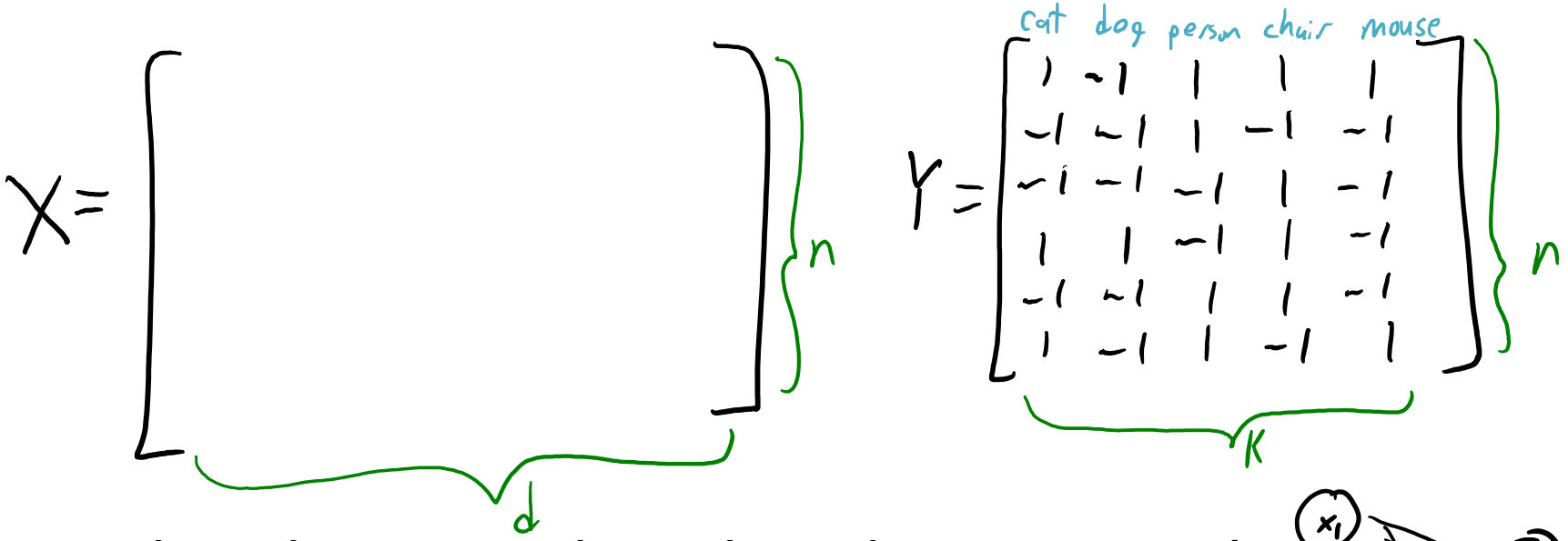
CPSC 440/540: Machine Learning

Fully-Convolutional Networks

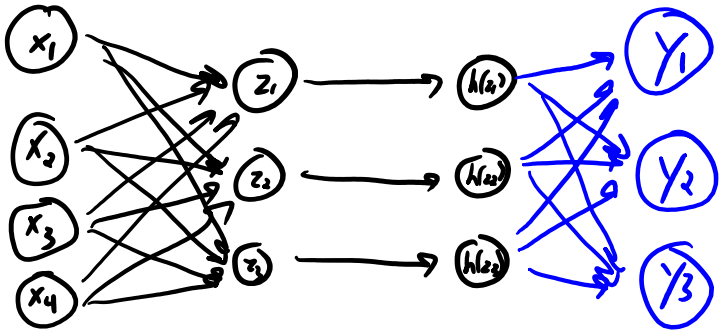
Winter 2022

Last Time: Multi-Label Classification

- Consider multi-label classification:
  - "Which of the 'k' objects are in this image?"



