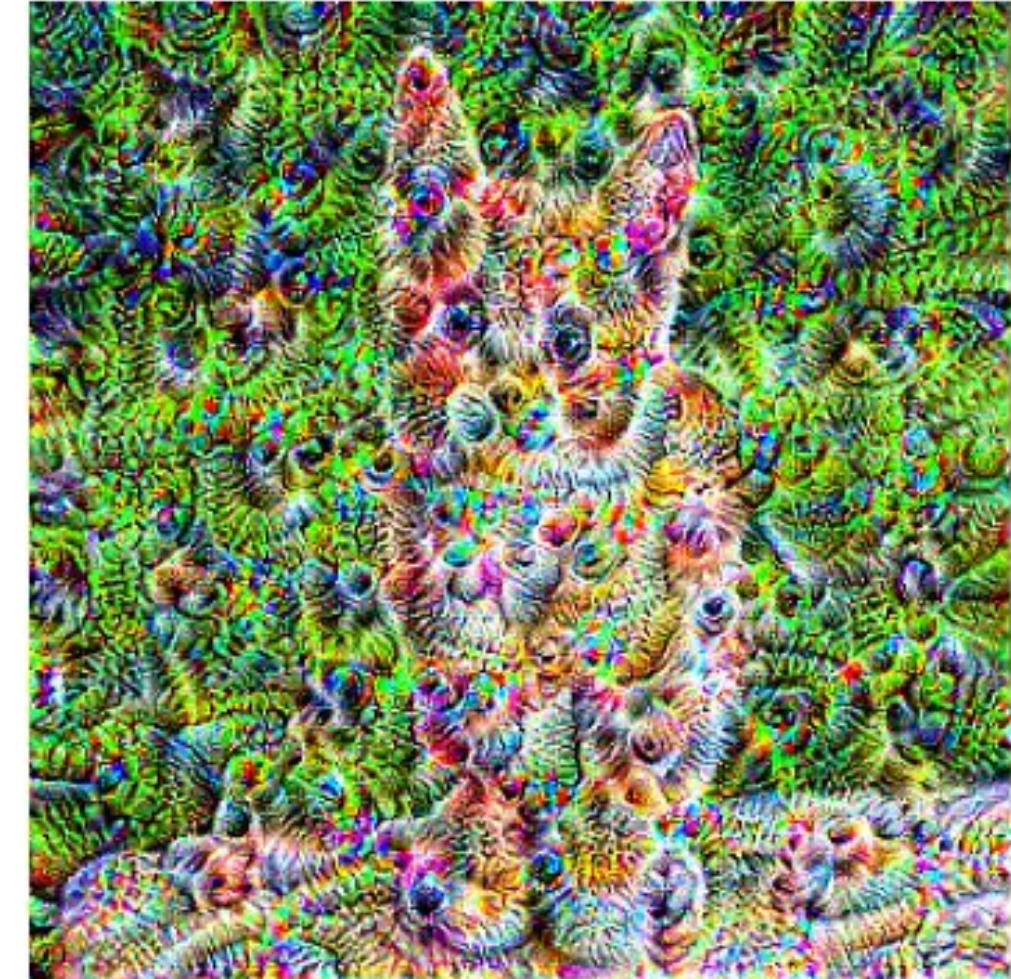
- Only capture first order information
- Not very reliable

