Topics:

- Optimization (Cont)
- Imbalance
- Convolution

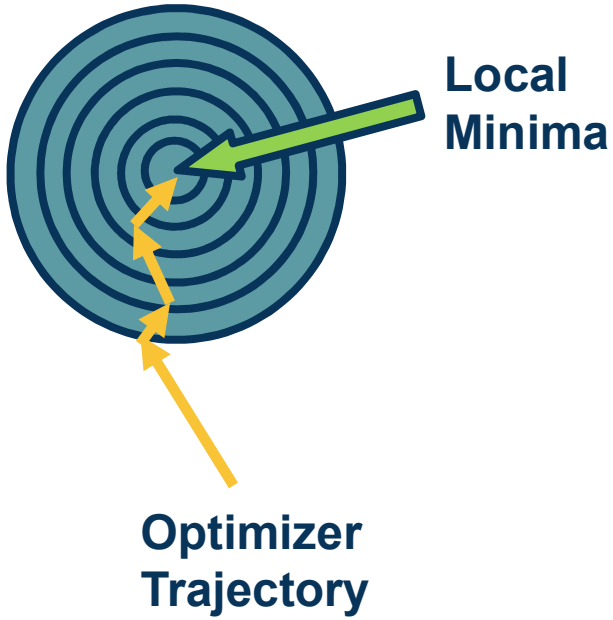
CS 4803-DL / 7643-A
ZSOLT KIRA

- **Assignment 2**
  - Implement convolutional neural networks

- **Facebook Lectures**: Data wrangling video available online
  - See dropbox link piazza @8 and M1L4 folder
  - Opportunity to talk to them Wed. 02/17 4-5pm

Even given a good neural network architecture, we need a **good optimization algorithm to find good weights**

- What **optimizer** should we use?
  - Different optimizers make **different weight updates** depending on the gradients
- How should we **initialize** the weights?
- What **regularizers** should we use?
- What **loss function** is appropriate?



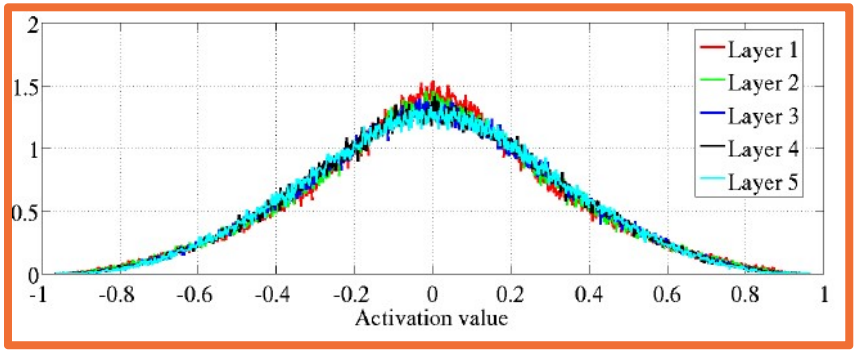
Optimization Considerations

Ideally, we'd like to maintain the variance at the output to be similar to that of input!

- This condition leads to a **simple initialization rule**, sampling from uniform distribution:

$$\text{Uniform}\left(-\frac{\sqrt{6}}{n_j+n_{j+1}}, +\frac{\sqrt{6}}{n_j+n_{j+1}}\right)$$

  - Where n_j is **fan-in** (number of input nodes) and n_{j+1} is **fan-out** (number of output nodes)



Distribution of activation values of a network with tanh non-linearities, for increasingly deep layers

From "Understanding the difficulty of training deep feedforward neural networks." AISTATS, **2010.**

- We can give the model flexibility through **learnable parameters γ (scale) and β (shift)**
- Network can learn to **not normalize** if necessary!
- This layer is called a **Batch Normalization (BN) layer**

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

From: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, Sergey Ioffe, Christian Szegedy

Learnable Scaling and Offset

Solution: Time-varying bias correction

Typically $\boldsymbol{\beta_1 = 0.9,\ \ \beta_2 = 0.999}$

So $\hat{v}_i$ will be small number divided by (1-0.9=0.1) resulting in more reasonable values (and $\hat{G}_i$ larger)

$$v_i = \beta_1 v_{i-1} + (1-\beta_1)\left(\frac{\partial L}{\partial w_{i-1}}\right)$$

$$G_i = \beta_2 G_{i-1} + (1-\beta_2)\left(\frac{\partial L}{\partial w_{i-1}}\right)^2$$

$$\hat{v}_i = \frac{v_i}{1-\beta_1^t} \qquad \hat{G}_i = \frac{G_i}{1-\beta_2^t}$$

$$w_i = w_{i-1} - \frac{\alpha\, \hat{v}_i}{\sqrt{\hat{G}_i + \epsilon}}$$

Regularization

Many **standard regularization methods** still apply!

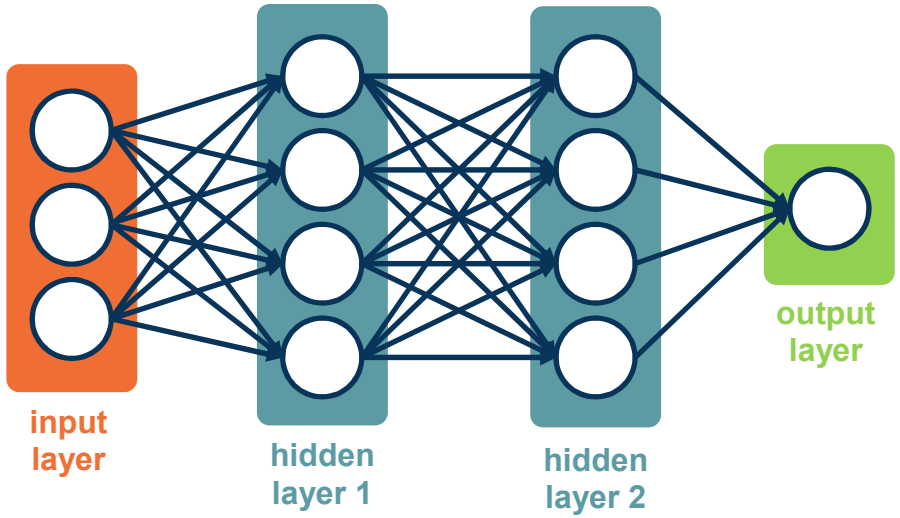
L1 Regularization

$$L = |y - Wx_i|^2 + \lambda|W|$$

where $|W|$ is element-wise

Example regularizations:

- L1/L2 on weights (encourage small values)
- L2: $L = |y - Wx_i|^2 + \lambda|W|^2$ (weight decay)
- Elastic L1/L2: $|y - Wx_i|^2 + \alpha|W|^2 + \beta|W|$

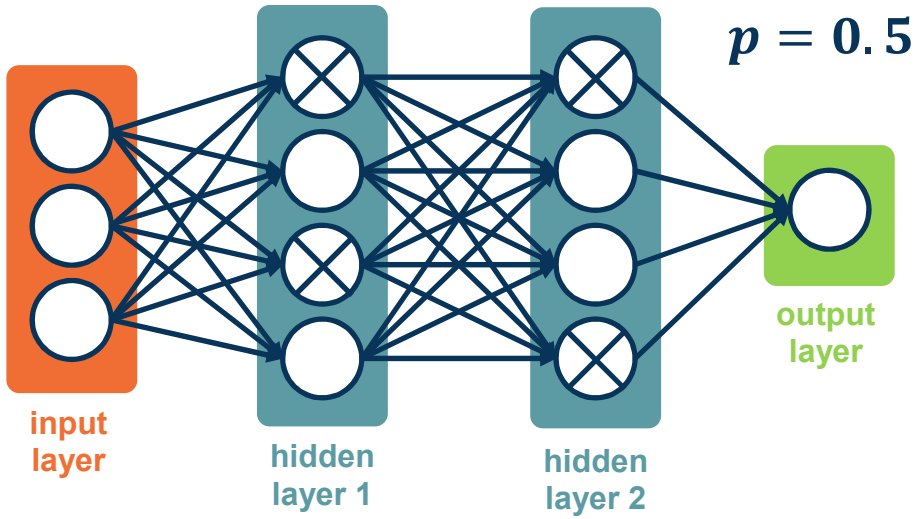


Problem: Network can learn to rely strong on a few features that work really well

- May cause **overfitting** if not representative of test data

From: Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Srivastava et al.

Preventing Co-Adapted Features



An idea: For each node, keep its output with probability p

- Activations of deactivated nodes are essentially zero

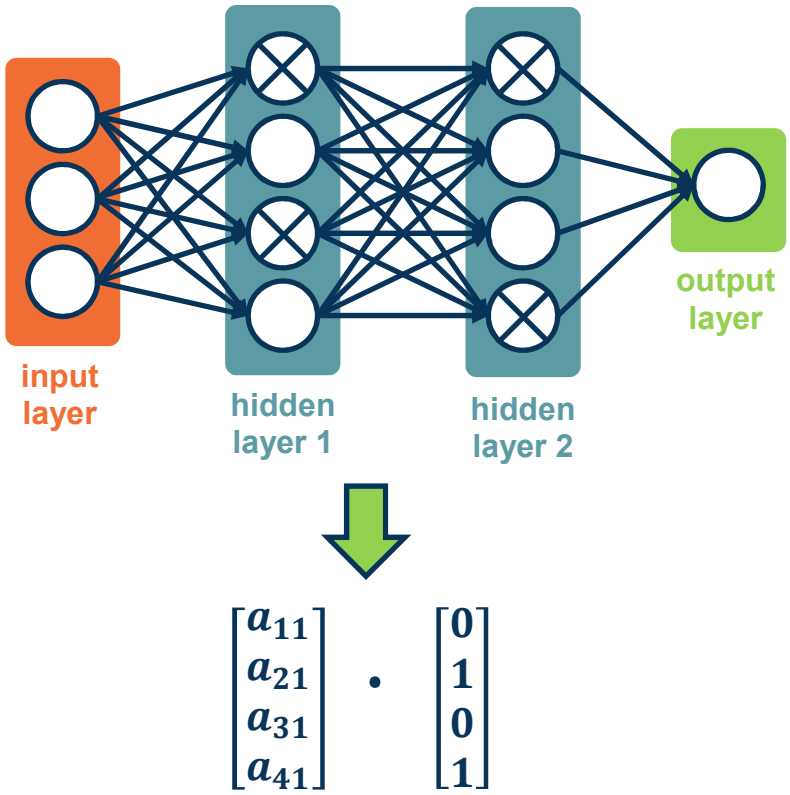
Choose whether to mask out a particular node **each iteration**

From: Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Srivastava et al.

Dropout Regularization

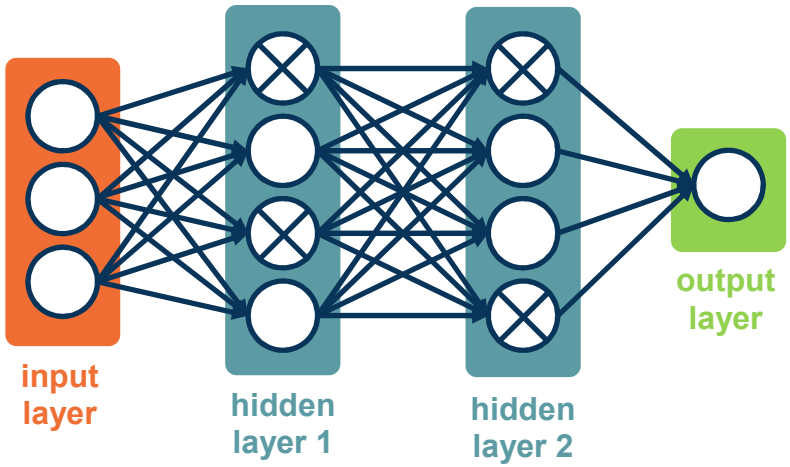
Dropout Implementation

- In practice, implement with a **mask** calculated each iteration
- During testing, no nodes are dropped



From: Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Srivastava et al.

- During training, each node has an expected $p * fan\_in$ nodes
- During test all nodes are activated
- **Principle:** Always try to have similar train and test-time input/output distributions!



Solution: During test time, scale outputs (or equivalently weights) by p

- i.e. $W_{test} = pW$

$$\begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ a_{41} \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

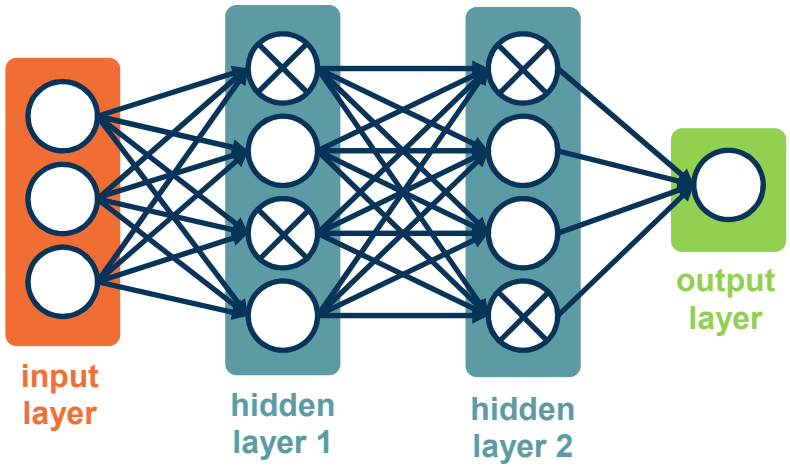
- Alternative: Scale by $\frac{1}{p}$ at train time

From: Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Srivastava et al.