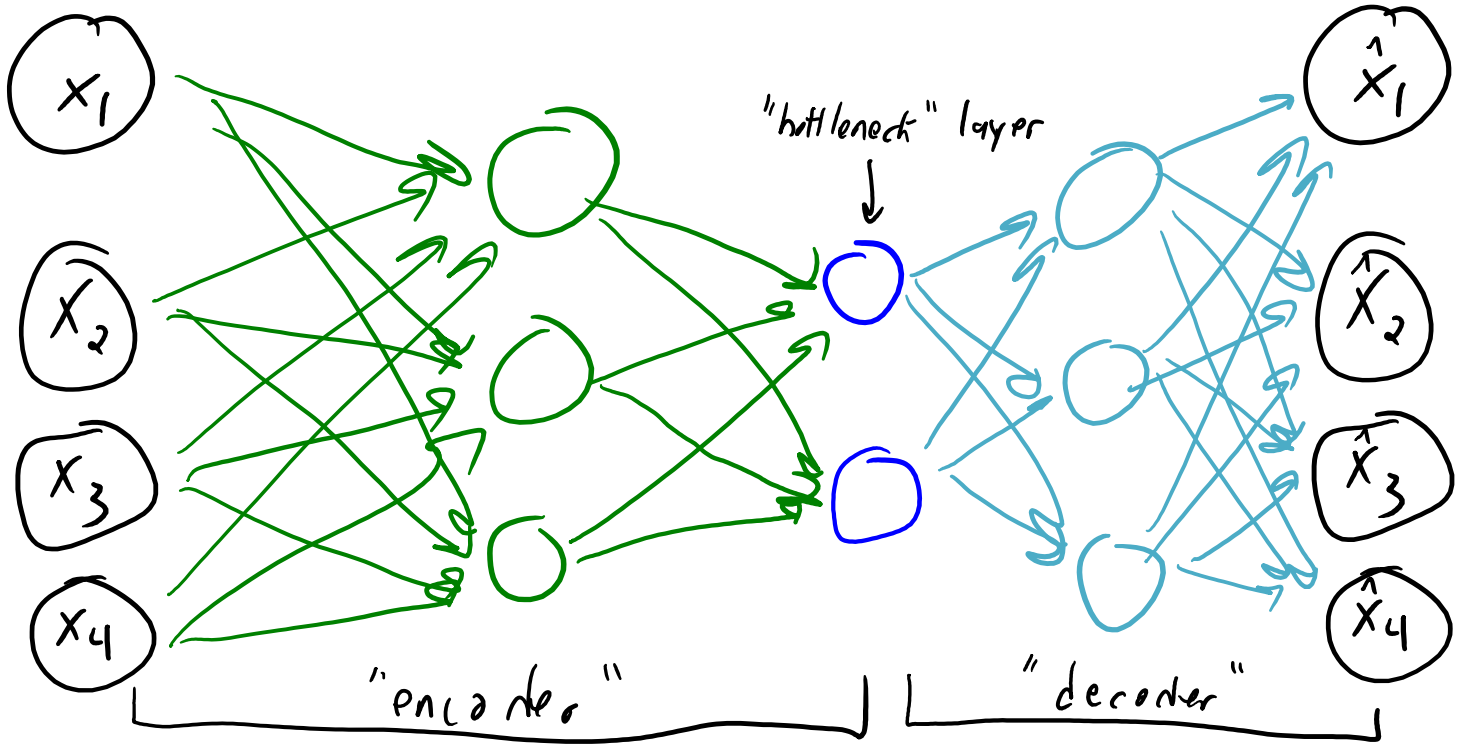
- We considered an encoding-decoding approach:
  - Fewer parameters than independent classifiers.
  - Captures dependencies through shared hidden layer.



http://image-net.org/challenges/LSVRC/2013/

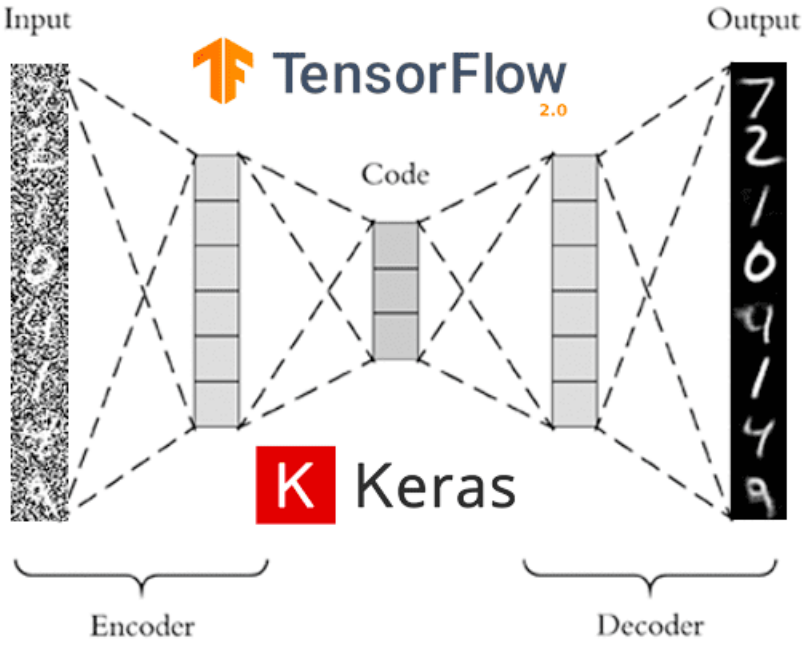
Some clarification about “encoders”/”decoders”

- (Vanilla) autoencoders have clear “bottleneck” / “encoder” / “decoder” structure (…right?)