Closely related – Deep Dream

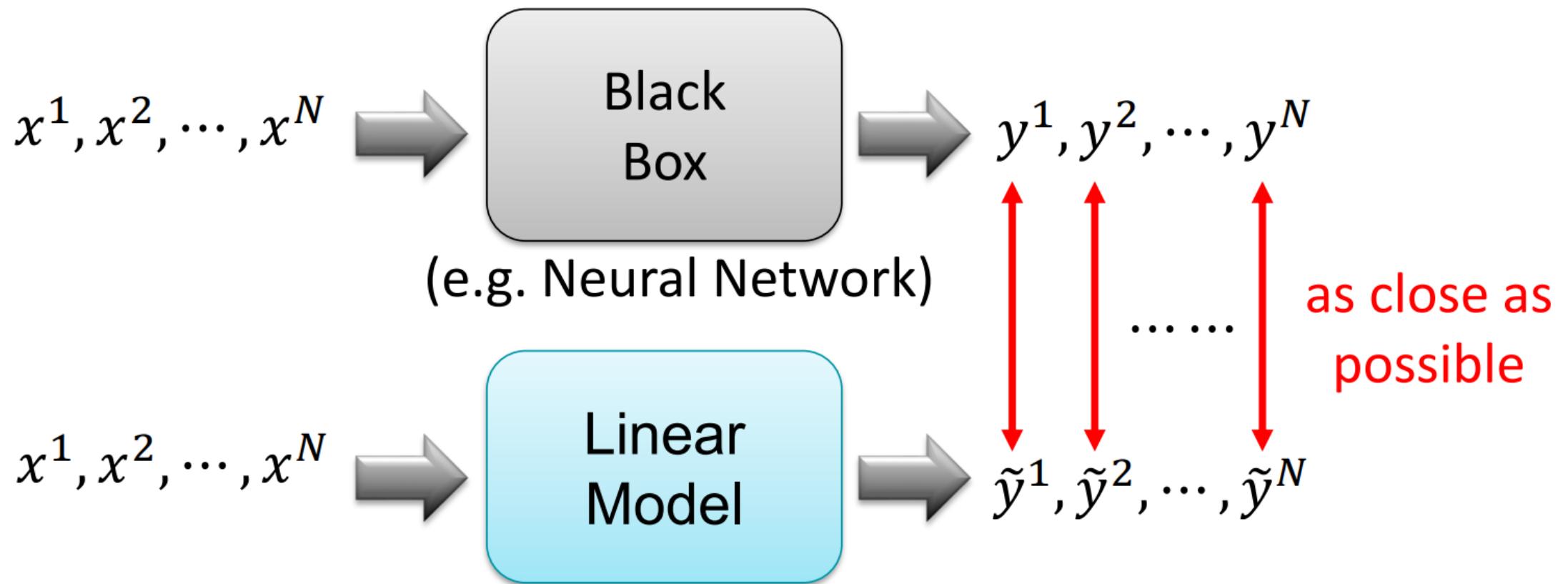
Original Image



Deep Dream Image



Locally Interpretable Manifold Embedding



Locally Interpretable Manifold Embedding

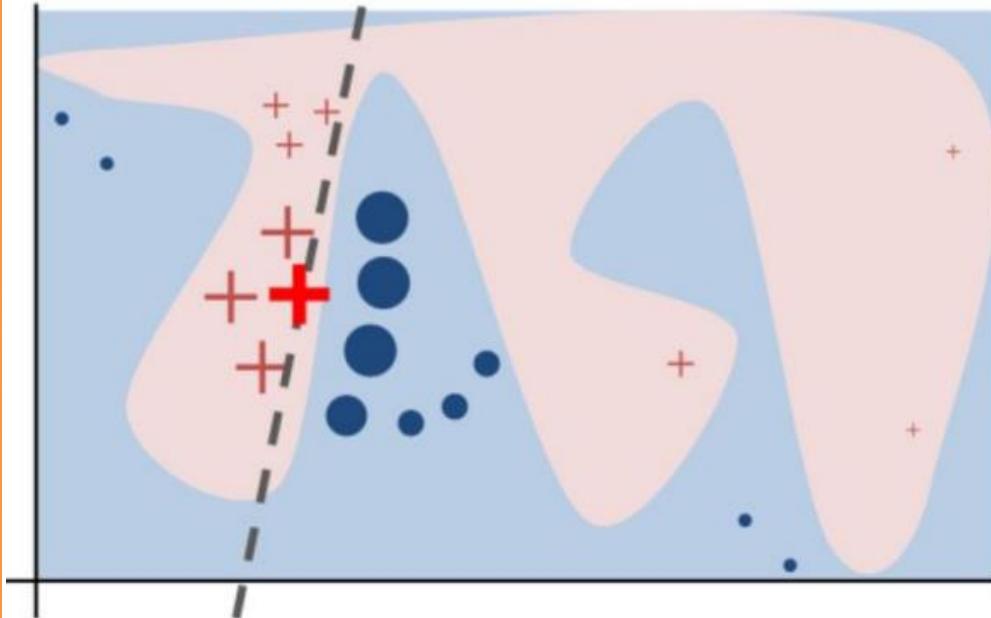


Figure 3: Toy example to present intuition for LIME. The black-box model's complex decision function f (unknown to LIME) is represented by the blue/pink background, which cannot be approximated well by a linear model. The bold red cross is the instance being explained. LIME samples instances, gets predictions using f , and weighs them by the proximity to the instance being explained (represented here by size). The dashed line is the learned explanation that is locally (but not globally) faithful.

Locally Interpretable Manifold Embedding

Fidelity-Interpretability trade-off: We want an explainer that is faithful (replicate our model's behavior locally) and interpretable. Towards this goal, LIME minimizes the following

$$\xi(x) = \operatorname{argmin}_{g \in G} \mathcal{L}(f, g, \pi_x) + \Omega(g) \quad (1)$$

- f: an original predictor
- x: original features
- g: explanation model which could be a linear model, decision tree, or falling rule lists
- Pi: proximity measure between an instance of z to x to define locality around x. It weighs z' (perturbed instances) depending upon their distance from x.

First Term: the measure of the unfaithfulness of g in approximating f in the locality defined by Pi. This is termed as locality-aware loss in the original paper

Last term: a measure of model complexity of explanation g. For example, if your explanation model is a decision tree it can be the depth of the tree or in the case of linear explanation models it can be the number of non-zero weights

LIME: Sparse Linear Explanation:

1. $g(z') = w \cdot z'$ (Making the explanation model linear)
2. Locally-aware loss = square loss
3. $\pi_i(z) : \exp(-D(x,z)^2/\sigma^2)$ (proximity weighing for the samples)
4. $D(x,z)$: Distance function

$$\mathcal{L}(f, g, \pi_x) = \sum_{z, z' \in \mathcal{Z}} \pi_x(z) (f(z) - g(z'))^2 \quad (2)$$

Algorithm 1 Sparse Linear Explanations using LIME

Require: Classifier f , Number of samples N

Require: Instance x , and its interpretable version x'

