

# CS131 Review Section #3

Lectures 14-20

# Roadmap

- Lecture 14 (Visual Bag of Words):
  - Visual Bag of Words
  - Spatial Pyramid Matching
  - Naive Bayes
- Lecture 15 (Detecting Objects by Parts):
  - Object detection in General
  - Sliding Window + Spatial Pyramid
  - Deformable Parts Model
- Lecture 16 (Ontology)
- Lecture 17 (Motion)
  - Optical Flow
  - Lucas-Kanade & Iterative Lucas-Kanade & ILK + Spatial Pyramid
  - Horn-Schunck Method

# Roadmap

- Lecture 18 (Tracking):
  - Simple KLT Tracker
  - 2D Transformations
  - Iterative KLT Tracker
- Lecture 19 (Deep Learning):
  - Perceptron & Linear Classifier
  - Loss Function
  - Gradient Descent & Back-Prop
  - Neural Networks
- Lecture 20 (Convolutional Neural Networks):
  - Convolutional Filter
  - Architecture Design

# Lecture 14 - Visual Bag of Words

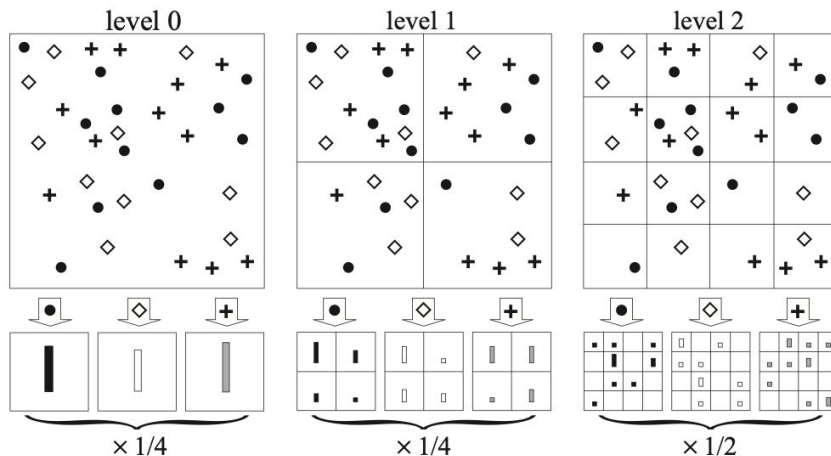
- A way to represent images (or other visual modalities, e.g., spatial-temporal words for videos)
- Procedure
  - Extract features from a training set of images
  - Build visual vocabulary and weight the words
  - Quantize the test image into histograms of visual words
- Feature extraction methods
  - Regular grid
  - Interest point
- Visual vocabulary construction - clustering
  - K-means
- Quantization - nearest neighbor search

# Lecture 14 - Spatial Pyramid Matching

- A popular way of organizing images for better image representation to account for different granularity. Agnostic to specific representation method.
- Idea: split images into a hierarchy of resolutions

- Visual bag of words + pyramids:

- Split an image into N levels with different resolutions
- For each level, compute its corresponding histograms
- Fuse different levels by weighting



Lazebnik et al., 2006

# Lecture 14 - Naive Bayes

- A classifier not only used for vision
- Goal: classify images using their posterior probability:

$$c^* = \arg \max_c P(c | \mathbf{x})$$

- Idea: leverage Bayes Theorem:

$$P(c | \mathbf{x}) = \frac{P(c) P(\mathbf{x} | c)}{\sum_{c'} P(c') P(\mathbf{x} | c')}$$

Where  $P(c)$  is the prior probability on classes

# Lecture 14 - Naive Bayes

- Combining visual bag of words:

$$P(c | \mathbf{x}) = \frac{P(c) \prod_{i=1}^m P(x_i | c)}{\sum_{c'} P(c') \prod_{i=1}^m P(x_i | c')}$$

Where  $P(\mathbf{x} | c) = \prod_{i=1}^m P(x_i | c)$  under the assumption that visual words are conditionally independent given the class.  $m$  is the number of words in the vocabulary.