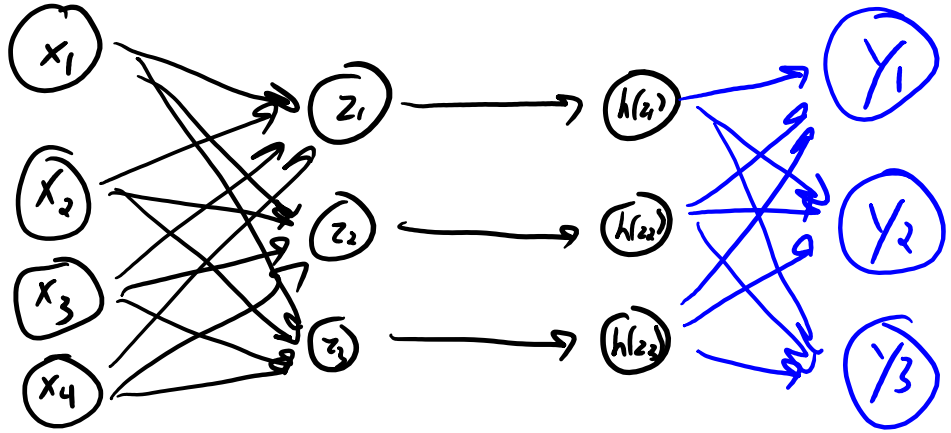
Some clarification about “encoders”/”decoders”

- (Vanilla) autoencoders have clear “bottleneck” / “encoder” / “decoder” structure (…right?)
- Denoising autoencoders: not necessarily…



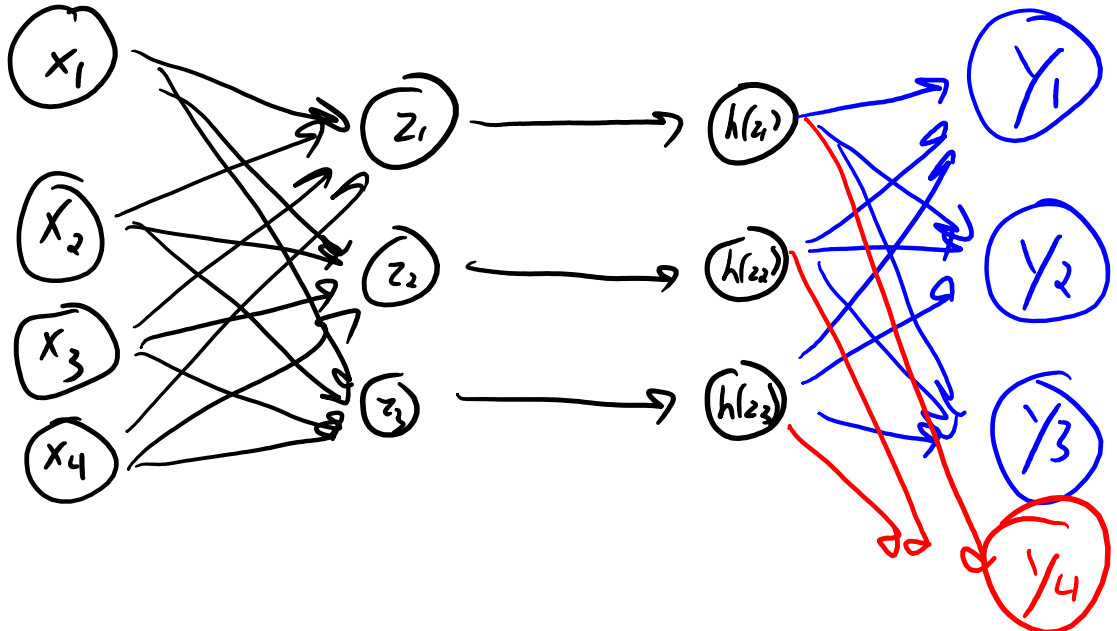
Some clarification about “encoders”/”decoders”

- (Vanilla) autoencoders have clear “bottleneck” / “encoder” / “decoder” structure (…right?)
- Denoising autoencoders: not necessarily…
- “Encoder” / “decoder” structure for multilabel classifiers:
  - Just terminology we decide to use!
  - Definitely not an **auto**encoder!



Some clarification about unsupervised pre-training

- One version of what I meant last time: semi-supervised learning
  - 1. Train an autoencoder on all of Instagram (e.g.)
  - 2. Use its features for softmax regression on your (small) labeled dataset
- Could also "fine-tune" the whole network;
  this would be one version of unsupervised pre-training