Inference with Dropout

Interpretation 1: The model should not rely too heavily on particular features

- If it does, it has probability $\mathbf{1} - \boldsymbol{p}$ of losing that feature in an iteration



From: Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Srivastava et al.

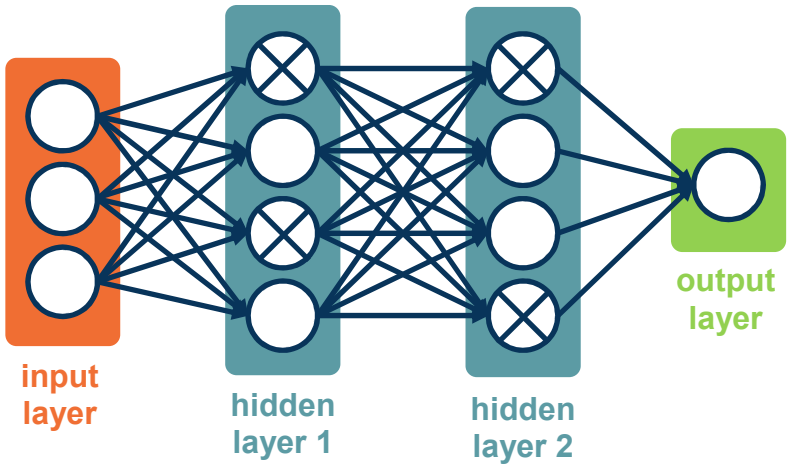
Why Dropout Works

Interpretation 1: The model should not rely too heavily on particular features

- If it does, it has probability $\mathbf{1} - \boldsymbol{p}$ of losing that feature in an iteration

Interpretation 2: Training $\mathbf{2}^{\boldsymbol{n}}$ networks:

- Each configuration is a network
- Most are trained with 1 or 2 mini-batches of data



From: Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Srivastava et al.

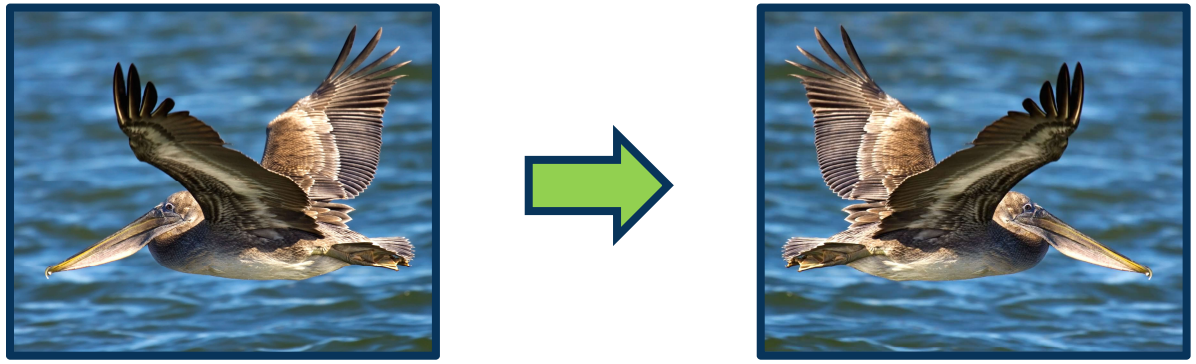
Why Dropout Works

Data Augmentation

Data augmentation – Performing a range of **transformations** to the data

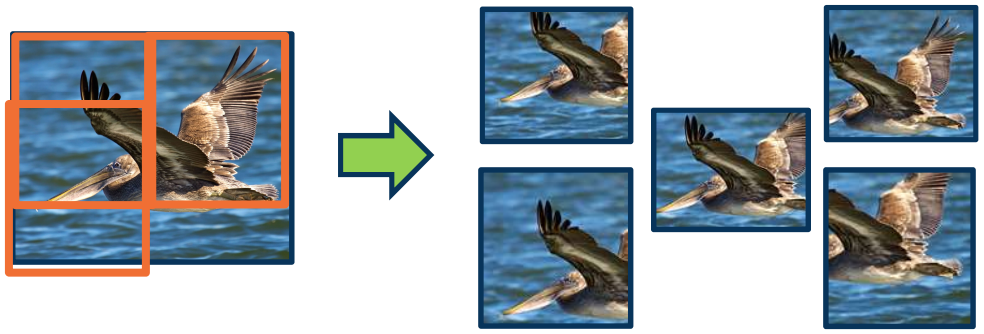
- This essentially **"increases"** your dataset
- Transformations should not change meaning of the data (or label has to be changed as well)

Simple example: Image Flipping



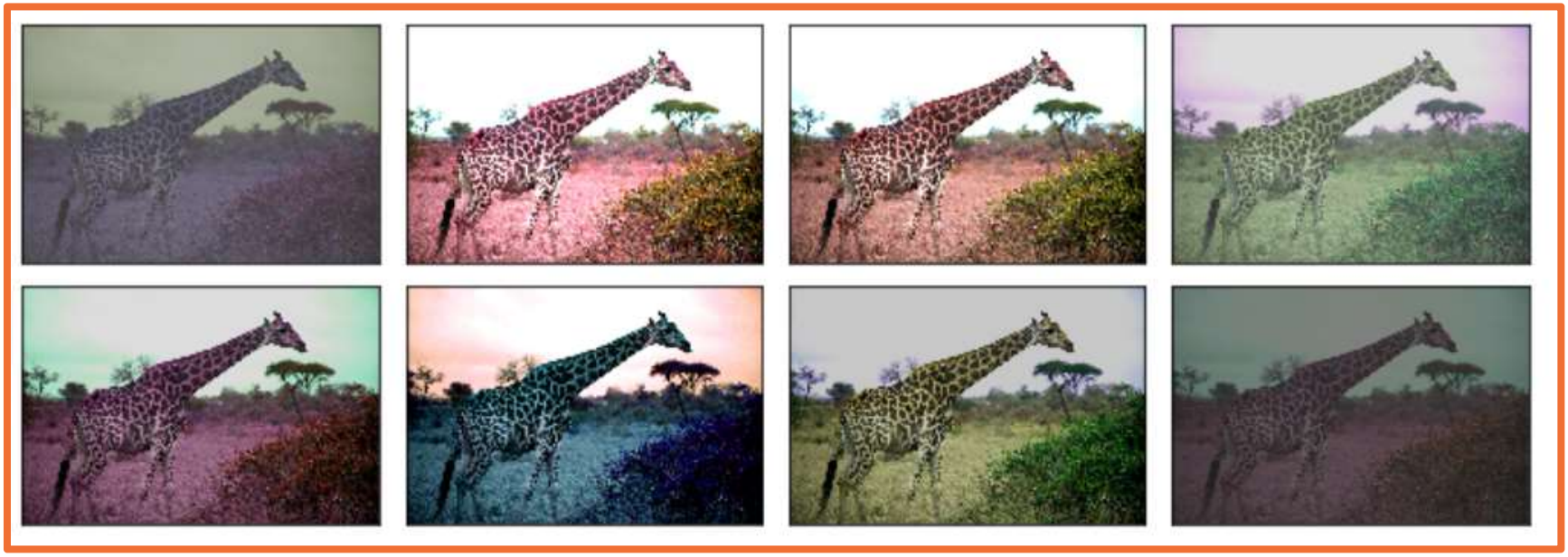
Random crop

- Take different crops during training
- Can be used during inference too!



CutMix

Color Jitter



From https://mxnet.apache.org/versions/1.5.0/tutorials/gluon/data_augmentation.html

Color Jitter

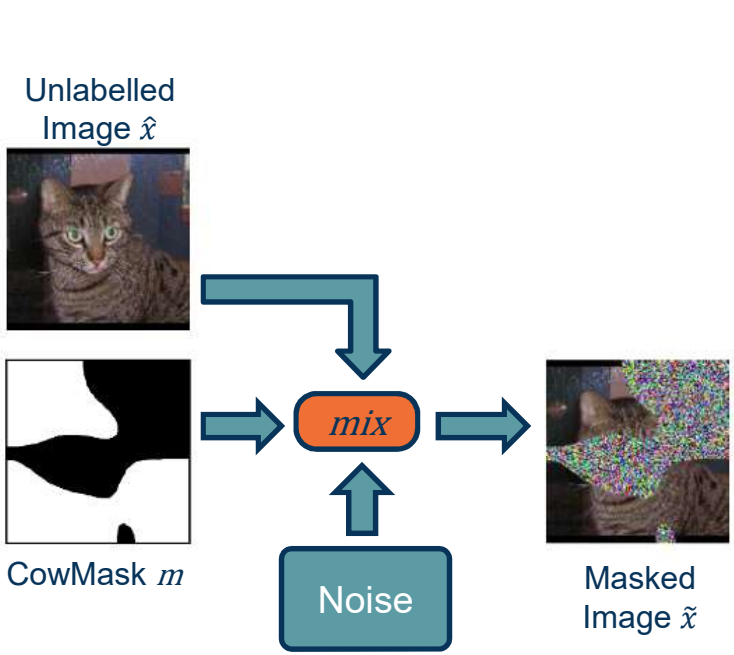
We can apply **generic affine transformations:**

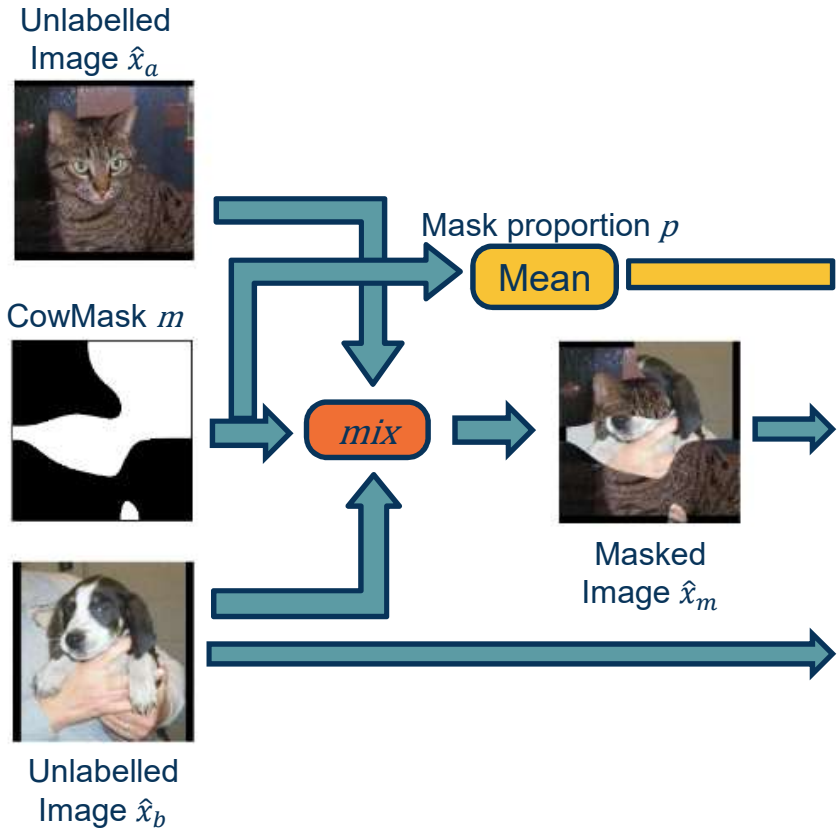
- **Translation**
- **Rotation**
- **Scale**
- **Shear**

We can **combine these transformations** to add even more variety!

From https://mxnet.apache.org/versions/1.5.0/tutorials/gluon/data_augmentation.html

Combining Transformations





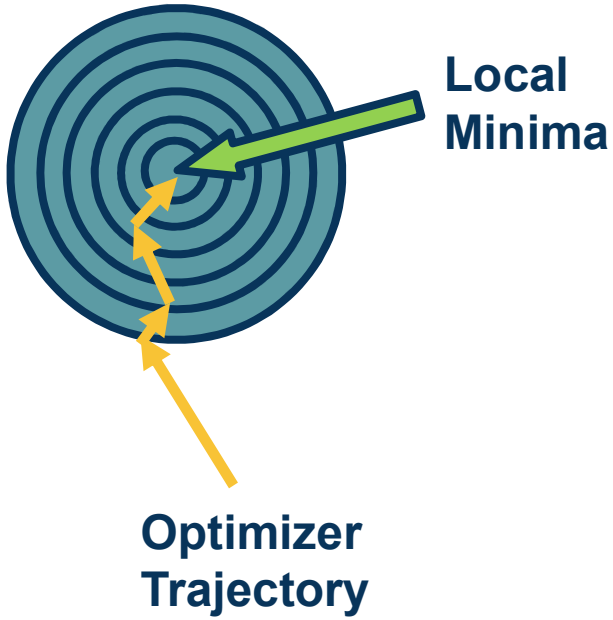
CowMix

From French et al., "Milking CowMask for Semi-Supervised Image Classification"

Other Variations

The Process of Training Neural Networks

- Training deep neural networks is an art form!
- Lots of things matter (together) – the key is to find a combination that works
- **Key principle**: Monitoring everything to understand what is going on!
  - Loss and accuracy curves
  - Gradient statistics/characteristics
  - Other aspects of computation graph



Proper Methodology

Always start with **proper methodology**!