- As we only need the argmax, we can ignore the denominator and turn the multiplication into summation by taking the log:

$$\log P(c | \mathbf{x}) \propto \log P(c) + \sum_{i=1}^m \log P(x_i | c)$$
$$c^* = \arg \max_c \log P(c) + \sum_{i=1}^m \log P(x_i | c)$$

# Lecture 15 - Object Detection in General

- Recognition & Classification & Detection

- Recognition (general) == Classification
- Detection = Localization + Classification

- Precision & recall

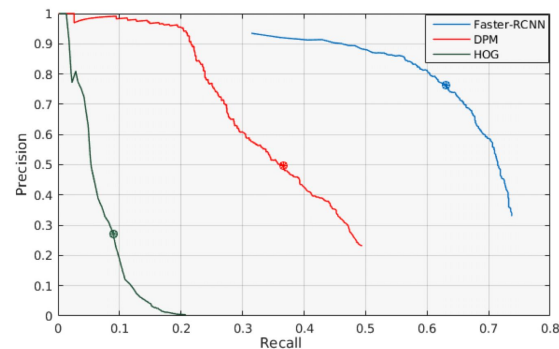
$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

- IoU

$$\frac{\text{Area of Intersection}}{\text{Area of Union}}$$

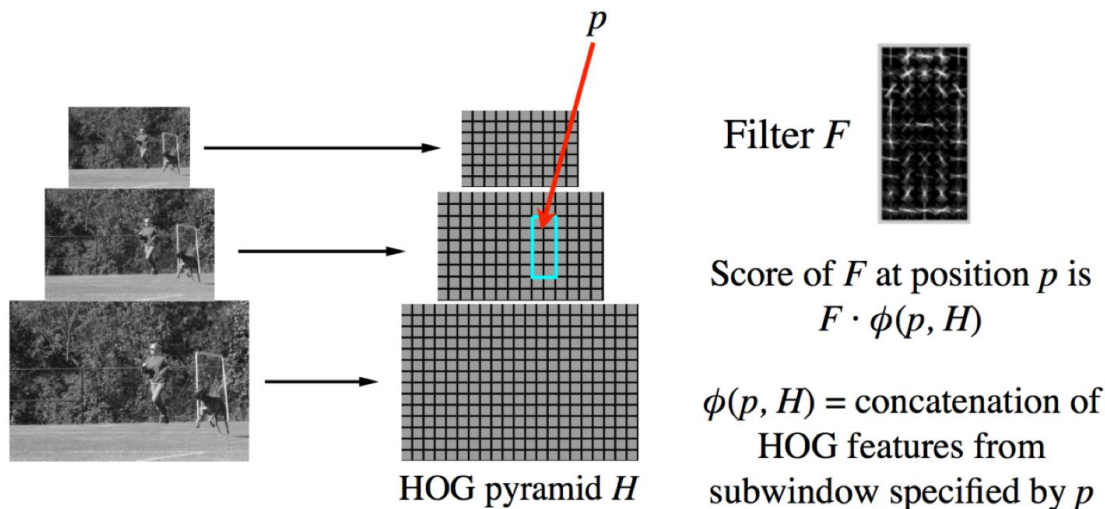
	<u>Predicted 1</u>	<u>Predicted 0</u>
<u>True 1</u>	true positive	false negative
<u>True 0</u>	false positive	true negative





# Lecture 15 - Sliding Window + Spatial Pyramid

- Sliding window: simply slide a fix-sized window across the image to see if there's potential object in each location
- Spatial pyramid: to take care of objects of different sizes



# Lecture 15 - Deformable Parts Model

- Idea: rather than detecting the entire object, detect the more generalizable parts instead
- A deformable parts model can be decomposed as:
  - A model for an object with  $n$  parts is a  $(n + 2)$  tuple:

$$(F_0, P_1, \dots, P_n, b)$$

Diagram illustrating the components of the tuple  $(F_0, P_1, \dots, P_n, b)$ :

- $F_0$ : Root filter (indicated by a red arrow from the text "Root filter" to  $F_0$ )
- $P_1$ : Model for 1st part (indicated by a red arrow from the text "Model for 1st part" to  $P_1$ )
- $b$ : Bias term (indicated by a red arrow from the text "Bias term" to  $b$ )