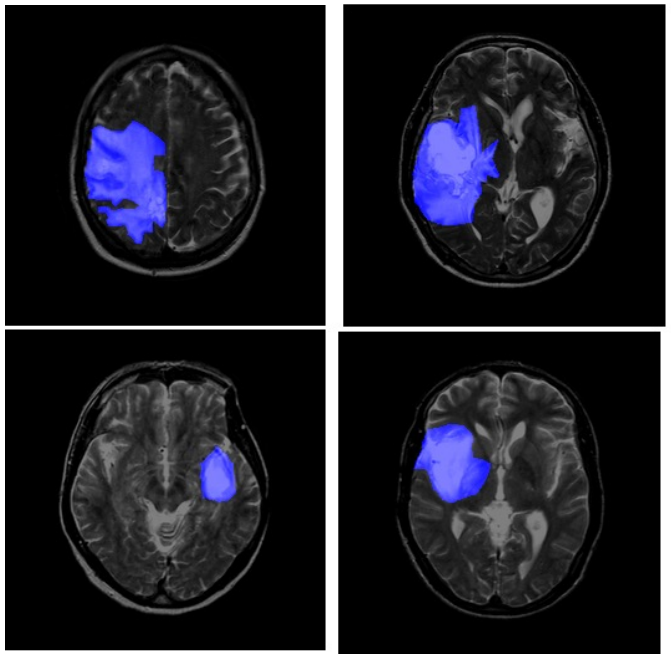
bonus!

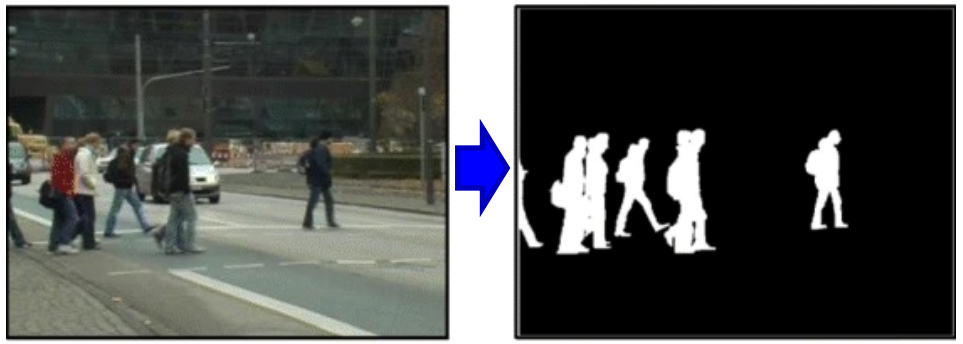
Some clarification about unsupervised pre-training

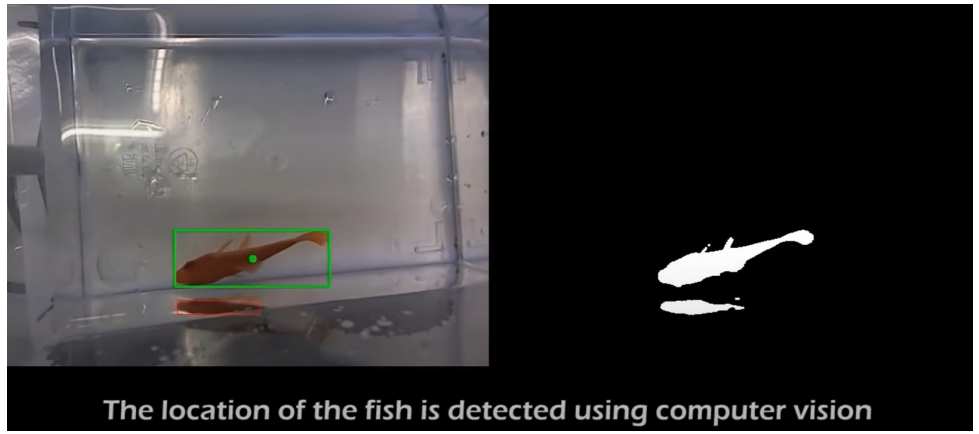
- What “unsupervised pre-training” used to mean
- Old scheme for deep networks: **stacked** denoising autoencoders
  - Train a two-layer denoising autoencoder
  - Freeze the encoder layer as first layer of your deep net
  - Train a denoising autoencoder on activations from that layer
  - Freeze its encoder as the second layer of your deep net
  - Repeat
  - Fine-tune with SGD at the end
- People don’t do this anymore: we can do end-to-end SGD now

Motivation: Pixel Classification

- Suppose we want to assign a binary label to each pixel in an image:
  - Tumour vs. non-tumour, pedestrian vs. non-pedestrian, and so on.