- **Not uncommon** even in published papers to get this wrong

Separate data into: **Training, validation, test set**

- **Do not look** at test set performance until you have decided on everything (including hyper-parameters)

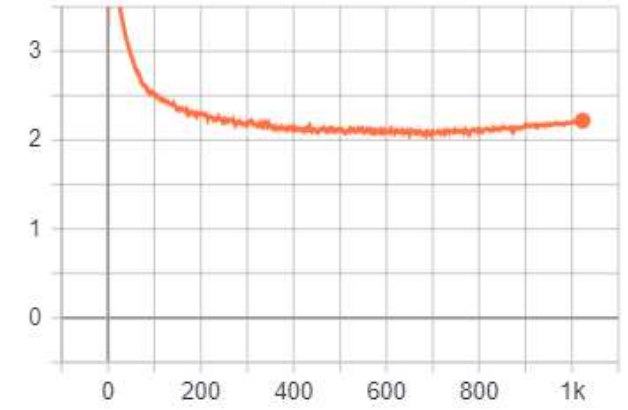
Use **cross-validation** to decide on hyper-parameters if amount of data is an issue

Check the bounds of your loss function

- E.g. cross-entropy ranges from $[\mathbf{0}, \infty]$
- Check initial loss at small random weight values
  - E.g. $-\mathbf{log}(\boldsymbol{p})$ for cross-entropy, where $\boldsymbol{p} = \mathbf{0.5}$

Another example: Start without regularization and make sure loss goes up when added

Key Principle: Simplify the dataset to make sure your model can properly (over)-fit before applying regularization



Validation Loss

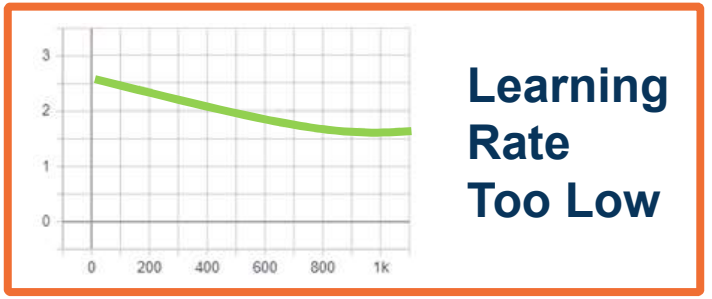
Change in loss indicates speed of learning:

- Tiny loss change -> too small of a learning rate
- Loss (and then weights) turn to NaNs -> too high of a learning rate

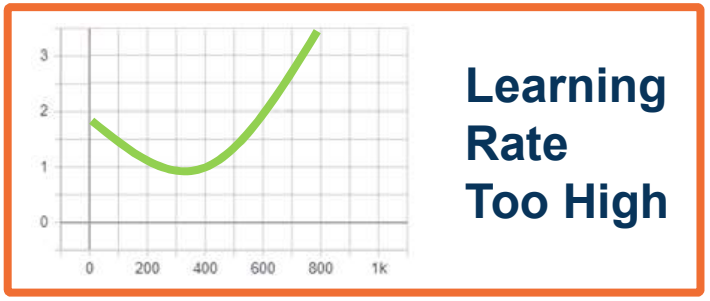
Other bugs can also cause this, e.g.:

- Divide by zero
- Forgetting the log!

In pytorch, use autograd's detect anomaly to debug

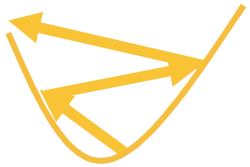


Learning Rate Too Low



Learning Rate Too High

```
with autograd.detect_anomaly():
  output = model(input)
  loss = criterion(output, labels
  loss.backward()
```

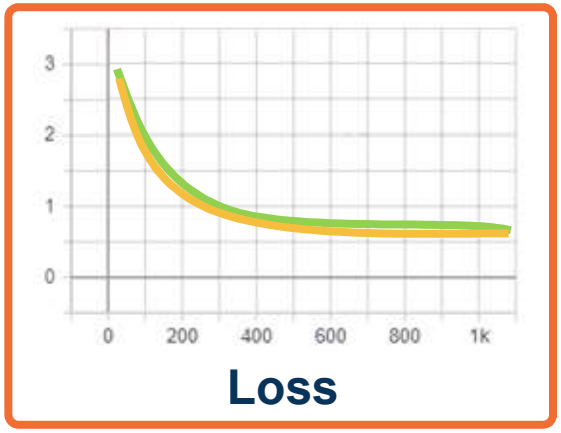


Loss and Not a Number (NaN)

Georgia Tech

Overfitting

- Classic machine learning signs of under/overfitting still apply!
- **Over-fitting**: Validation loss/accuracy starts to get worse after a while
- **Under-fitting:** Validation loss very close to training loss, or both are high
- **Note:** You can have higher training loss!
  - Validation loss has no regularization
  - Validation loss is typically measured at the end of an epoch

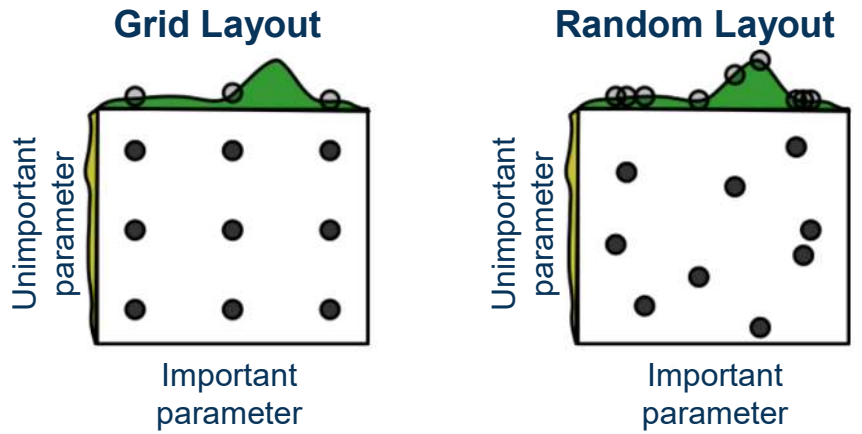


Many hyper-parameters to tune!

- Learning rate, weight decay crucial
- Momentum, others more stable
- **Always tune** hyper-parameters; even a good idea will fail un-tuned!

Start with coarser search:

- E.g. learning rate of {0.1, 0.05, 0.03, 0.01, 0.003, 0.001, 0.0005, 0.0001}
- Perform finer search around good values



From: Bergstra et al., "Random Search for Hyper-Parameter Optimization", JMLR, 2012

Automated methods are OK, but intuition (or random) can do well given enough of a tuning budget

Inter-dependence of Hyperparameters

Note that hyper-parameters and even module selection are **interdependent**!

Examples:

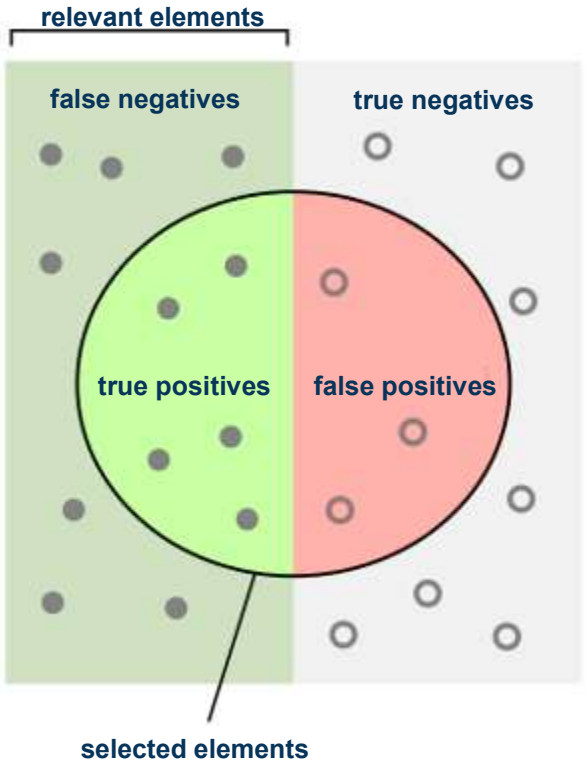
- Batch norm and dropout **maybe not be needed together** (and sometimes the combination is worse)
- The learning rate should be **changed proportionally to batch size** – increase the learning rate for larger batch sizes
  - **One interpretation:** Gradients are more reliable/smoother

Note that we are optimizing a **loss function**

What we actually care about is **typically different metrics that we can't differentiate**:

- Accuracy
- Precision/recall
- Other specialized metrics

The relationship between the two can be complex!



From https://en.wikipedia.org/wiki/Precision_and_recall

Relationship Between Loss and Other Metrics

Georgia Tech

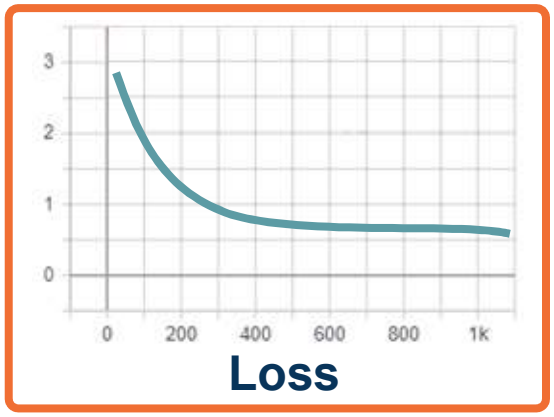
- **Example:** Cross entropy loss

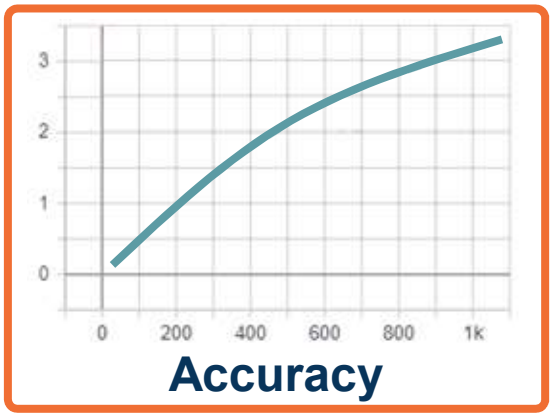
$$L = -log\ P(Y = y_i | X = x_i)$$

- **Accuracy** is measured based on:

$$argmax_i(P(Y = y_i\ | X = x_i))$$

- Since the correct class score only has to be slightly higher, we can have **flat loss curves but increasing accuracy**!