- Each part-based model defined as:

$$(F_i, v_i, d_i)$$

$F_i$  filter for the  $i$ -th part

$v_i$  "anchor" position for part  $i$  relative to the root position

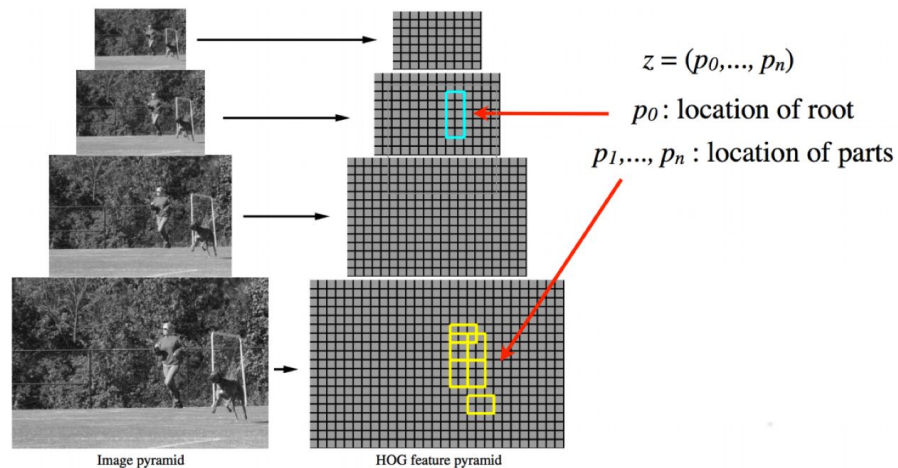
$d_i$  defines a deformation cost for each possible placement of the part relative to the anchor position



# Lecture 15 - Deformable Parts Model

- In order to detect parts with different spatial sizes, we again need a spatial pyramid

$p_i = (x_i, y_i, l_i)$  specifies the level and position of the  $i$ -th filter




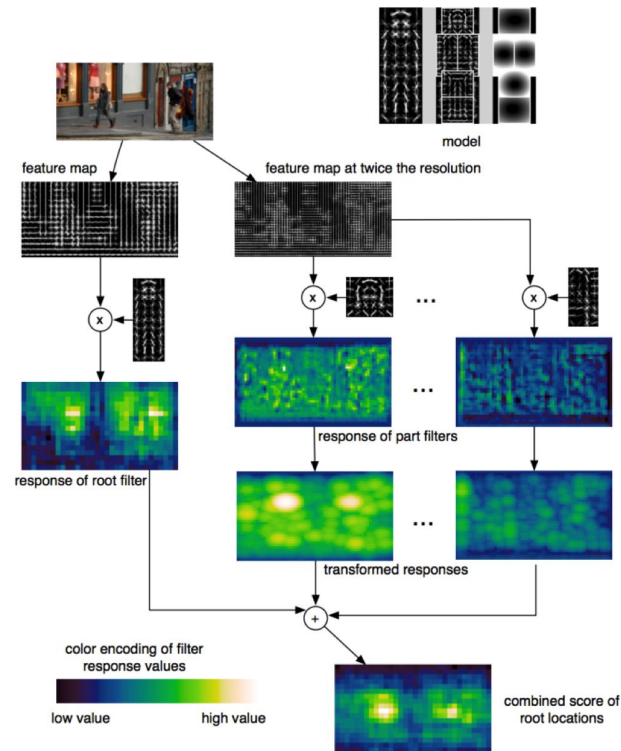
# Lecture 15 - Deformable Parts Model

- The score for the detection (for the entire object) is defined as the sum of scores for the global and part detectors minus the sum of deformation costs (which measures the how much each part drifts away from its expected position) for each part:

$$\begin{aligned} & \textit{detection score} \\ &= \sum_{i=0}^n F_i \phi(p_i, H) - \sum_{i=1}^n d_i(\Delta x_i, \Delta y_i, \Delta x_i^2, \Delta y_i^2) \end{aligned}$$

# Lecture 15 - Deformable Parts Model

- The entire detection pipeline 
- DPM pros & cons:
  - Pros:
    - Intuitive
    - Agnostic to specific detection method
    - Works especially well for certain classes
  - Cons:
    - Parts need to be selected manually
    - Each part needs a separately trained detector
    - The model needs to be re-built for each new class



# Lecture 17 - Optical Flow

- Captures the **apparent motion** of brightness patterns
- Estimate optical flow:

- Small motion: for Taylor expansion to work

- Brightness constancy:

$$I(x, y, t) = I(x + u(x, y), y + v(x, y), t + 1) \rightarrow I_x \cdot u + I_y \cdot v + I_t \approx 0 \rightarrow \nabla I \cdot [u \ v]^T + I_t = 0$$