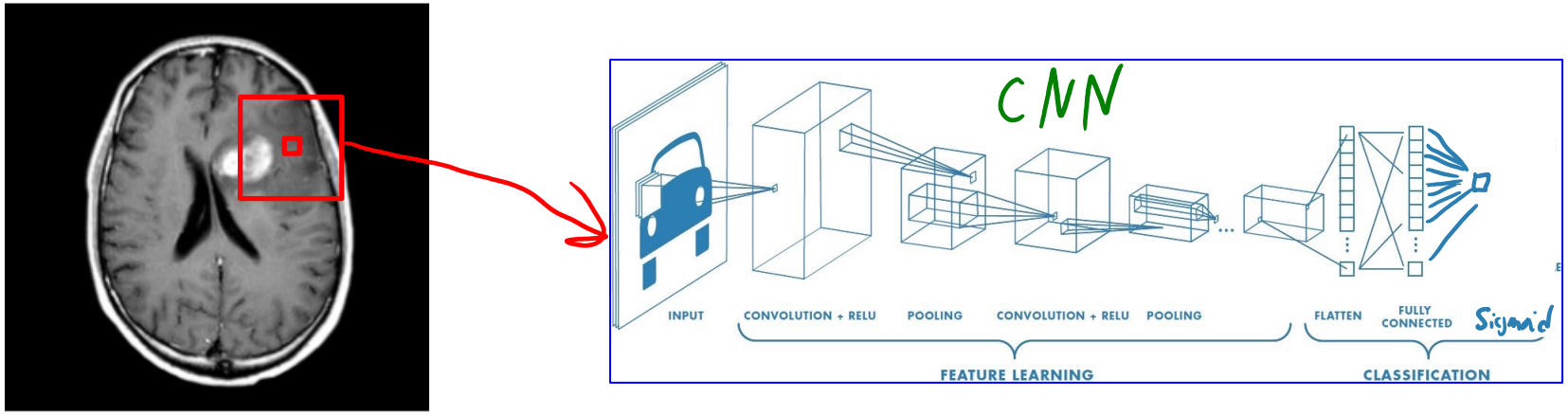


- How can we use CNNs for this problem?

https://ipsjcva.springeropen.com/articles/10.1186/s41074-019-0062-2
https://www.youtube.com/watch?v=YbNmL6hSNKw

Naïve Approach: Sliding Window Classifier

- Train a CNN that predicts pixel label given its neighbourhood.

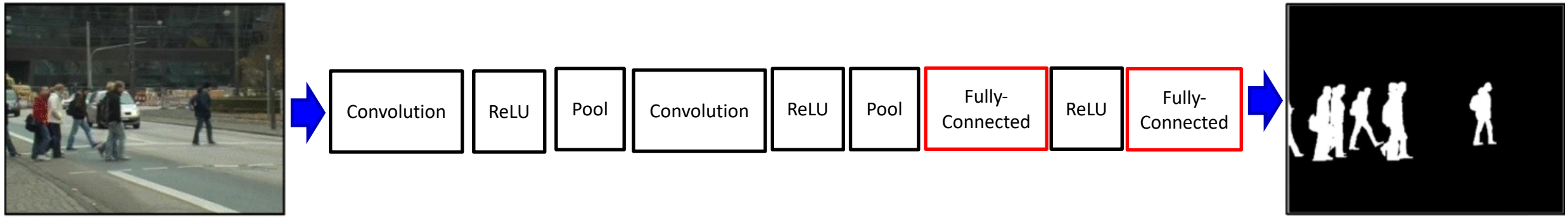


- To label all pixels, apply the CNN to each pixel.
  - Advantages:
    - Turns pixel labeling into image classification.
    - Can be applied to images of different sizes.
  - Disadvantage: this is slow.
    - (Cost of applying CNN) * (number of pixels in the image).

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Encoding-Decoding for Pixel Classification

- Similar to multi-label, could use CNN to generate an image encoding.
  - With output layer making a prediction at each pixel.

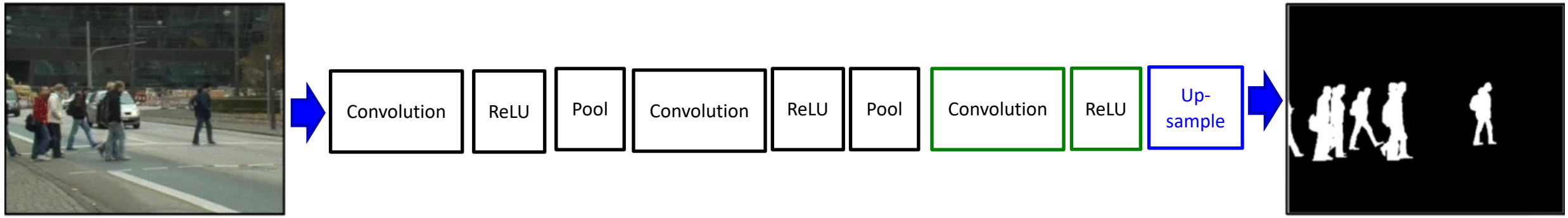


- Much-faster classification.
  - Small number of "global" convolutions, instead of repeated "local" convolutions.
- But, the encoding has mixed up all the space information.
  - Due to fully-connected layers.
  - Fully-connected layer needs to learn "how to put the image together".
    - And images must be the same size.

https://ipsjcva.springeropen.com/articles/10.1186/s41074-019-0062-2

Fully-Convolutional Networks

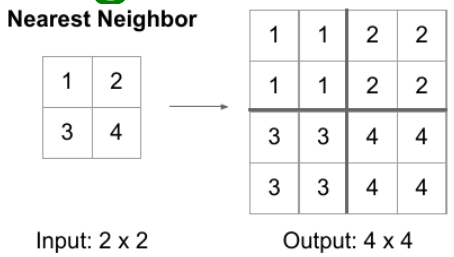
- Fully-convolutional networks (FCNs):
  - CNNs with no fully-connected layers (only convolutional and pooling).