Require: Similarity kernel π_x , Length of explanation K

$\mathcal{Z} \leftarrow \{\}$

for $i \in \{1, 2, 3, \dots, N\}$ **do**

$z'_i \leftarrow \text{sample_around}(x')$

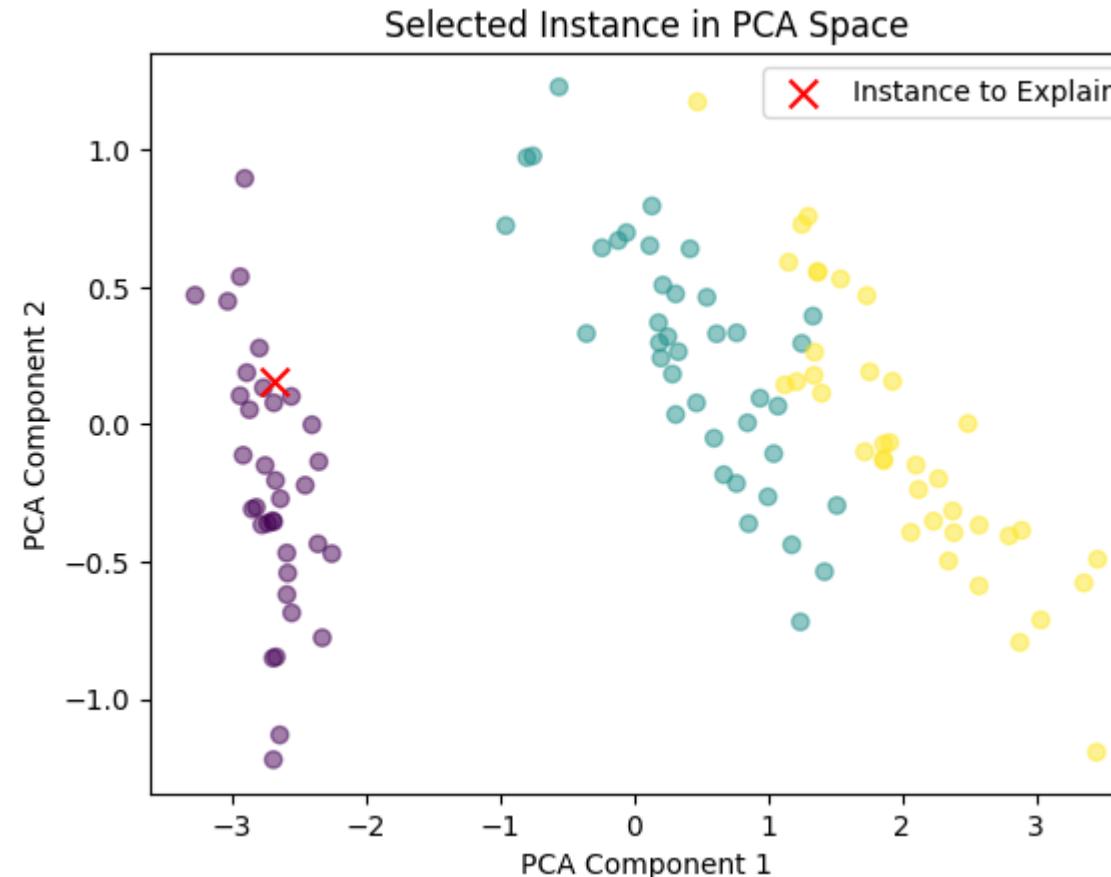
$\mathcal{Z} \leftarrow \mathcal{Z} \cup \langle z'_i, f(z_i), \pi_x(z_i) \rangle$

end for

$w \leftarrow \text{K-Lasso}(\mathcal{Z}, K)$ \triangleright with z'_i as features, $f(z)$ as target

return w

LIME on Iris



Prediction probabilities

setosa	1.00
versicolor	0.00
virginica	0.00

NOT setosa

setosa

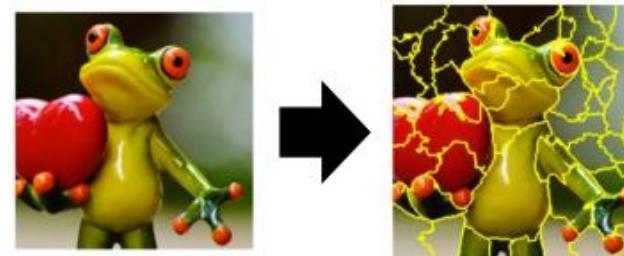
- petal width (cm) <= 0.30
- petal length (cm) <= 1.60
- sepal length (cm) <= ...
- 3.00 < sepal width (cm) ...

Feature Value

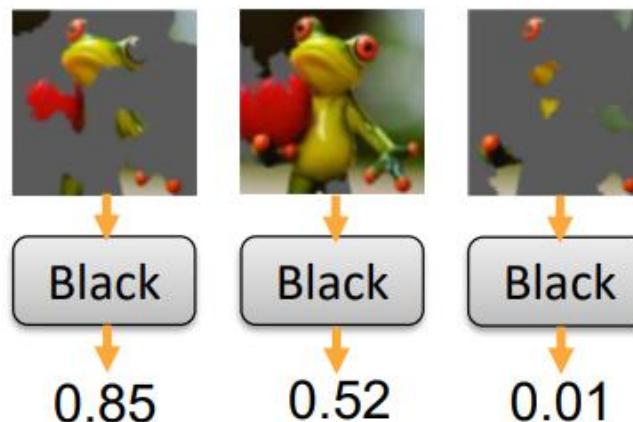
petal width (cm)	0.20
petal length (cm)	1.60
sepal length (cm)	4.70
sepal width (cm)	3.20

LIME on Images

LIME – Image



- 1. Given a data point you want to explain
- 2. Sample at the nearby - Each image is represented as a set of superpixels (segments).



Randomly delete some segments.