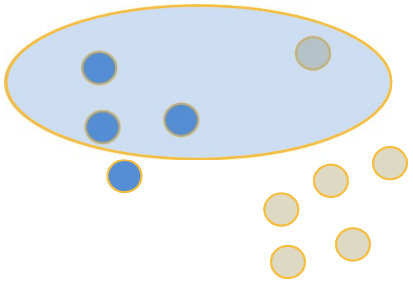


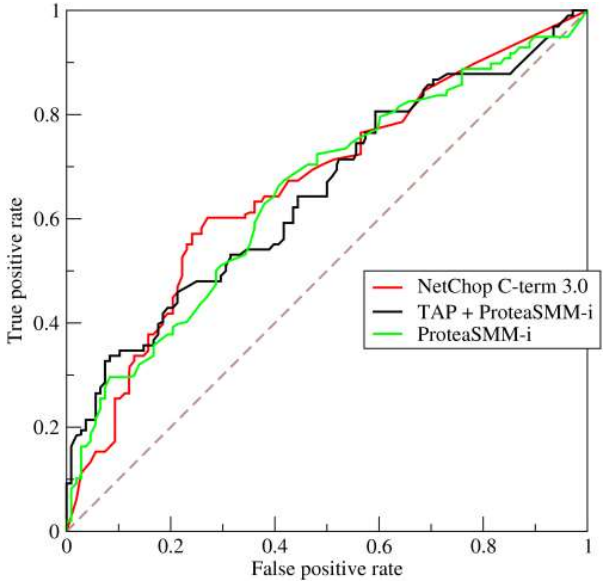


Simple Example: Cross-Entropy and Accuracy

Example: Precision/Recall or ROC Curves

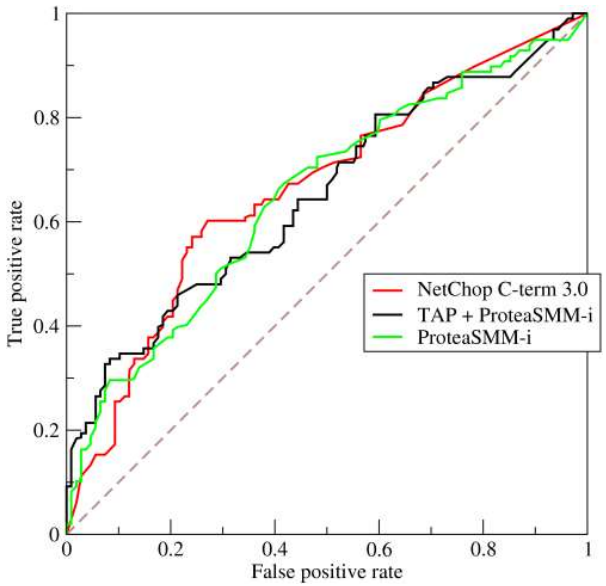
- **Precision/Recall curves** represent the inherent tradeoff between number of positive predictions and correctness of predictions
- **Definitions**
  - True Positive Rate: $\boldsymbol{TPR} = \frac{\boldsymbol{tp}}{\boldsymbol{tp+fn}}$
  - False Positive Rate: $\boldsymbol{FPR} = \frac{\boldsymbol{fp}}{\boldsymbol{fp+tn}}$
  - $\boldsymbol{Accuracy} = \frac{\boldsymbol{tp+tn}}{\boldsymbol{tp+tn+fp+fn}}$





From https://en.wikipedia.org/wiki/Receiver_operating_characteristic

- **Precision/Recall curves** represent the inherent tradeoff between number of positive predictions and correctness of predictions
- **Definitions**
  - True Positive Rate: $\boldsymbol{TPR} = \frac{\boldsymbol{tp}}{\boldsymbol{tp+fn}}$
  - False Positive Rate: $\boldsymbol{FPR} = \frac{\boldsymbol{fp}}{\boldsymbol{fp+tn}}$
  - $\boldsymbol{Accuracy} = \frac{\boldsymbol{tp+tn}}{\boldsymbol{tp+tn+fp+fn}}$
- We can obtain a **curve** by varying the (probability) threshold**:**
  - **Area under the curve (AUC)** common single-number metric to summarize
- Mapping between this and loss is **not simple**!

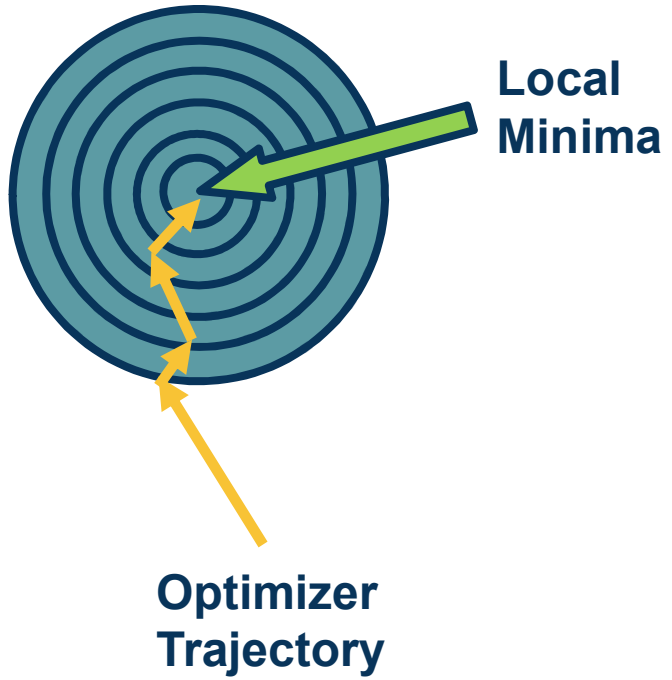


From https://en.wikipedia.org/wiki/Receiver_operating_characteristic

Example: Precision/Recall or ROC Curves

Resource:

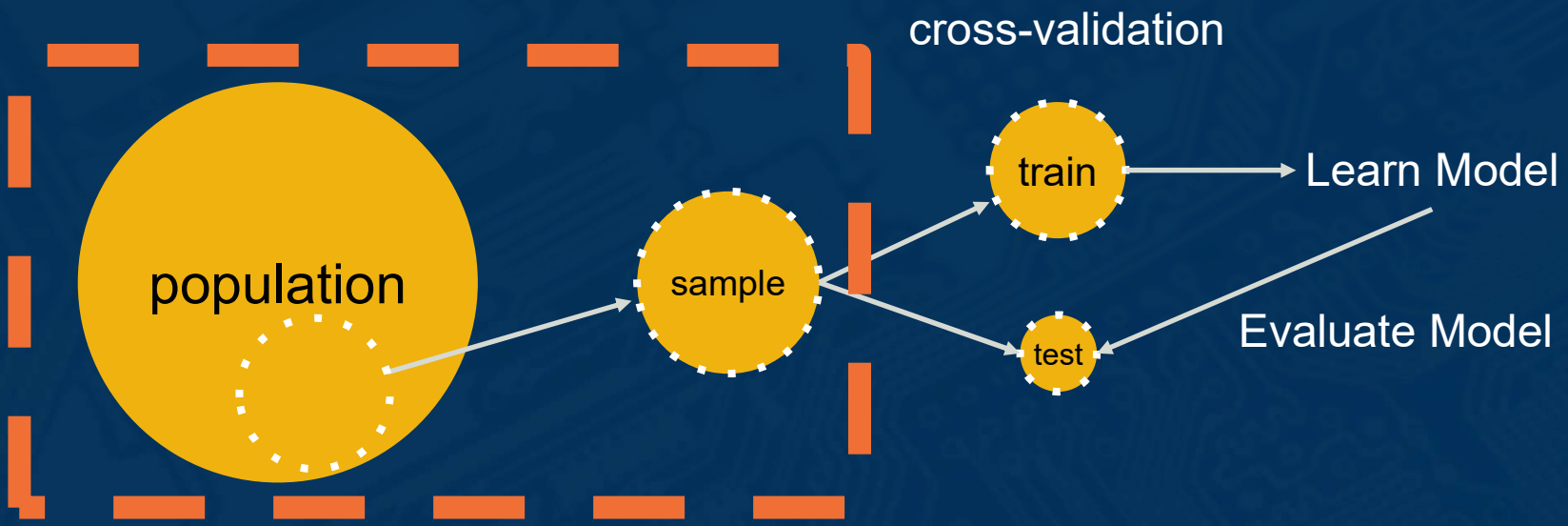
- A disciplined approach to neural network hyper-parameters: Part 1 -- learning rate, batch size, momentum, and weight decay, Leslie N. Smith



Cross-Validation & Class Imbalance

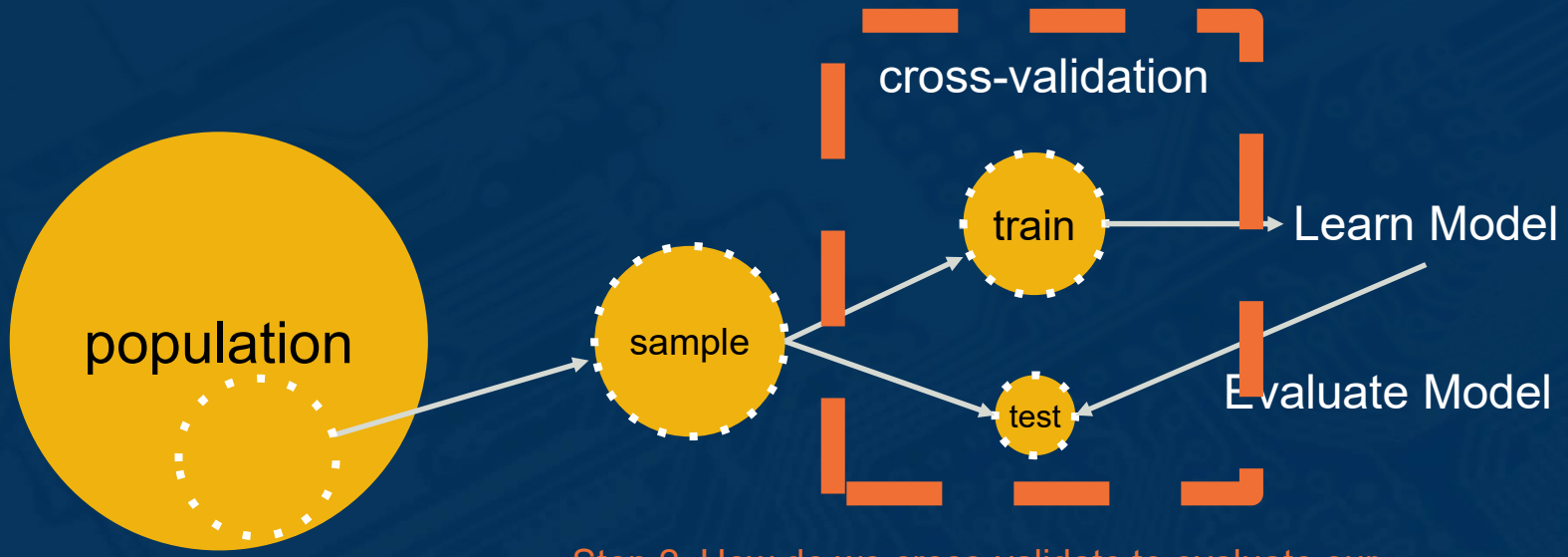
Georgia Tech

The Data Wrangling Process



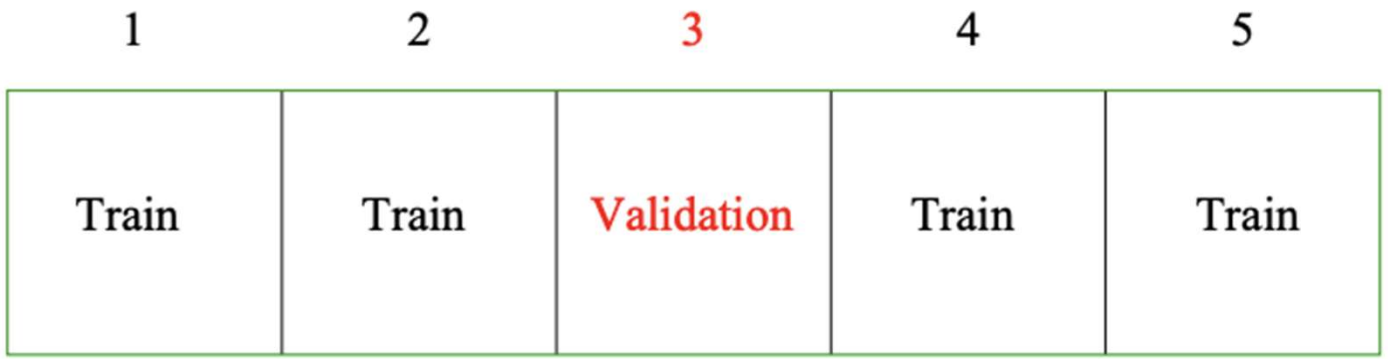
Step 1. What is the **population** of interest? What **sample S** are we evaluating, and is sample S **representative** of the population?

The Data Wrangling Process



Step 2. How do we cross-validate to evaluate our model? How do we avoid overfitting and data mining?