- Maintains fast classification of the encoding-decoding approach.
- Same parameters used across space at all layers.
  - This allows using the network on inputs of different sizes.
  - Needs upsampling layer(s) to get back to image size.
- FCNs quickly achieved state of the art results on many tasks.

Traditional Upsampling Methods

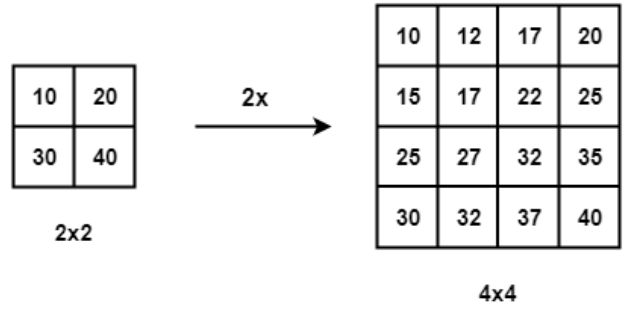
- In upsampling, we want to go from a small image to a bigger image.
  - This requires interpolation: guessing “what is in between the pixels”.
- Classic upsampling operator is nearest neighbours interpolation:
  - But this creates blocky/pixelated images.



https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba

Traditional Upsampling Methods

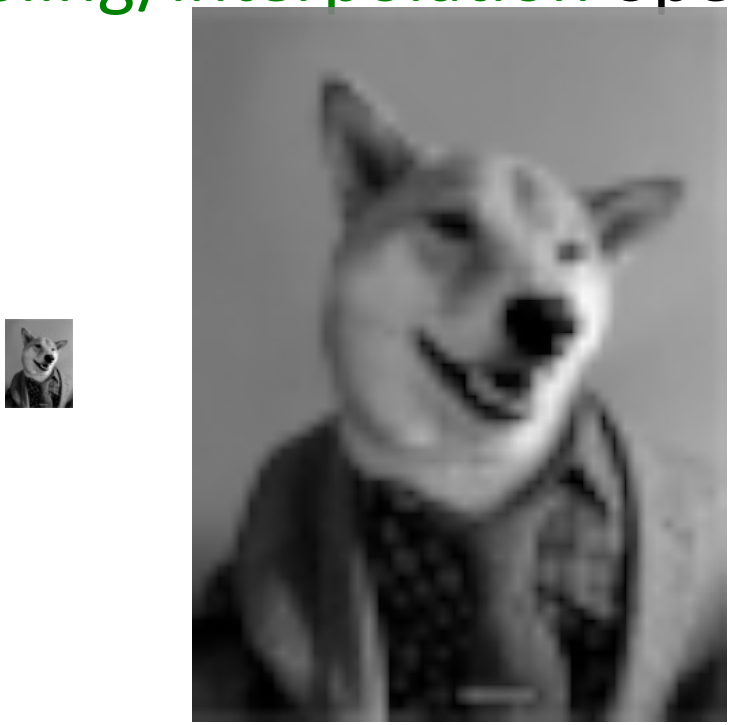
- Another classic method is bilinear interpolation:
  - Weighted combination of corners:
  - More smooth methods include “bicubic” and “splines”.



2x2

4x4

- In FCNs, we learn the upsampling/interpolation operator.



https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba

Upsampling with Transposed Convolution

- FCN Upsampling layer is implemented with a transposed convolution.
  - Sometimes called "deconvolution" in ML or "fractionally-strided convolution".
    - But not related to deconvolution in signal processing.