Compute the probability of “frog” by black box

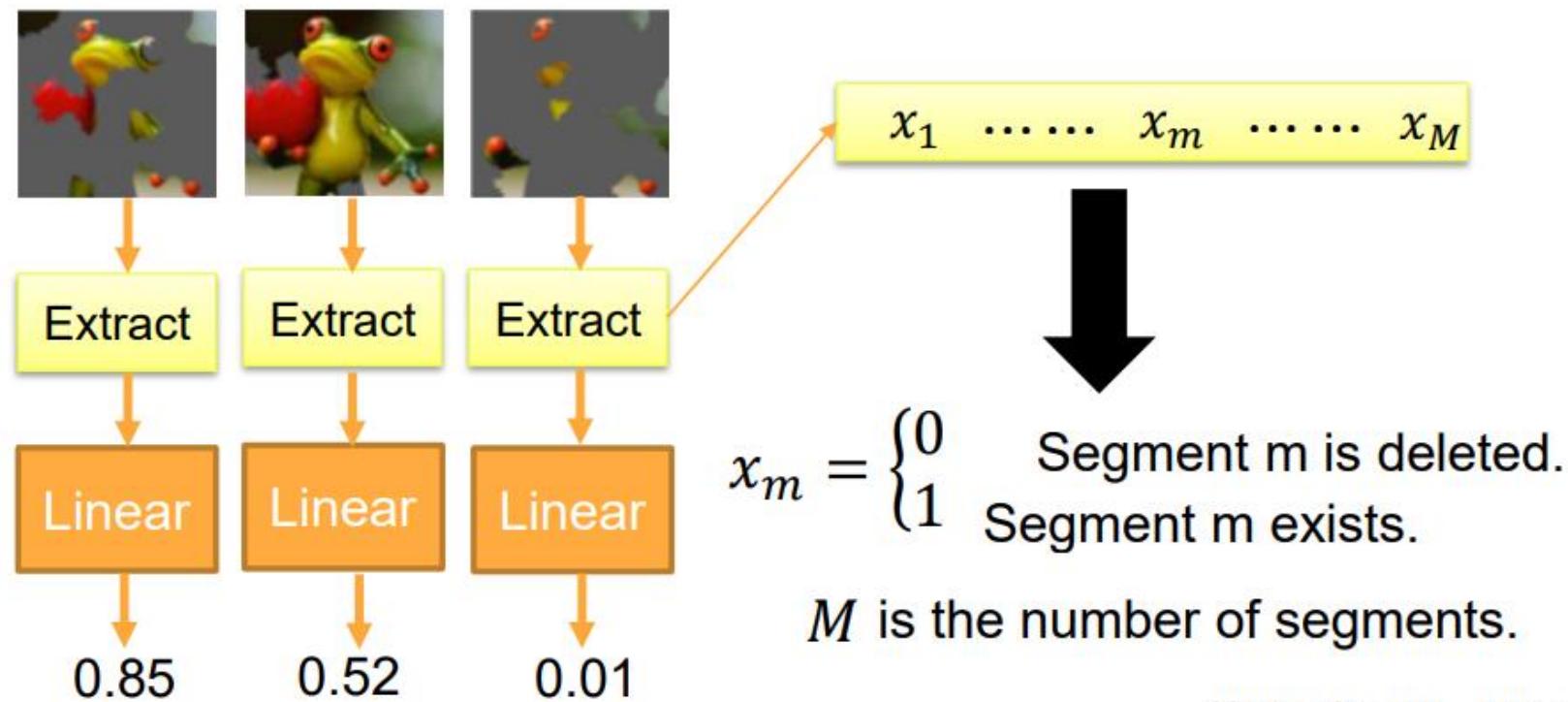
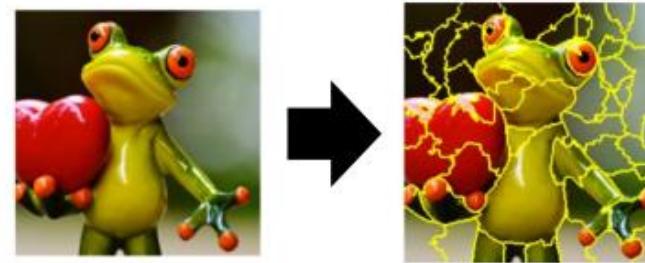
Ref: <https://medium.com/@kstseng/lime-local-interpretable-model-agnostic-explanation%E6%8A%80%E8%A1%93%E4%BB%8B%E7%B4%B9-a67b6c34c3f8>

(Slide Credit – Hung-yi Lee)

LIME on Images

LIME – Image

- 3. Fit with linear (or interpretable) model



(Slide Credit – Hung-yi Lee)

LIME on Images

LIME – Image

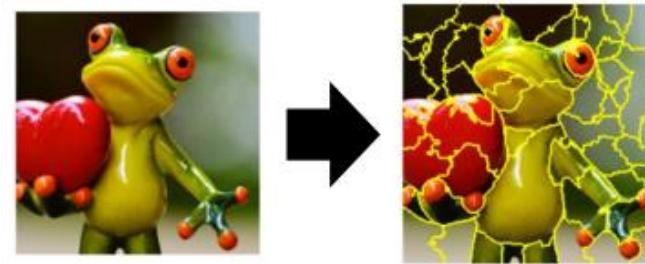
- 4. Interpret the model you learned



Extract

Linear

0.85



$$y = w_1x_1 + \dots + w_mx_m + \dots + w_Mx_M$$

$$x_m = \begin{cases} 0 & \text{Segment } m \text{ is deleted.} \\ 1 & \text{Segment } m \text{ exists.} \end{cases}$$

M is the number of segments.

If $w_m \approx 0$ → segment m is not related to “frog”

If w_m is positive → segment m indicates the image is “frog”

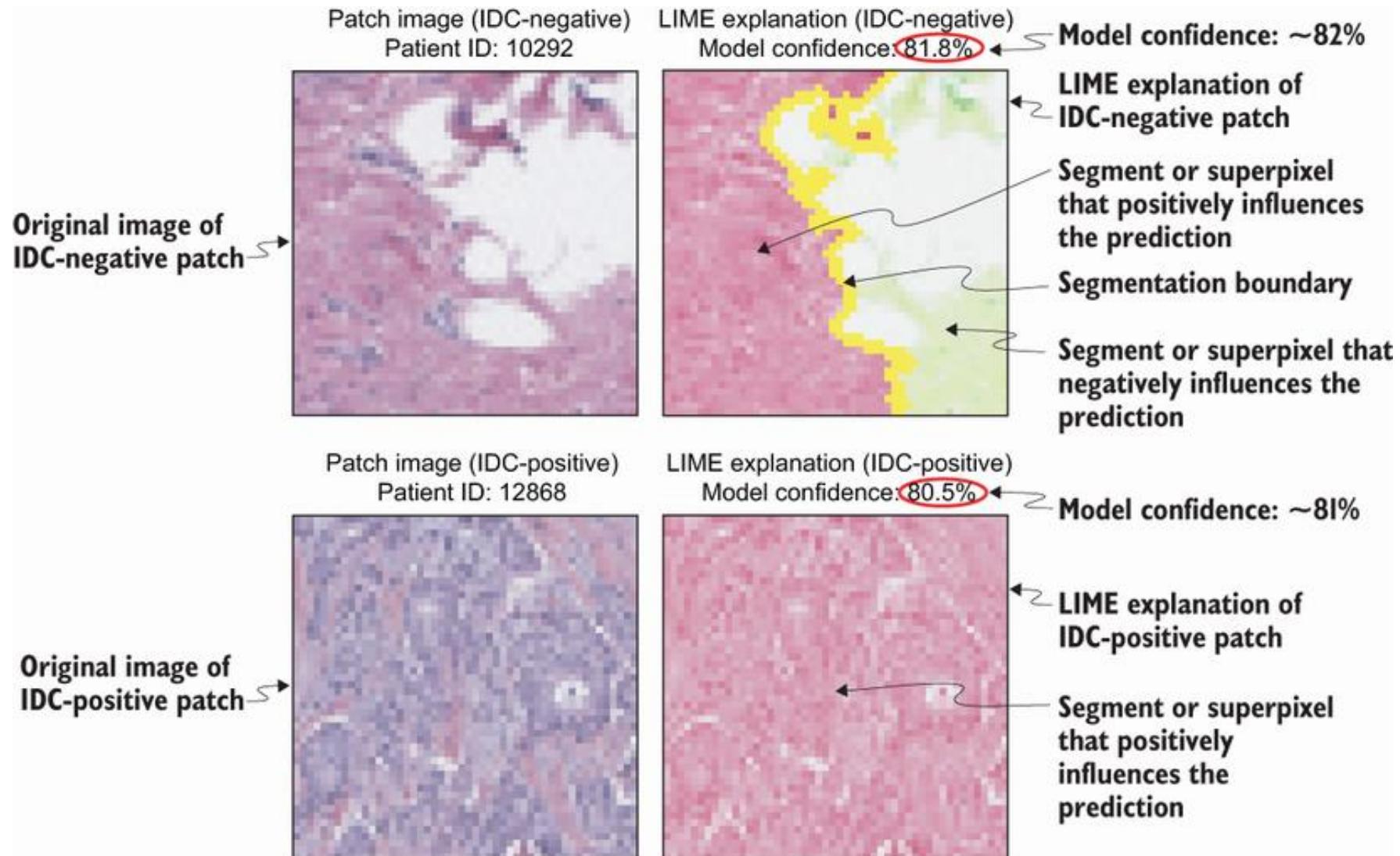
If w_m is negative → segment m indicates the image is not “frog”