Cross-Validation

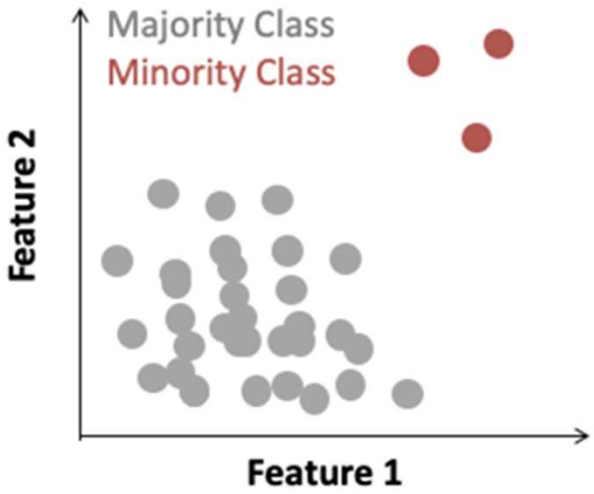


(Hastie et al., 2011)

Cross-Validation Best Practices

1. Random search vs. Grid Search for Hyperparameters (Bergstra and Bengio, 2012)
2. Confirm hyperparameter range is sufficient such as plotting out-of-bag (OOB) error rate
3. Temporal cross-validation considerations
4. Check for overfitting

Class Imbalance



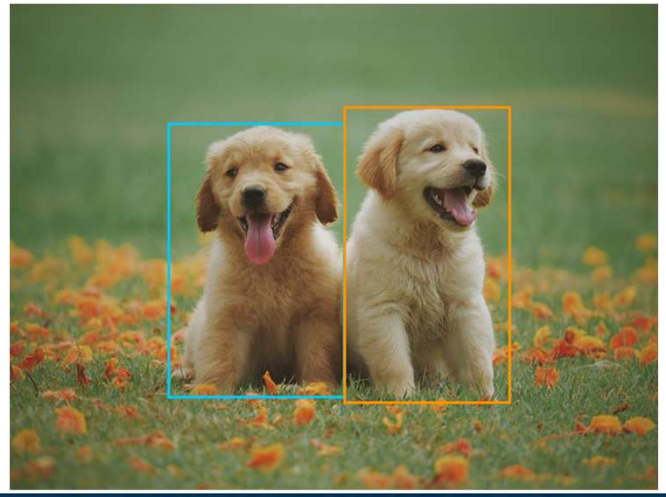
(Altenburger and Ho, *under review 2020*)

Class Imbalance

(Altenburger and Ho, *under review 2020*)

Object Detection

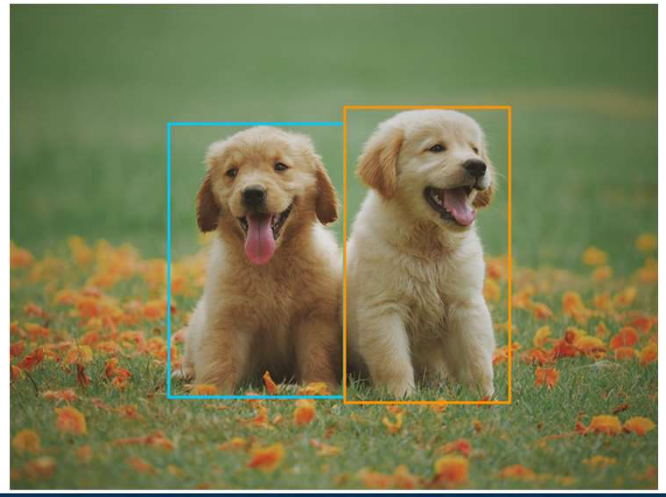
Region CNN (R-CNN) and Single Shot Detector (SSD) are models that can localize and classify many objects in an image



R-CNN: Girshick, SSD: Liu

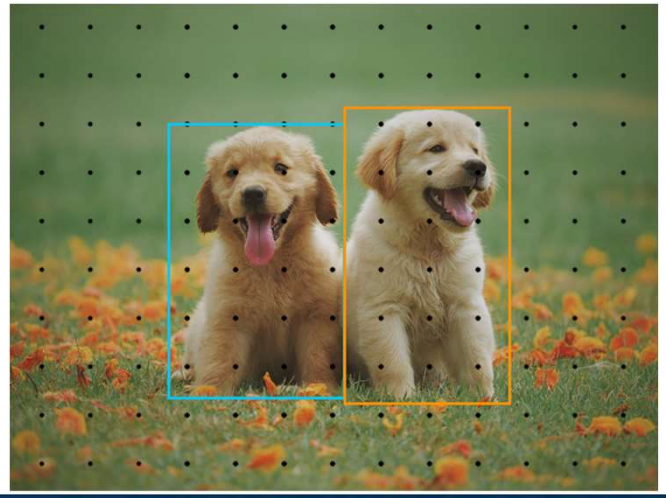
Class Imbalance: Object Detection

Object detection models (ex: R-CNN and SSD) densely sample many boxes of different sizes at different “anchor” locations in the image



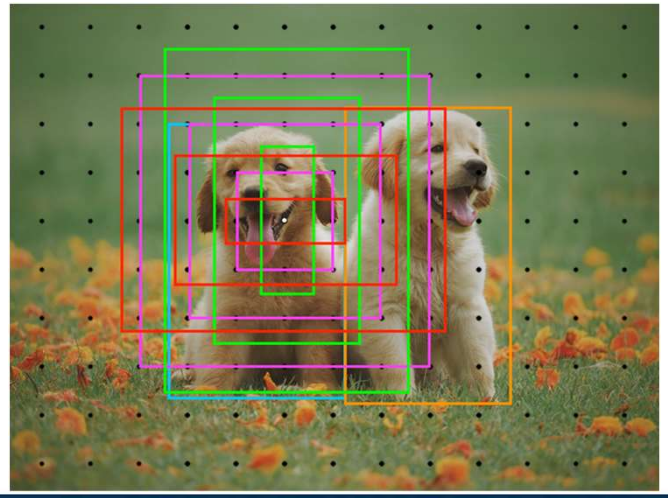
Class Imbalance: Object Detection

Object detection models (ex: R-CNN and SSD) densely sample many boxes of different sizes at different “anchor” locations in the image



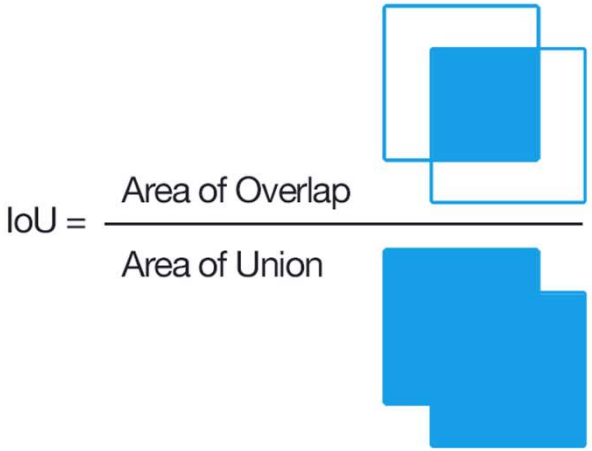
Class Imbalance: Object Detection

Object detection models (ex: R-CNN and SSD) densely sample many boxes of different sizes at different “anchor” locations in the image



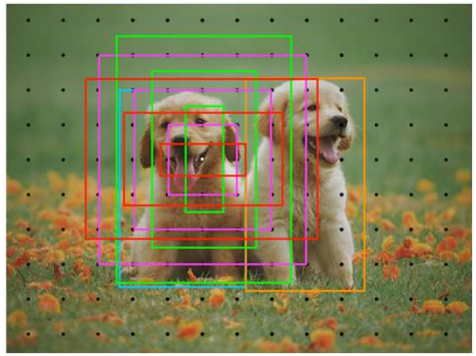
Class Imbalance: Object Detection

- Goal: Classify a proposal box into foreground or background
- IoU: intersection over union
- A proposal box is assigned a ground truth label of:
  - *Foreground*, if IoU with ground truth box > 0.5
  - *Background*, otherwise



Class Imbalance: Object Detection

foreground boxes >>> # background boxes!



Class Imbalance: Focal Loss

Cross Entropy: easy examples incur a non-negligible loss, which in aggregate mask out the harder, rare examples

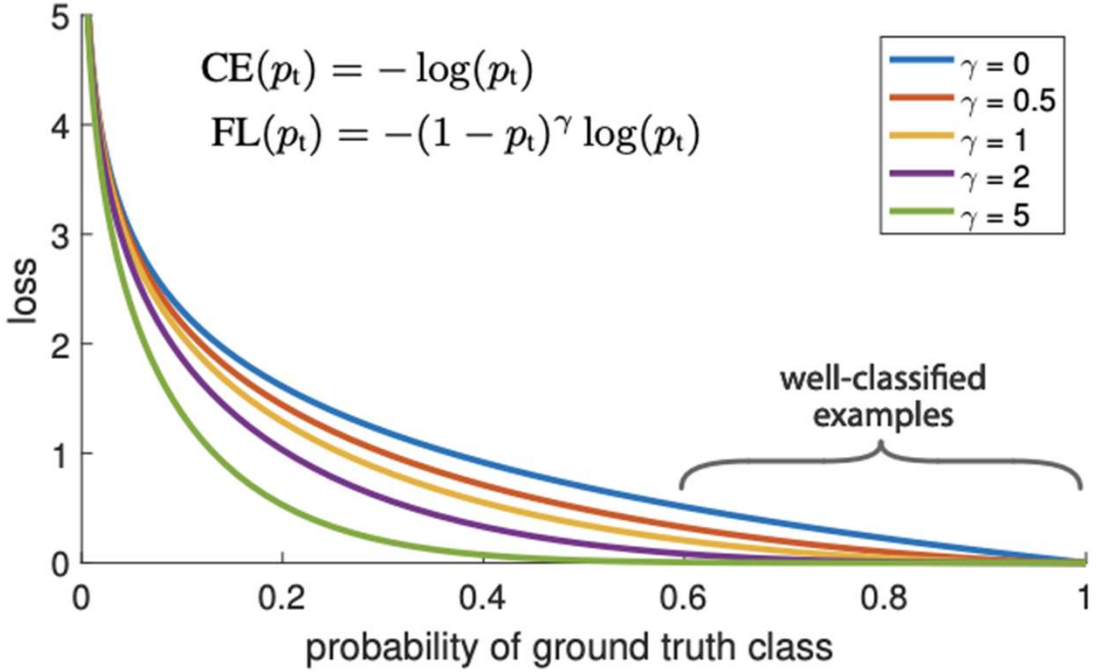
$$\mathrm{CE}(p, y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1-p) & \text{otherwise.} \end{cases}$$

Focal Loss: down-weights easy examples, to give more attention to difficult examples

$$\mathrm{FL}(p_t) = -(1-p_t)^{\gamma} \log(p_t).$$

(Lin et al., 2017)

Class Imbalance: Focal Loss

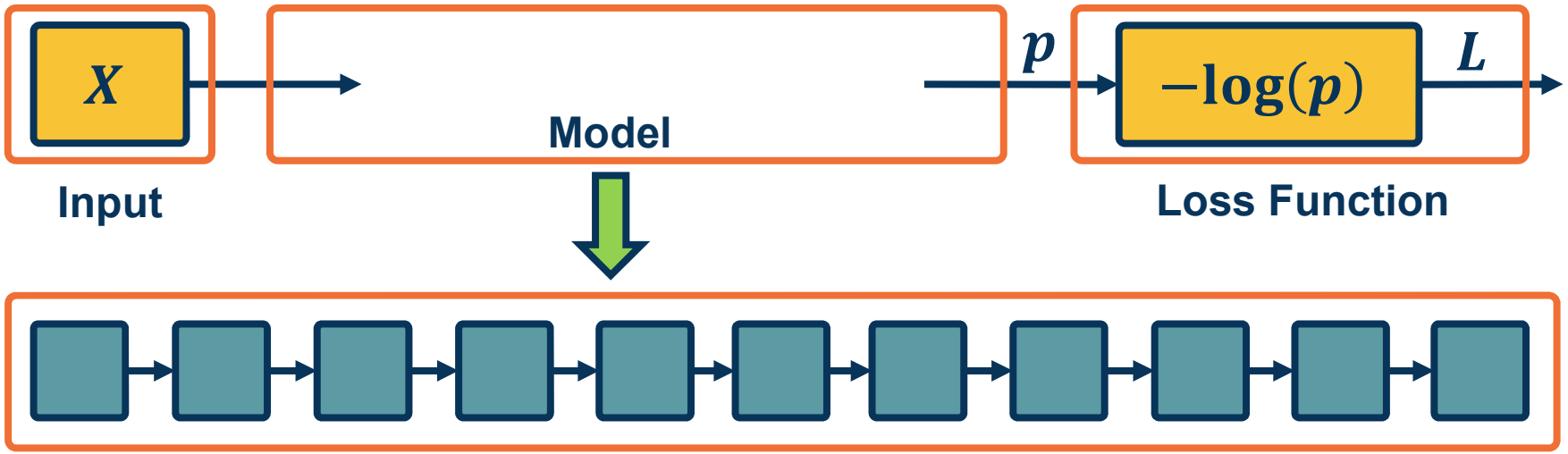


(Lin et al., 2017)

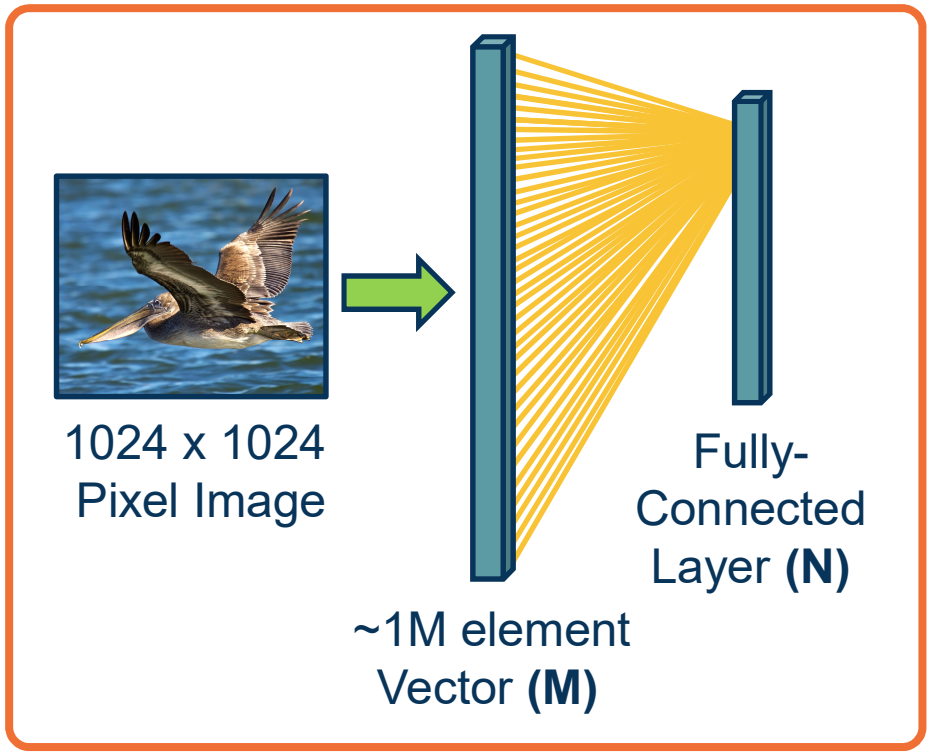
Convolution Layers

Backpropagation, and automatic differentiation, allows us to optimize **any** function composed of differentiable blocks

- **No need to modify** the learning algorithm!
- The complexity of the function is only limited by **computation and memory**



The connectivity in linear layers **doesn't always make sense**



How many parameters?