- Spatial coherence:

$$\begin{bmatrix} I_x(\mathbf{p}_1) & I_y(\mathbf{p}_1) \\ I_x(\mathbf{p}_2) & I_y(\mathbf{p}_2) \\ \vdots & \vdots \\ I_x(\mathbf{p}_{25}) & I_y(\mathbf{p}_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(\mathbf{p}_1) \\ I_t(\mathbf{p}_2) \\ \vdots \\ I_t(\mathbf{p}_{25}) \end{bmatrix}$$

# Lecture 17 - Lucas-Kanade

- Lucas-Kanade

- $$\underbrace{\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}}_{A^T A} \begin{bmatrix} u \\ v \end{bmatrix} = - \underbrace{\begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}}_{A^T b}$$

Where summations are done over all pixels within the window

- Conditions for it to be solvable:

- **$A^T A$**  should be invertible
    - **$A^T A$**  should not be too small due to noise
      - eigenvalues  $\lambda_1$  and  $\lambda_2$  of  **$A^T A$**  should not be too small
    - **$A^T A$**  should be well-conditioned
      - $\lambda_1 / \lambda_2$  should not be too large ( $\lambda_1$  = larger eigenvalue)

# Lecture 17 - Iterative Lucas-Kanade

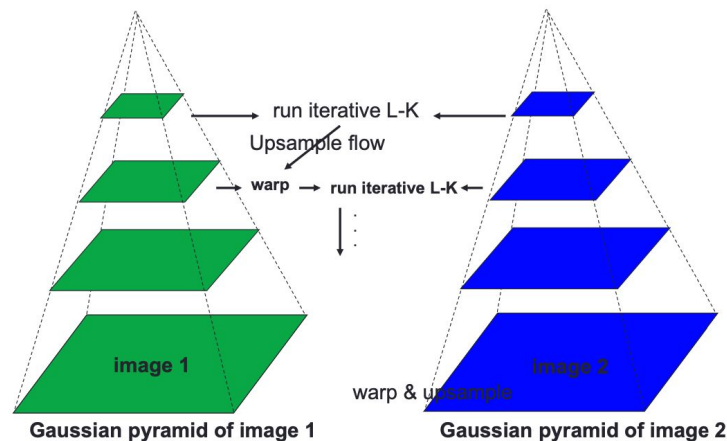
- Iterative Lucas-Kanade:
  - Add in higher order terms in Taylor expansion to find a better solution
  - Procedure:
    - Estimate optical flow vectors using the vanilla Lucas-Kanade
    - Warp towards the next time step using this estimated vector field
    - Repeat until the result is good enough



# Lecture 17 - ILC + Spatial Pyramid

- Iterative Lucas-Kanade + Spatial Pyramid:

- Comes in handy when the small motion assumption doesn't hold
- Procedure:
  - Run ILK starting from the most coarse level
  - Upsample the estimated flow vectors
  - Warp toward the next finer grained level
  - Repeat until the most fine-grained level



# Lecture 17 - Horn-Schunck

- Horn-Schunck method:

- Aims at smooth flow across the image
- Minimizes the following energy function:

$$E(u, v) = \int |I_2(\mathbf{p} + \mathbf{w}) - I_1(\mathbf{p})|^2 + \lambda(|\nabla u|^2 + |\nabla v|^2) d\mathbf{p}$$

Or

$$E = \iint [(I_x u + I_y v + I_t)^2 + \alpha^2 (\|\nabla u\|^2 + \|\nabla v\|^2)] dx dy$$

Specifically,  $\mathbf{p}$  denotes the keypoint we are tracking,  $\mathbf{w}$  denotes the optical flow vector of it.  
 $\lambda$  /  $\alpha$  is the weight used to balance the two parts.