(Slide Credit – Hung-yi Lee)

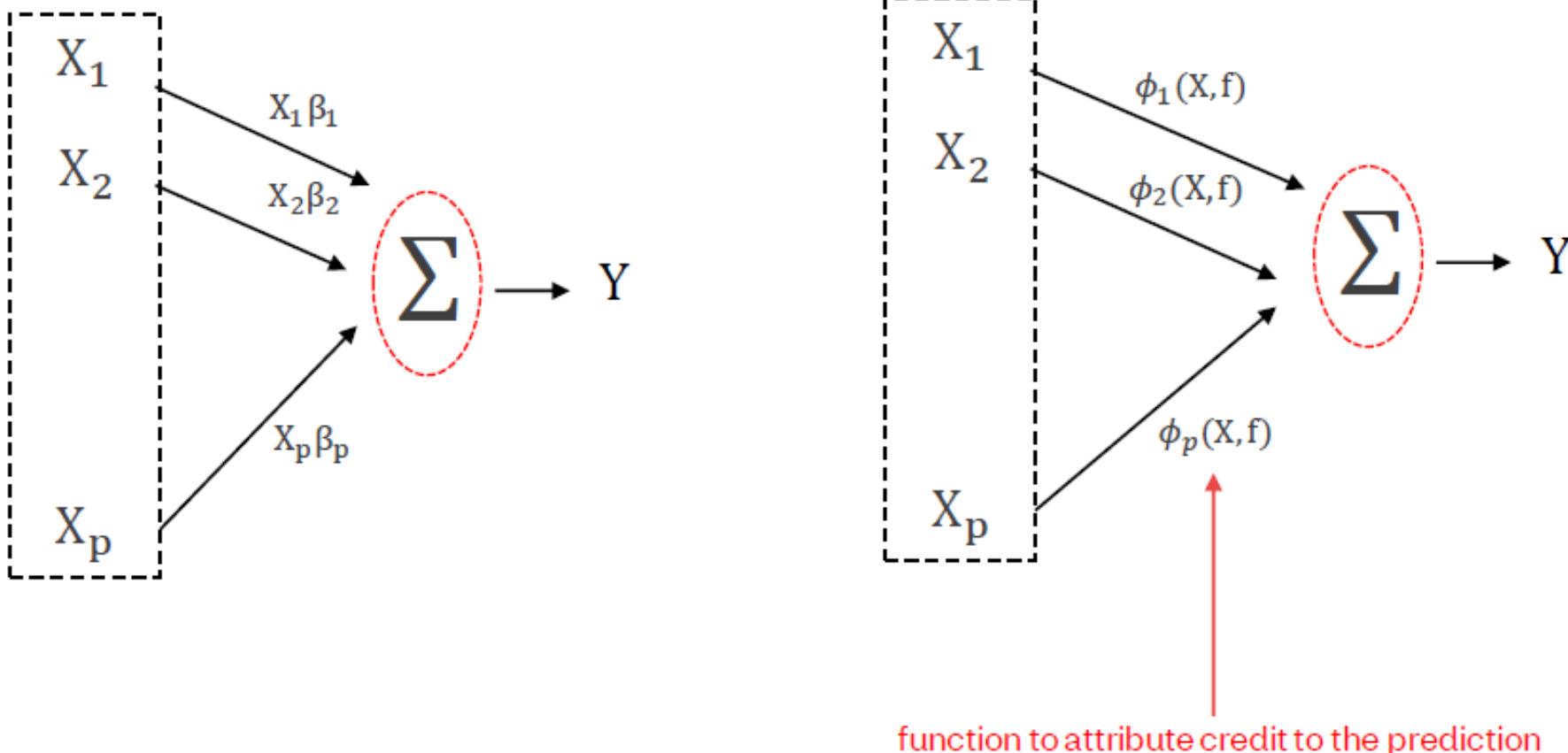
DYI LIME



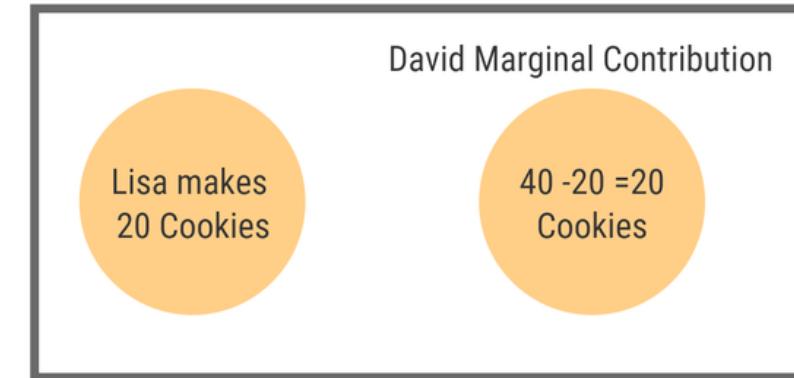
LIME on Images



SHAP: Shapley Additive exPlanations



SHAP: Shapley Additive exPlanations



- For Lisa, the contribution to the coalition is 30 cookies in the first case and her contribution to the coalition in the second case is 20 cookies. The Shapley value will be $(20+30)/2 = 25$
- To find the Shapley value of David, we need to average them: $(10+20)/2 = 15$.

SHAP: Shapley Additive exPlanations

A method of dividing up the gains or costs among players according to the value of their individual contributions.