- M*N (weights) + N (bias)

Hundreds of millions of parameters **for just one layer**

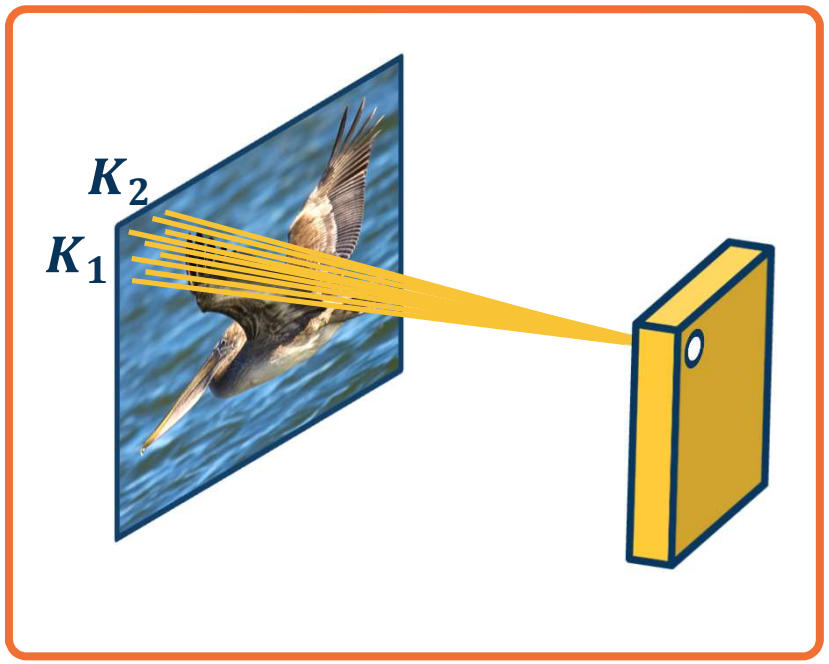
More parameters => More data needed

Is this necessary?

Image features are spatially localized!

- Smaller features repeated across the image
  - Edges
  - Color
  - Motifs (corners, etc.)
- No reason to believe one feature tends to appear in one location vs. another (stationarity)

Can we induce a *bias* in the design of a neural network layer to reflect this?



Each node only receives input from $K_1 \times K_2$ window (image patch)

- Region from which a node receives input from is called its **receptive field**

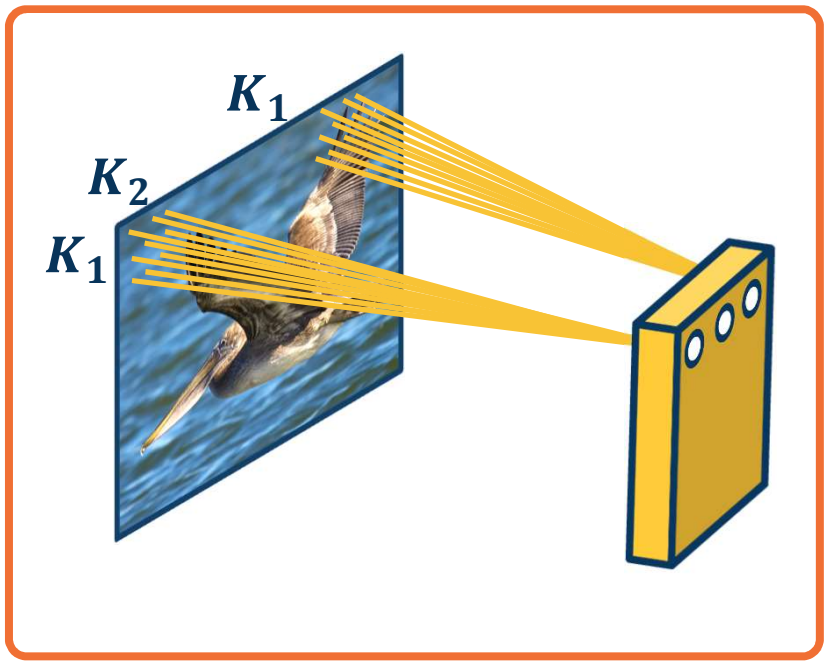
Advantages:

- Reduce parameters to $(K_1 \times K_2 + 1) * N$ where N is number of output nodes
- Explicitly maintain spatial information

Do we need to learn location-specific features?

Idea 1: Receptive Fields

Georgia Tech



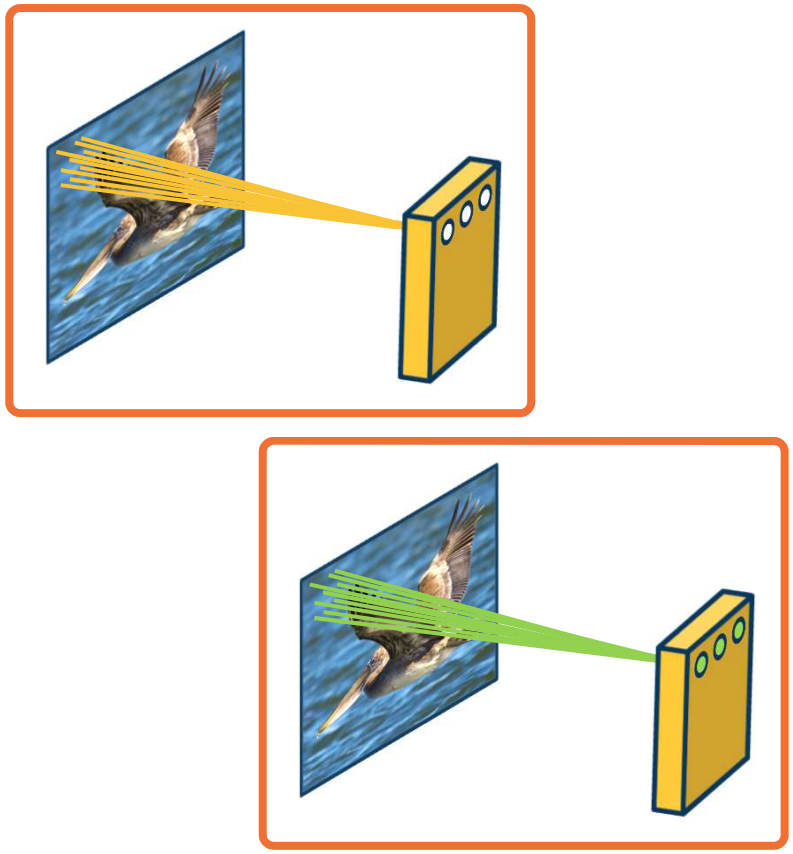
Nodes in different locations can **share** features

- No reason to think same feature (e.g. edge pattern) can't appear elsewhere
- Use same weights/parameters in computation graph **(shared weights)**

Advantages:

- Reduce parameters to $(K_1 \times K_2 + 1)$
- Explicitly maintain spatial information

Idea 2: Shared Weights

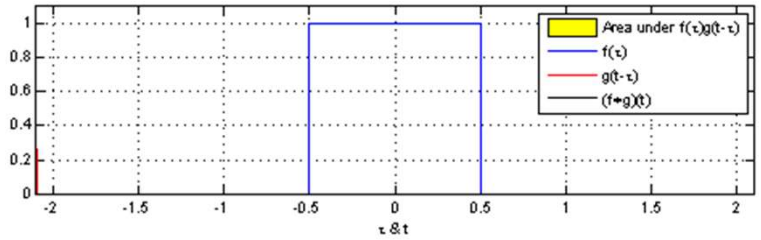


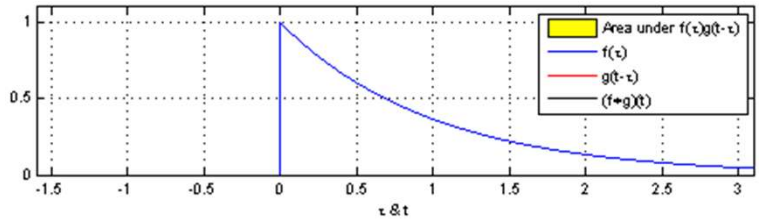
We can learn **many** such features for this one layer

- Weights are **not** shared across different feature extractors
- **Parameters:** $(K_1 \times K_2 + 1) * M$ where M is number of features we want to learn

Idea 3: Learn Many Features

This operation is **extremely common** in electrical/computer engineering!





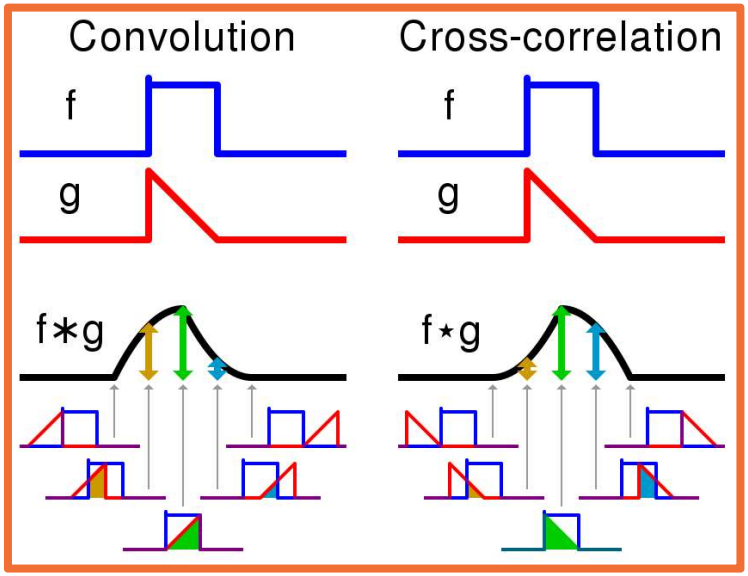
From https://en.wikipedia.org/wiki/Convolution

This operation is **extremely common** in electrical/computer engineering!

In mathematics and, in particular, functional analysis, **convolution** is a mathematical operation on two functions f and g producing a third function that is typically viewed as a modified version of one of the original functions, giving the area overlap between the two functions as a function of the amount that one of the original functions is translated.

Convolution is similar to **cross-correlation**.

It has **applications** that include probability, statistics, computer vision, image and signal processing, electrical engineering, and differential equations.