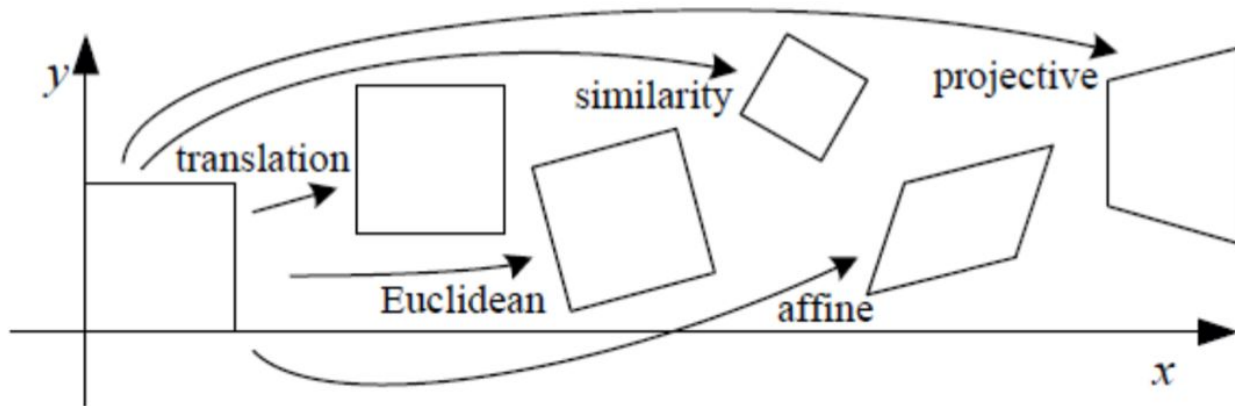
- It consists of two parts, where
  - First: smoothness between the pixel value along the flow vector (brightness constancy)
  - Second: smoothness among pixels spatially

# Lecture 18 - Simple KLT Tracker

- Procedure
  - Extract feature points for tracking (e.g., Harris)
  - Compute motion vectors for each point
  - Link these vectors in the successive frames to track the points
  - To prevent very large accumulated tracking error, detect new feature points after certain time steps
  - Repeat

# Lecture 18 - 2D Transformations

- Types:
  - Translation
  - Similarity Motion
  - Affine Motion
  - Euclidean Motion: Affine Motion with Length & Angle Preserved



# Lecture 18 - 2D Transformations

- Translation:

$$W(\mathbf{x}; \mathbf{p}) = \begin{bmatrix} 1 & 0 & b_1 \\ 0 & 1 & b_2 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Similarity Motion:

$$W(\mathbf{x}; \mathbf{p}) = \begin{bmatrix} a & 0 & b_1 \\ 0 & a & b_2 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Affine Motion:

$$W(\mathbf{x}; \mathbf{p}) = \begin{bmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- $\mathbf{p}$  denotes the parameters, i.e.,  $a$  and  $b$ .

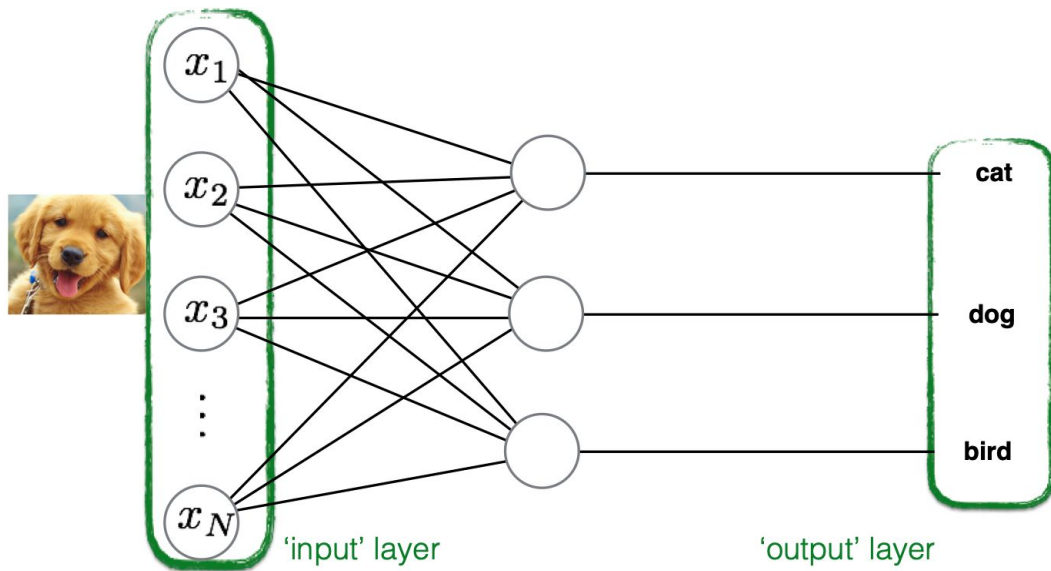
# Lecture 18 - Iterative KLT Tracker

Given the features from Harris detector:

1. Initialize  $\mathbf{p}_0$  and  $\Delta\mathbf{p}$ .
2. Compute the initial templates  $T(x)$  for each feature.
3. Transform the features in the image  $I$  with  $W(\mathbf{x}; \mathbf{p}_0)$ .
4. Measure the error:  $I(W(\mathbf{x}; \mathbf{p}_0)) - T(x)$ .
5. Compute the image gradients  $\nabla I = [I_x \ I_y]$ .
6. Evaluate the Jacobian  $\frac{\partial W}{\partial \mathbf{p}}$ .
7. Compute steepest descent  $\nabla I \frac{\partial W}{\partial \mathbf{p}}$ .
8. Compute Inverse Hessian  $H^{-1}$
9. Calculate the change in parameters  $\Delta\mathbf{p}$  
$$\Delta\mathbf{p} = H^{-1} \sum_x \left[ \nabla I \frac{\partial W}{\partial \mathbf{p}} \right]^T [T(x) - I(W(\mathbf{x}; \mathbf{p}_0))]$$
10. Update parameters  $\mathbf{p}_0 = \mathbf{p}_0 + \Delta\mathbf{p}$
11. Repeat 2 to 10 until  $\Delta\mathbf{p}$  is small.

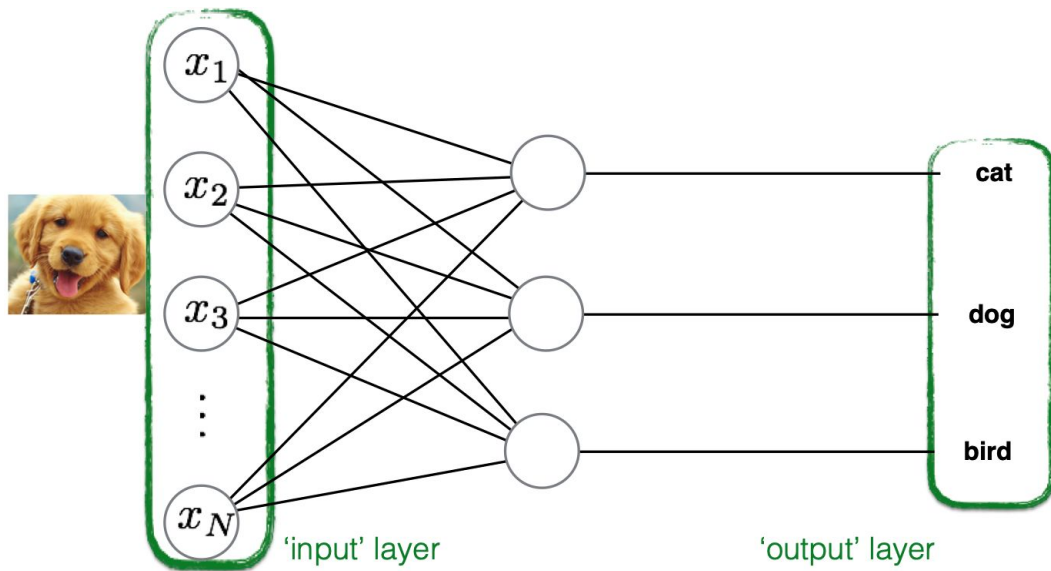
# Lecture 19 - Perceptron & Linear Classifier

- Essentially doing a polynomial mapping from input to the output, without non-linear transformations
- Essentially is a fully-connected neural net without non-linear activation functions
- $f(x, W) = Wx$   
W is the trainable weight matrix
- We can interpret the weights for linear models as ?



# Lecture 19 - Perceptron & Linear Classifier

- Essentially doing a polynomial mapping from input to the output, without non-linear transformations
- Essentially is a fully-connected neural net without non-linear activation functions
- $f(x, W) = Wx$   
W is the trainable weight matrix
- We can interpret the weights for linear models as **templates**





# Lecture 19 - Loss Function

- Measures how much the prediction of our model deviates from the ground-truth

$$L = \frac{1}{N} \sum_{i=1}^N L_i(y_i, \hat{y}_i)$$

Where  $L_i$  is the individual loss for data sample  $i$ .