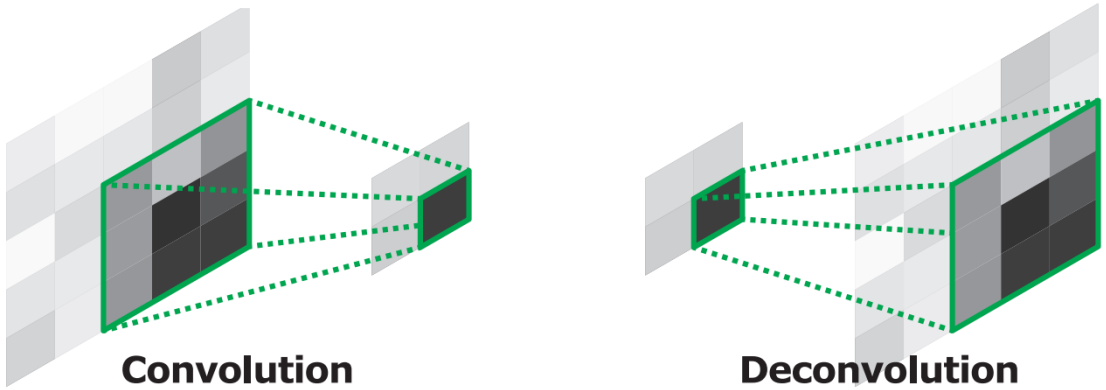
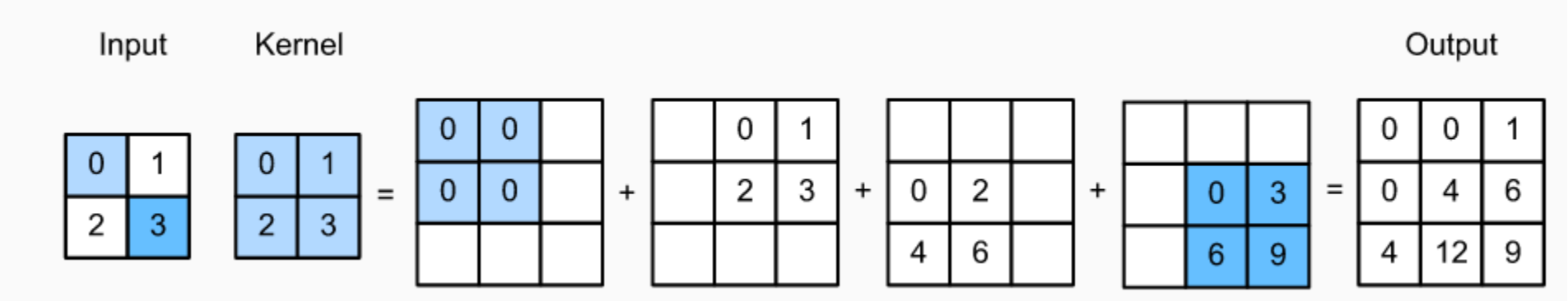


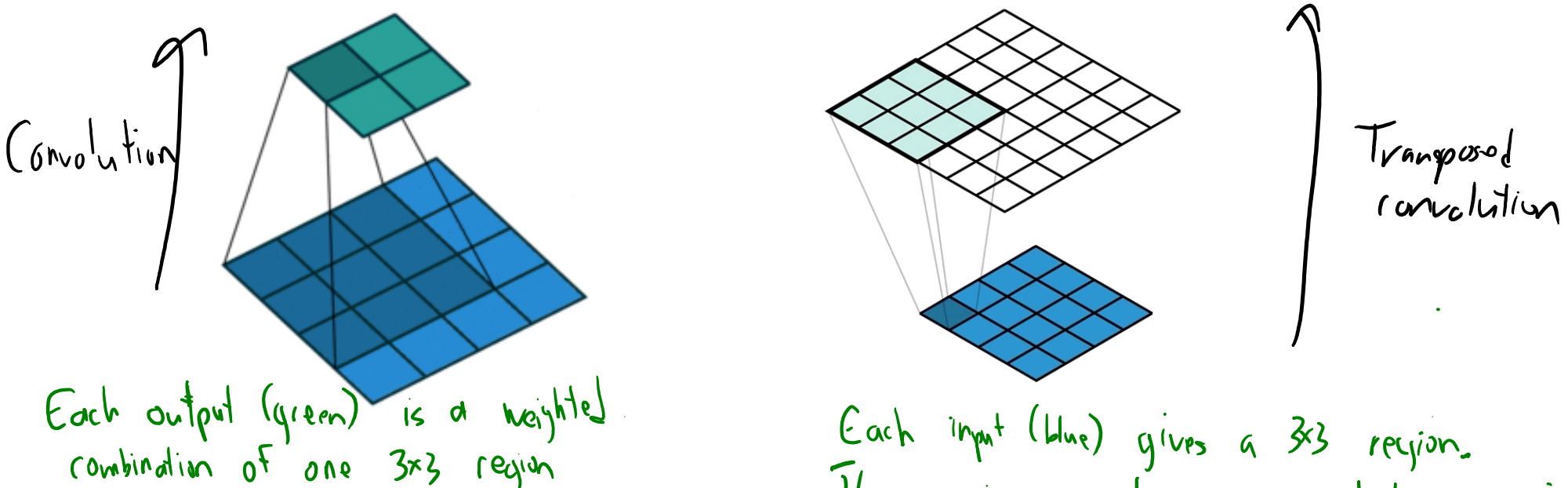
Figure 3. Illustration of deconvolution and unpooling operations.