Visual comparison of **convolution** and **cross-correlation.**

From https://en.wikipedia.org/wiki/Convolution

Notation: $F \otimes (G \otimes I) = (F \otimes G) \otimes I$

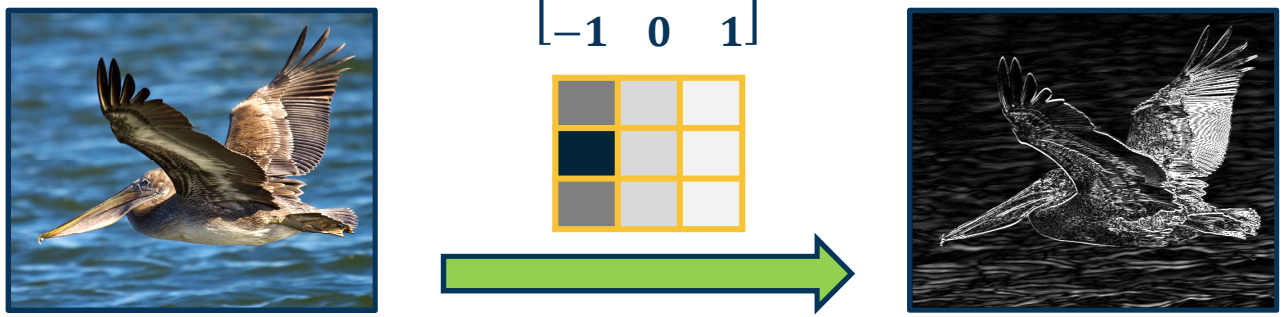
1D Convolution

$$y_k = \sum_{n=0}^{N-1} h_n \cdot x_{k-n}$$

$$y_0 = h_0 \cdot x_0$$
$$y_1 = h_1 \cdot x_0 + h_0 \cdot x_1$$
$$y_2 = h_2 \cdot x_0 + h_1 \cdot x_1 + h_0 \cdot x_2$$
$$y_3 = h_3 \cdot x_0 + h_2 \cdot x_1 + h_1 \cdot x_2 + h_0 \cdot x_3$$
$$\vdots$$

2D Convolution

$$K = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

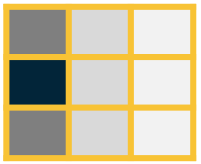


2D Convolution

Image

Kernel (or filter)

$$K = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$



Output / filter / feature map

We will make this convolution operation **a layer** in the neural network

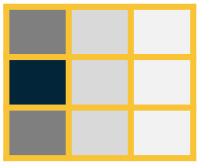
- Initialize kernel values randomly and optimize them!
- These are our parameters (plus a bias term per filter)

2D Convolution

Image

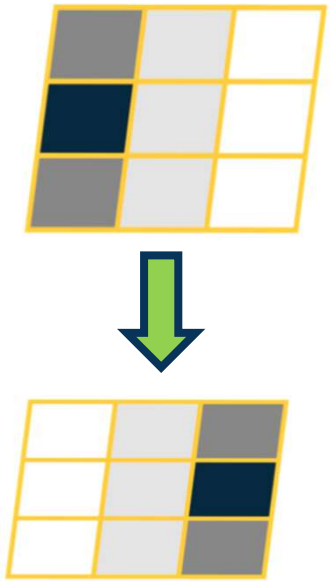
Kernel (or filter)

$$K = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

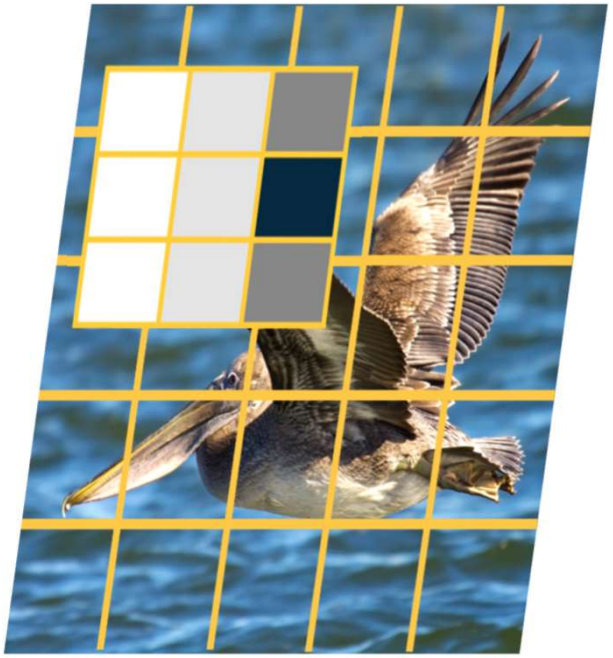


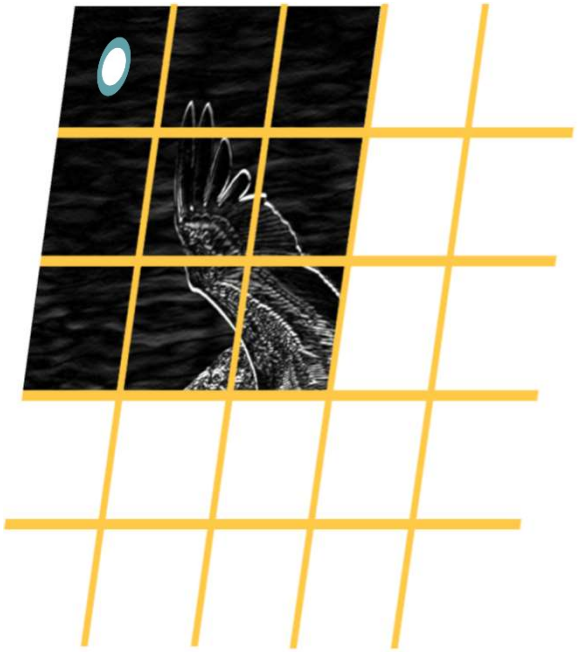
Output / filter / feature map

1. Flip kernel (rotate 180 degrees)



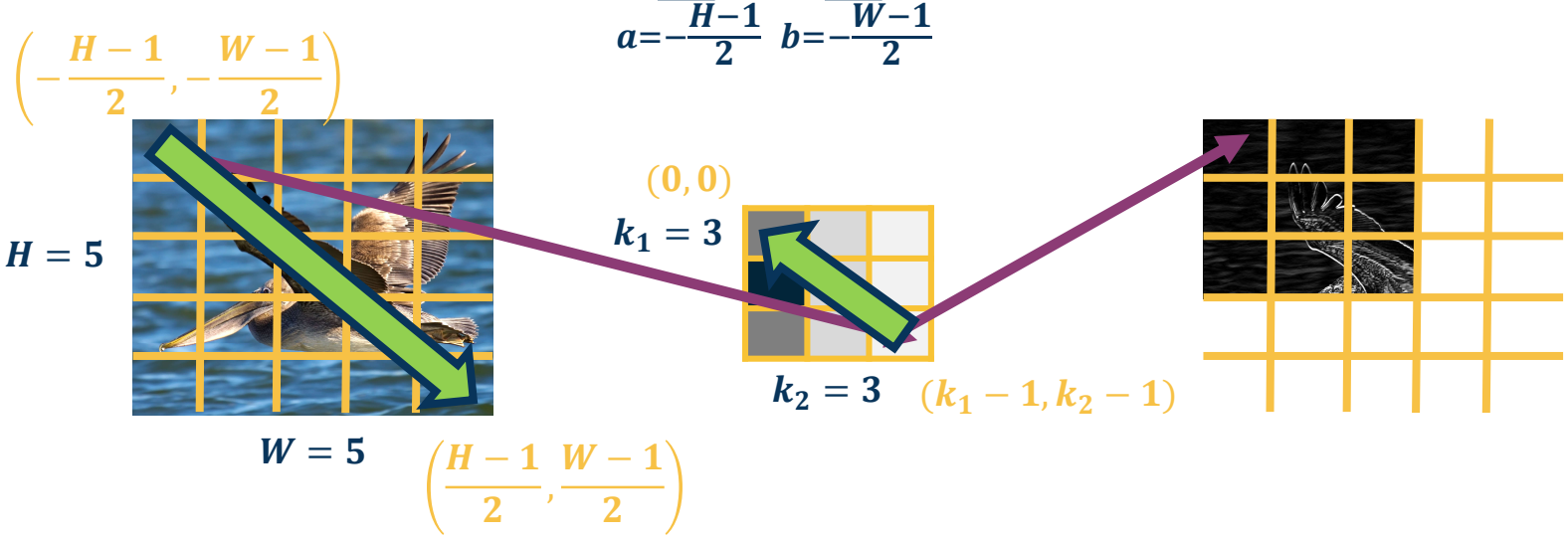
2. Stride along image





The Intuitive Explanation

$$y(r,c) = (x * k)(r,c) = \sum_{a=-\frac{H-1}{2}}^{\frac{H-1}{2},} \sum_{b=-\frac{W-1}{2}}^{\frac{W-1}{2},} x(a,b)\, k(r-a, c-b)$$



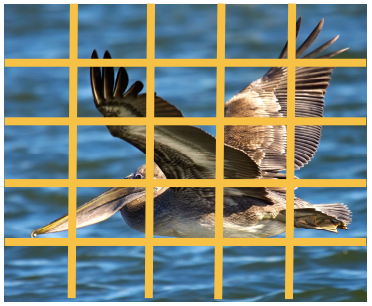
$$y(0,0) = x(-2,-2)k(2,2) + x(-2,-1)k(2,1) + x(-2,0)k(2,0) + x(-2,1)k(2,-1) + x(-2,2)k(2,-2) + \ldots$$

Mathematics of Discrete 2D Convolution

Georgia Tech

$$y(r,c) = (x * k)(r,c) = \sum_{a=-\frac{K_1-1}{2}}^{\frac{k_1-1}{2},} \sum_{b=-\frac{k_2-1}{2}}^{\frac{k_2-1}{2},} x(r-a, c-b)\, k(a,b)$$

$(0,0)$

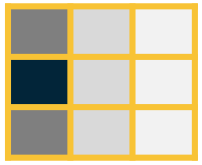


$H = 5$

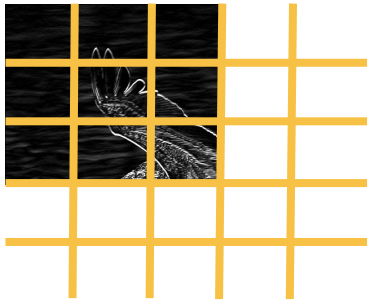
$W = 5$ $\quad (H-1, W-1)$

$(-\frac{k_1-1}{2}, -\frac{k_2-1}{2})$

$k_1 = 3$



$k_2 = 3$ $\quad (\frac{k_1-1}{2}, \frac{k_2-1}{2})$



Centering Around the Kernel

Georgia Tech

As we have seen:

- **Convolution:** Start at end of kernel and move back
- **Cross-correlation:** Start in the beginning of kernel and move forward (same as for image)

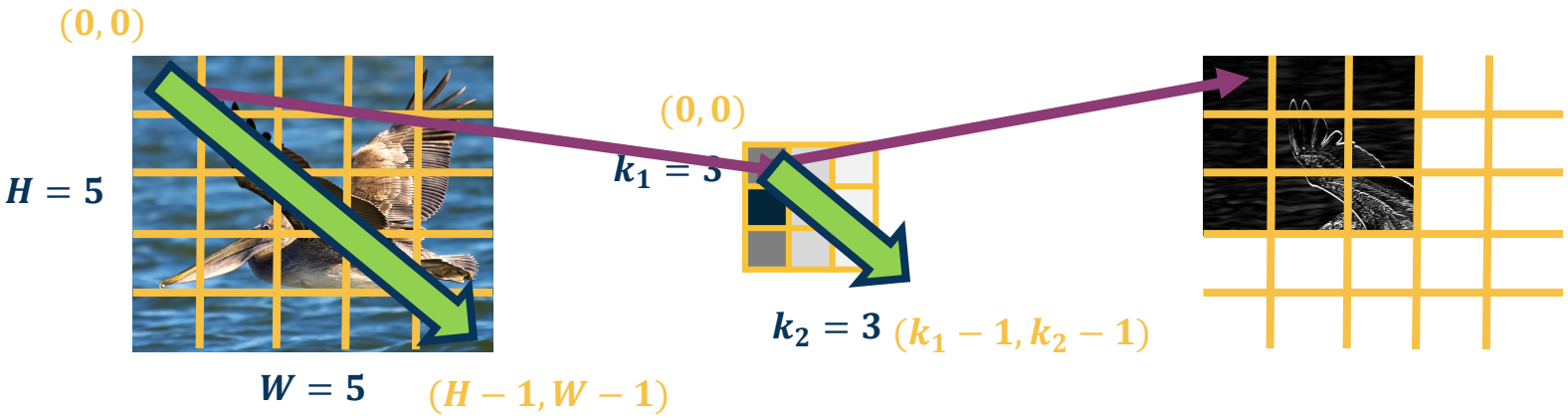
An **intuitive interpretation** of the relationship:

- Take the kernel, and rotate 180 degrees along center (sometimes referred to as “flip”)
- Perform cross-correlation
- (Just dot-product filter with image!)

$$K = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

$$K' = \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$$

$$y(r,c) = (x * k)(r,c) = \sum_{a=0}^{k_1-1} \sum_{b=0}^{k_2-1} x(r+a, c+b)\, k(a,b)$$

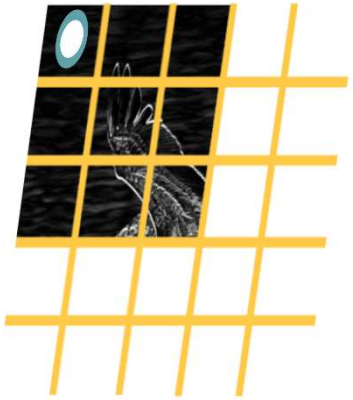


Since we will be learning these kernels, this change does not matter!