$$\phi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} (v(S \cup \{i\}) - v(S))$$

1. Marginal contribution

The contribution of each player is determined by what is gained or lost by removing them from the game. This is called their marginal contributions.

2. Interchangeable players have an equal value

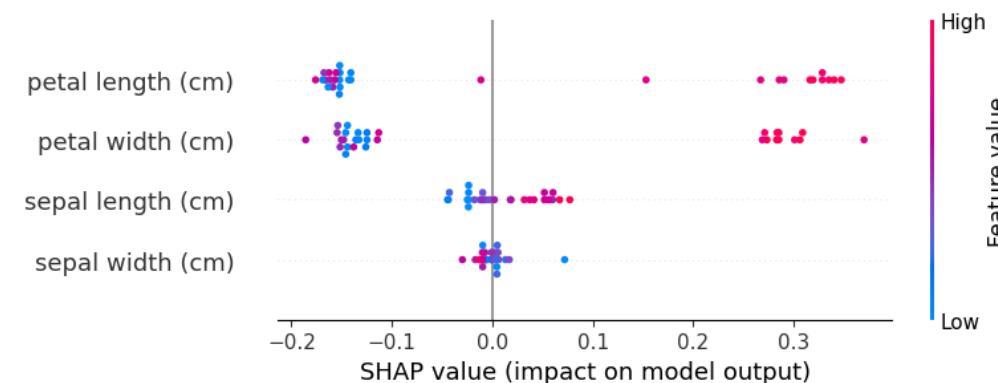
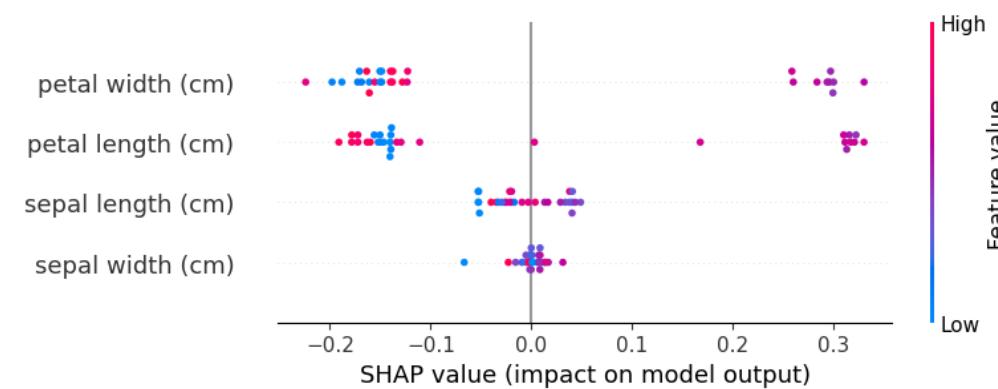
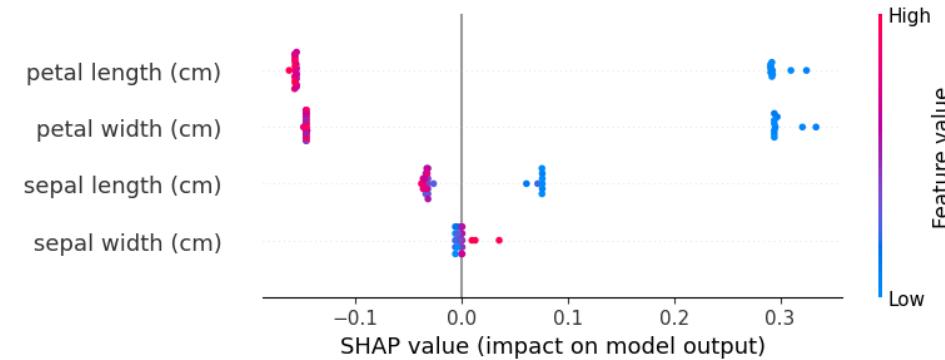
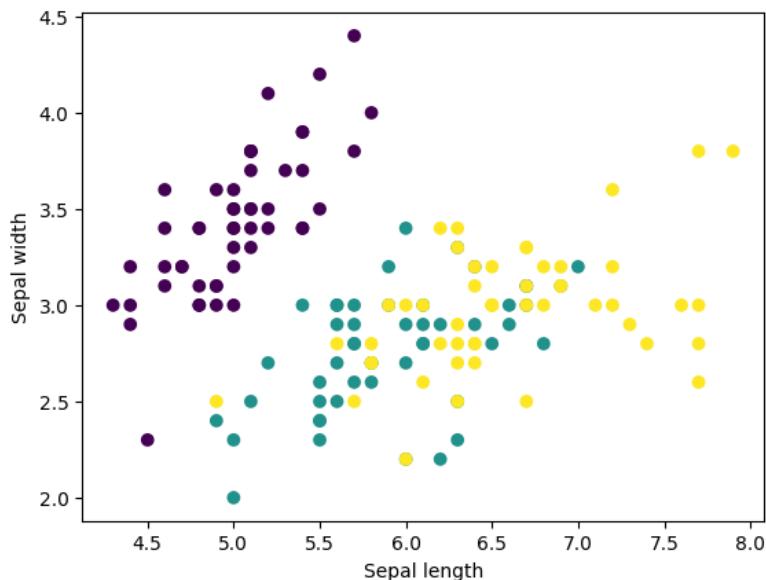
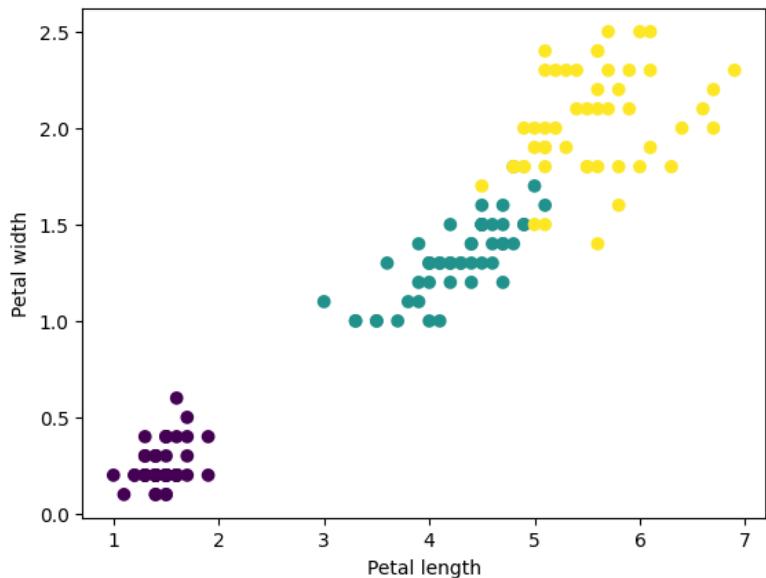
If two parties bring the same things to the coalition, they should have to contribute the same amount and should be rewarded for their contributions.