- Popular loss functions:
  - L2 loss:  $L_i(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$
  - L1 loss:  $L_i(y_i, \hat{y}_i) = |y_i - \hat{y}_i|$
  - Hinge loss:  $L_i(y_i, \hat{y}_i) = \max(0, 1 - y_i \hat{y}_i)$
  - Cross-entropy loss
  - ...

# Lecture 19 - Softmax Classifier

- Softmax function: a **normalization** function that squashes input to the range of  $[0, 1]$ , basically turning the input to probability space. It thus is often used as the output function of classifiers.

$$Prob[f(x_i, W) == k] = \frac{e^{\hat{y}_k}}{\sum_j e^{\hat{y}_j}}$$

- KL divergence:
  - A loss function used to measure the distance between two probability distributions
  - In practice, the two compared distributions are often being the predicted distribution and the ground-truth distribution

$$D_{KL} = \sum_y P(y) \log \frac{P(y)}{Q(y)}$$

# Lecture 19 - Gradient Descent & Back-Prop

- Optimization: ways to find the optimal parameter set  $W^*$
- Gradient descent:
  - A popular optimization method (arguably the most popular as for machine learning?)
  - Update rule:

$$W := W - \alpha \frac{dL}{dW}$$

- Back-prop:
  - Used to get the gradients across a deep network (e.g., a multi-layer perceptron)
  - Procedure:
    - First do forward prop to calculate the loss and intermediate values
    - Then compute gradients for each step through the backward prop
  - Practice (see blackboard)

# Lecture 19 - Neural Networks

- Essentially multi-layer perceptron + non-linear activation function
- Activation function:

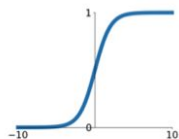
- Why is it necessary?

- Popular choices:

- Sigmoid
- Tanh
- ReLU
- Leaky ReLU
- Maxout
- ELU
- ...

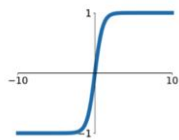
**Sigmoid**

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



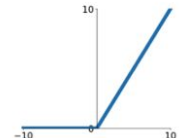
**tanh**

$$\tanh(x)$$



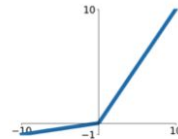
**ReLU**

$$\max(0, x)$$



**Leaky ReLU**

$$\max(0.1x, x)$$

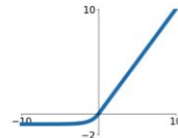


**Maxout**

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

**ELU**

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

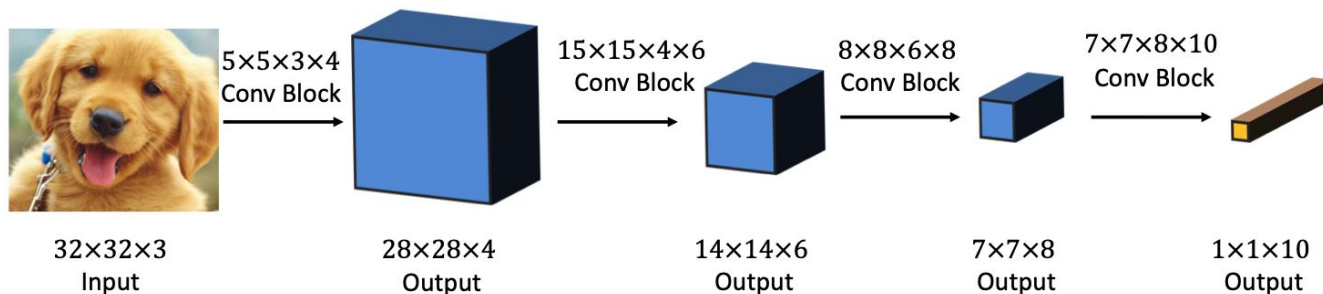


# Lecture 20 - Convolutional Filter

- Sliding window operation
- Output size calculation:

$$W_2 = \frac{W_1 - F + 2P}{S} + 1, H_2 = \frac{H_1 - F + 2P}{S} + 1$$

- Practice: how many parameters (excluding bias) in the first conv block?



- Q: how does CNN incorporate spatial information?

# Lecture 20 - Architecture Design

- Convolutional layer
- Non-linear activation function
- Pooling layer (effect?)

Thank you