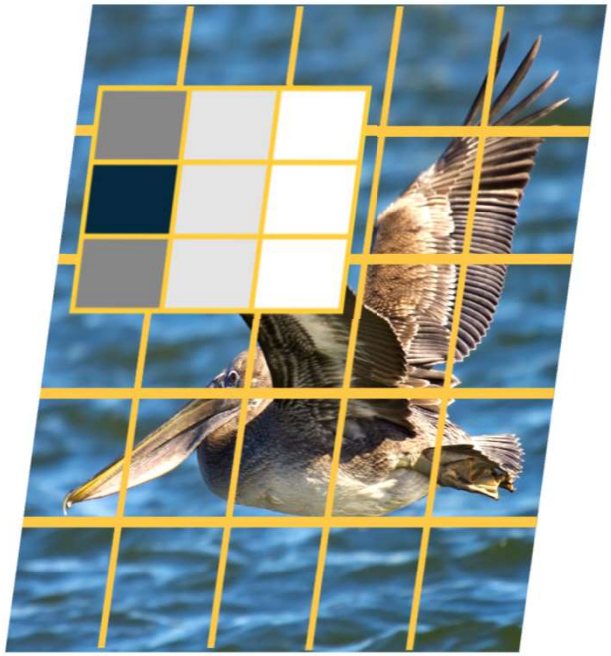
$$X(0:2,0:2) = \begin{bmatrix} 200 & 150 & 150 \\ 100 & 50 & 100 \\ 25 & 25 & 10 \end{bmatrix} \qquad K' = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

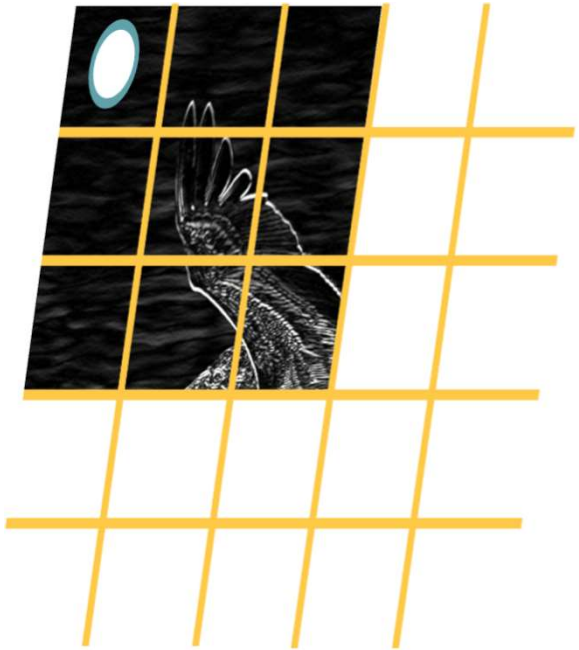
$X(0{:}2,0{:}2) \cdot K' = 65$ + bias

Dot product
(element-wise multiply and sum)

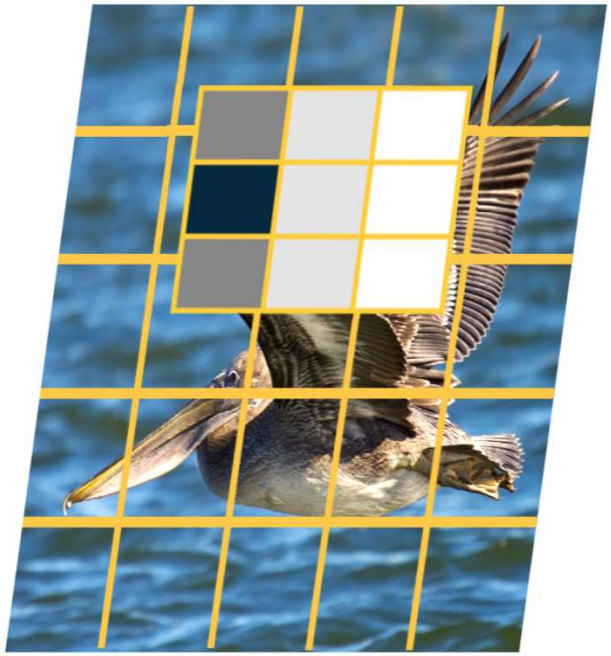


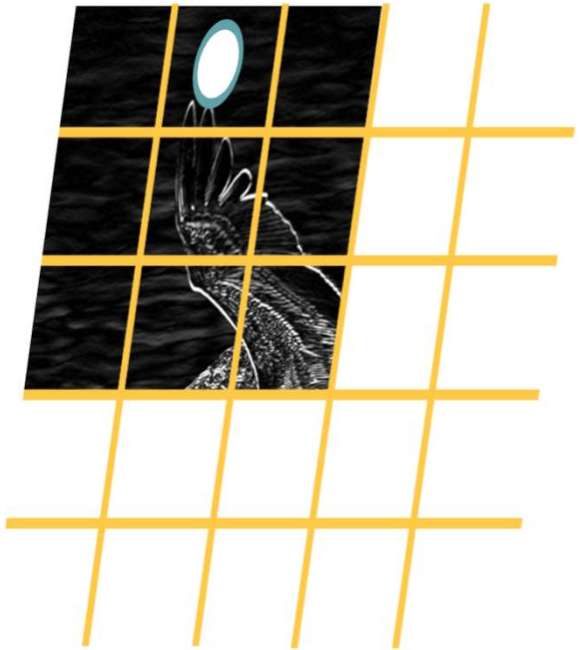
Cross-Correlation



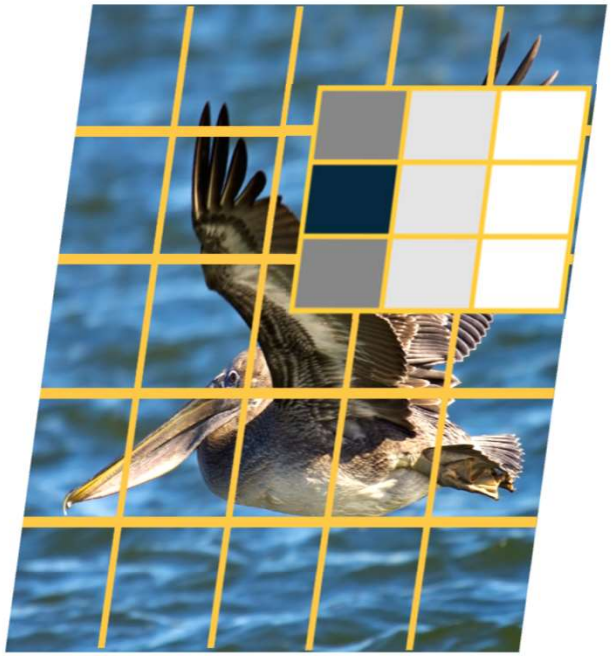


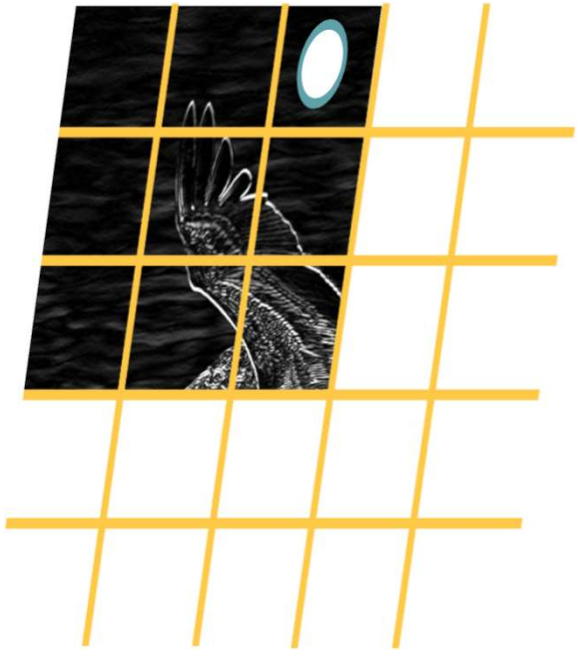
Convolution and Cross-Correlation



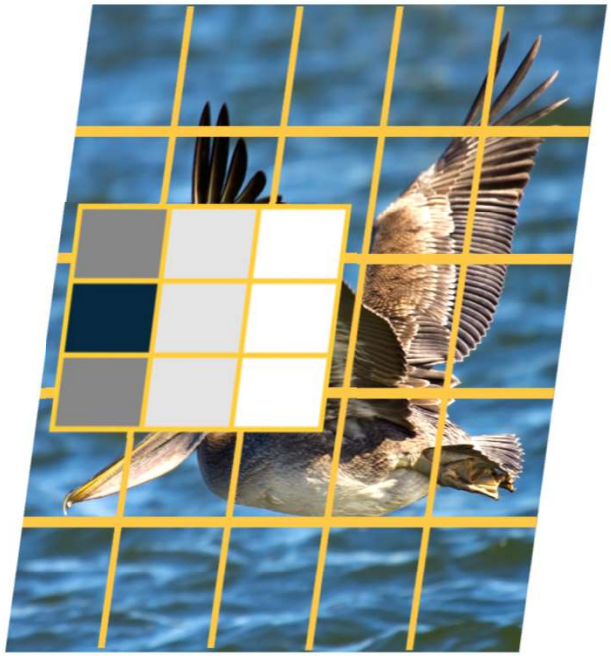


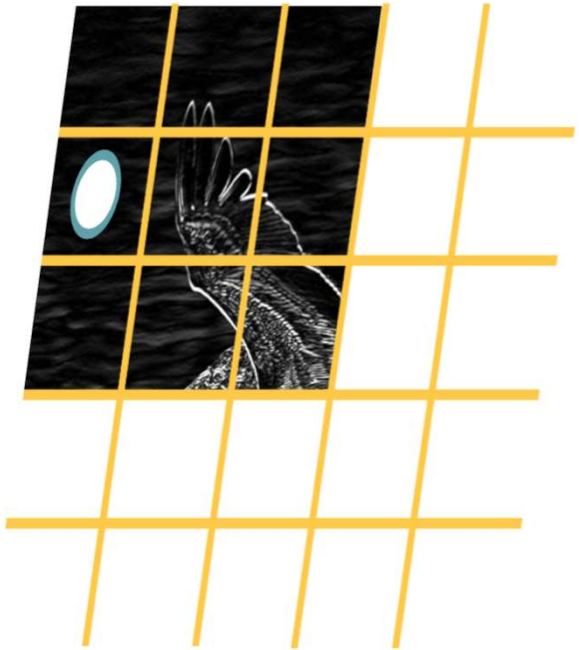
Convolution and Cross-Correlation



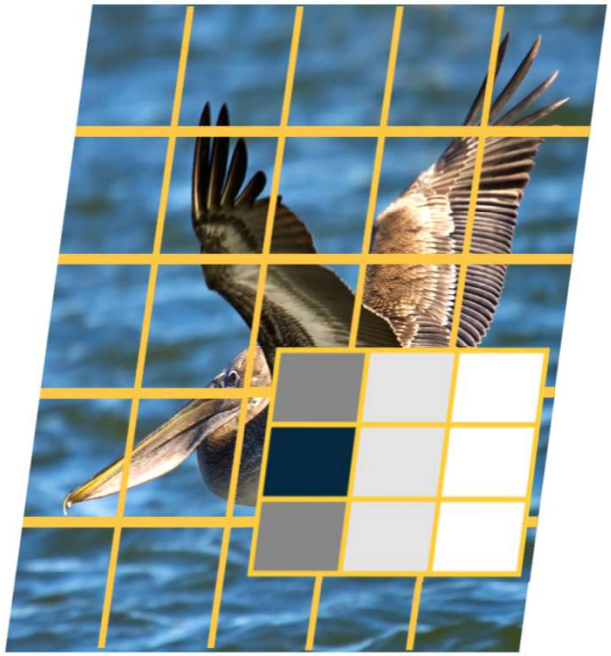


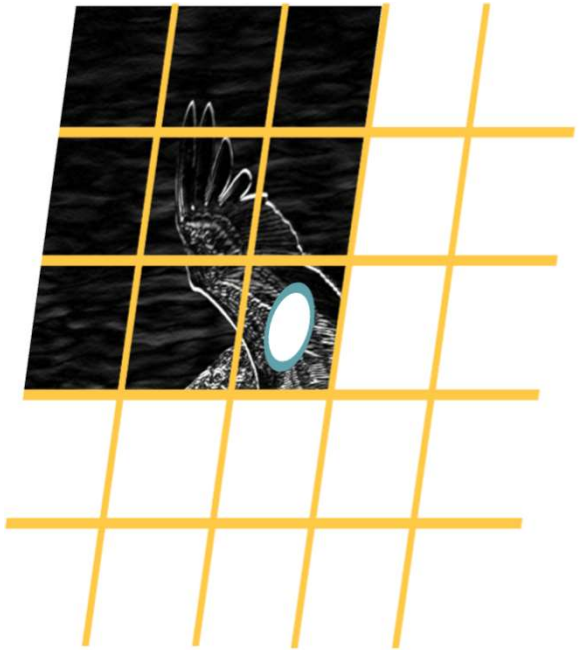
Convolution and Cross-Correlation





Convolution and Cross-Correlation





Convolution and Cross-Correlation

Why Bother with Convolutions?

Convolutions are just **simple linear operations**

Why bother with this and not just say it's a linear layer with small receptive field?

- There is a **duality** between them during backpropagation
- Convolutions have **various mathematical properties** people care about
- This is **historically** how it was inspired