3. Dummy player has zero values

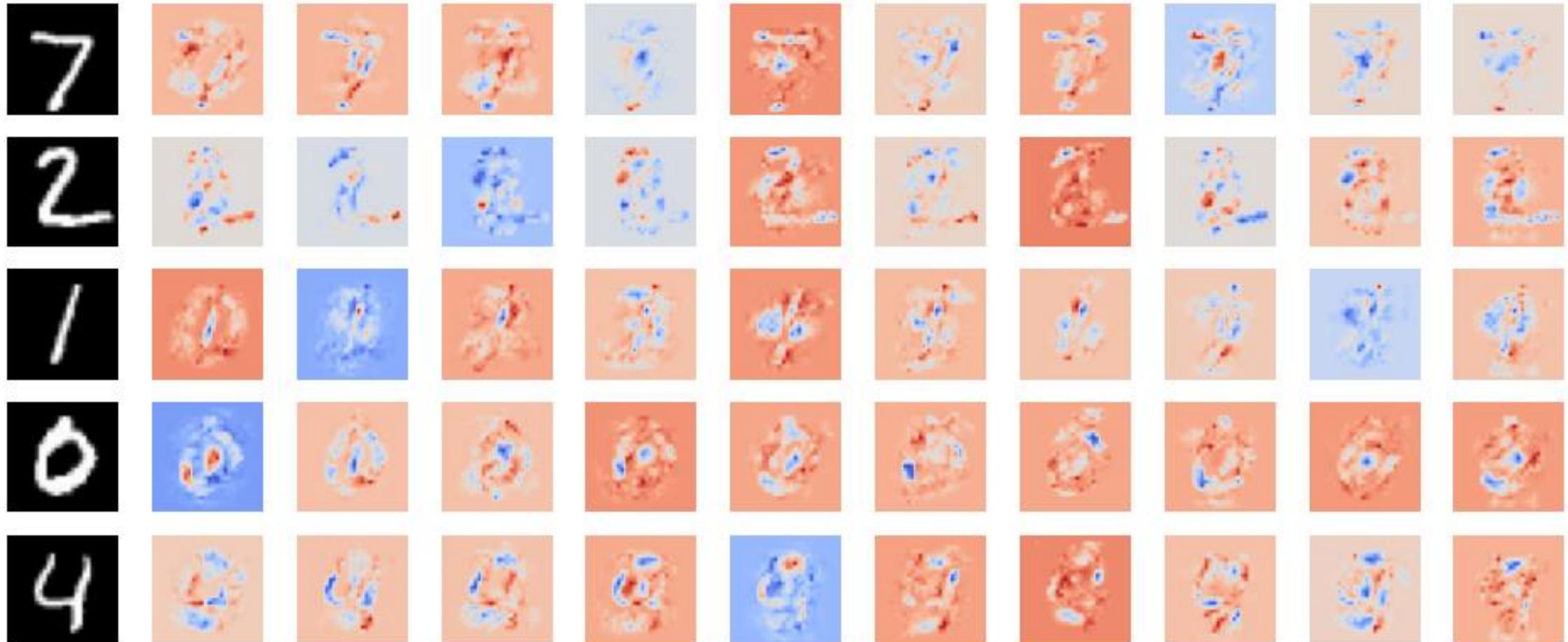
If a member of the coalition contributes nothing, then they should receive nothing. But it might not be fair in all cases, let us take an example of this thing more clear:

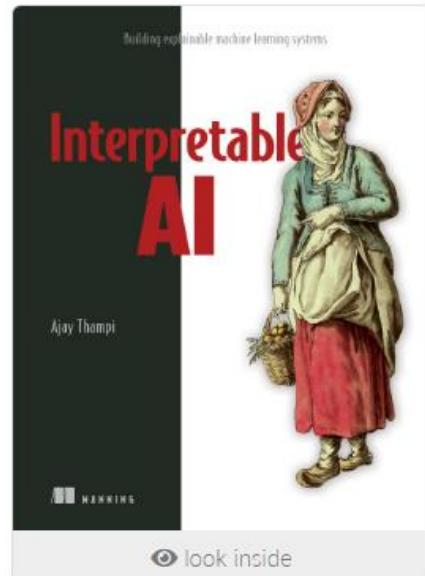
<https://abhishhek-maheshwarappa.medium.com/shap-values-for-explainable-ai-58652645d881>

SHAP: Shapley Additive exPlanations



SHAP for Images



[look inside](#)**Transparent and understandable AI systems***read this article now in
Manning's Free Content Center*[resources](#)[Source code](#)[Book Forum](#)[more](#)

Interpretable AI you own this product

Building explainable machine learning systems

★★★★★ 6 reviews

Ajay Thampi

May 2022 · ISBN 9781617297649 · 328 pages · printed in black & white

Data

eBook
pdf, ePub, online

print
includes eBook

subscription
from \$19.99

AI doesn't have to be a black box. These practical techniques help shine a light on your model's mysterious inner workings. Make your AI more transparent, and you'll improve trust in your results, combat data leakage and bias, and ensure compliance with legal requirements.

In *Interpretable AI*, you will learn:

- Why AI models are hard to interpret
- Interpreting white box models such as linear regression, decision trees, and generalized additive models
- Partial dependence plots, LIME, SHAP and Anchors, and other techniques such as saliency mapping, network dissection, and representational learning
- What fairness is and how to mitigate bias in AI systems
- Implement robust AI systems that are GDPR-compliant

eBook

~~\$47.99~~ **\$31.19**

you save \$16.80 (35%)

[add to cart](#)

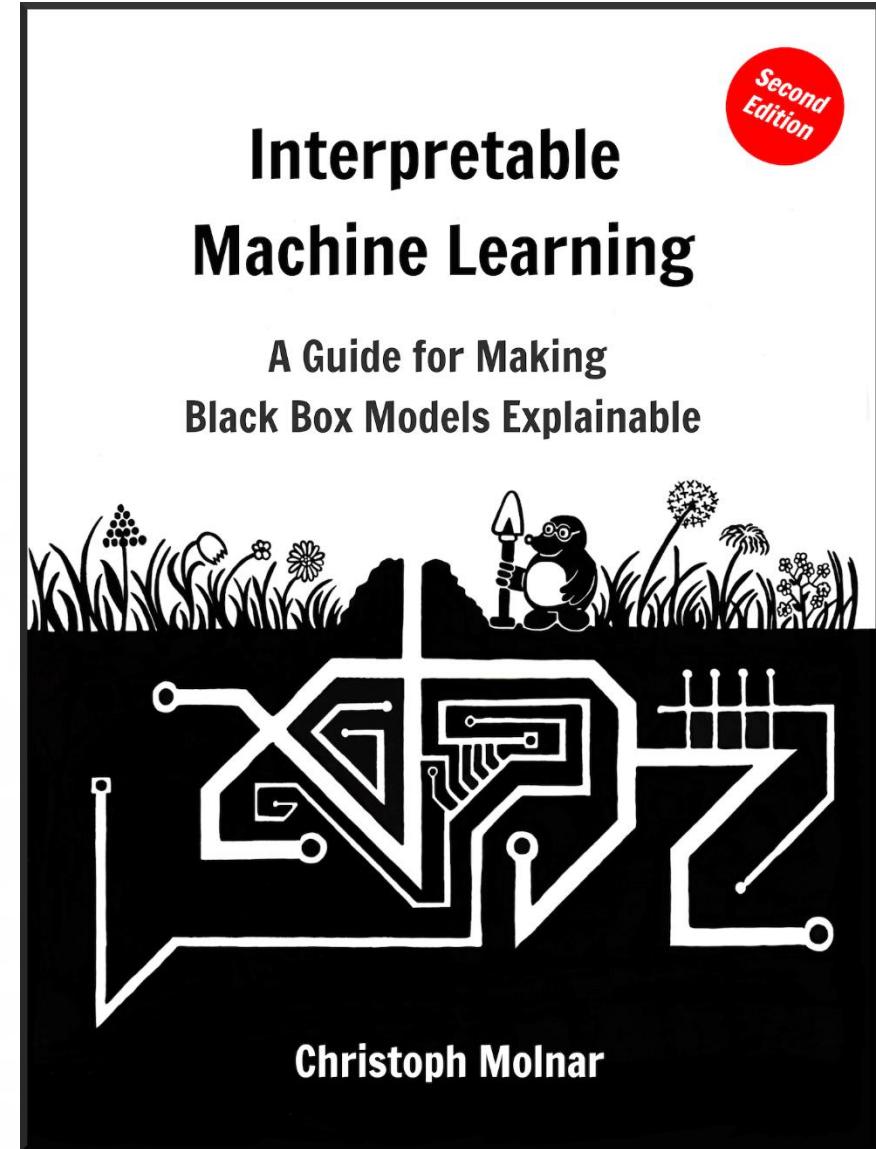
[buy now](#)

[free with subscription](#)



A sound introduction for practitioners to the exciting field of interpretable AI.

Pablo Roccatagliata, Torcuato Di Tella University



<https://christophm.github.io/interpretable-ml-book/>

Machine learning has great potential for improving products, processes and research. But **computers usually do not explain their predictions** which is a barrier to the adoption of machine learning. This book is about making machine learning models and their decisions interpretable.

After exploring the concepts of interpretability, you will learn about simple, **interpretable models** such as decision trees, decision rules and linear regression. The focus of the book is on model-agnostic methods for **interpreting black box models** such as feature importance and accumulated local effects, and explaining individual predictions with Shapley values and LIME. In addition, the book presents methods specific to deep neural networks.

All interpretation methods are explained in depth and discussed critically. How do they work under the hood? What are their strengths and weaknesses? How can their outputs be interpreted? This book will enable you to select and correctly apply the interpretation method that is most suitable for your machine learning project. Reading the book is recommended for machine learning practitioners, data scientists, statisticians, and anyone else interested in making machine learning models interpretable.

Colab: Interpretable ML