- Convolution generates 1 pixel by taking weighted combination of several pixels.
  - And we learn the weights.
- Transposed convolution generates several pixels by weighting 1 pixel.
  - And we learn the weights.
  - This generates overlapping regions, which get added together to make final image.

https://www.cv-foundation.org/openaccess/content_iccv_2015/papers/Noh_Learning_Deconvolution_Network_ICCV_2015_paper.pdf

Upsampling with Transposed Convolution



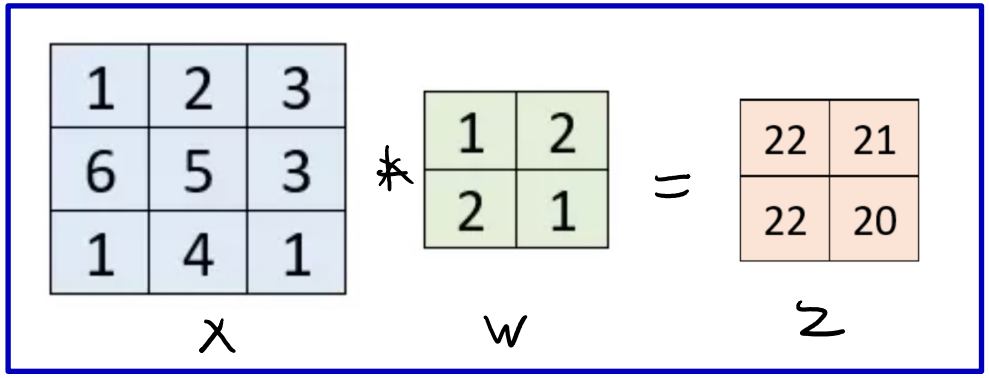


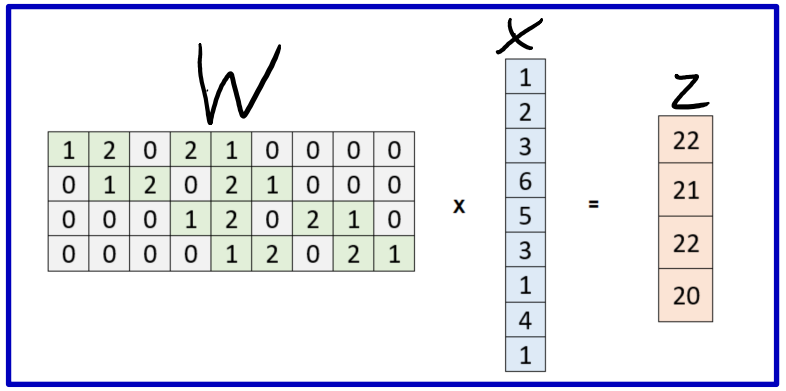
- Animations here and here.

https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba

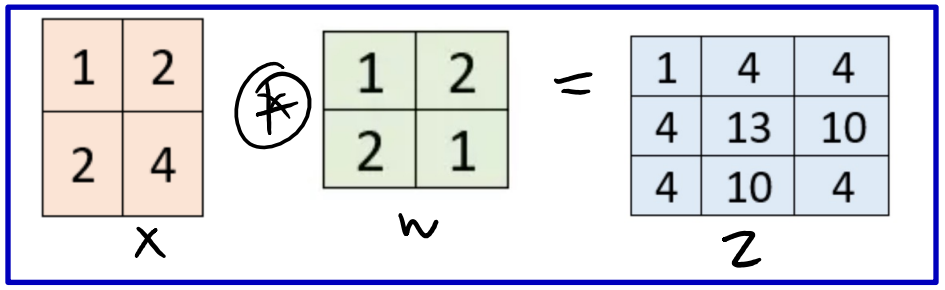
Why is it called “transposed” convolution?

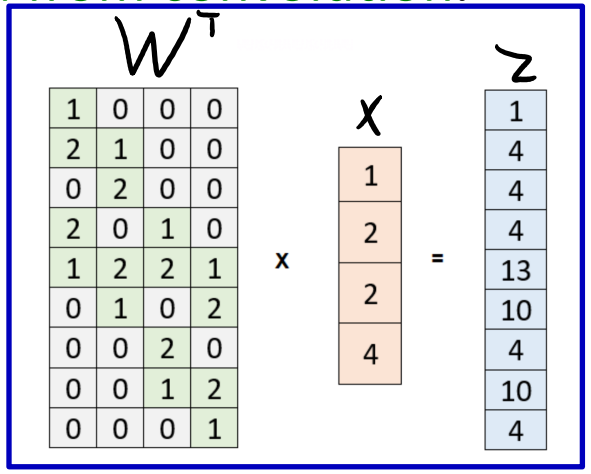
- We can write the convolution operator as a matrix multiplication, z = W’ x.





- In transposed convolution, non-zero pattern of ‘W’ is transposed from convolution.
  - You can implement transposed convolution as a convolution.





- In this example the filter is the same, but does not need to be:
  - Transposed convolution is not the “reverse” of convolution (it only “reverses” the size).

https://www.machinecurve.com/index.php/2019/09/29/understanding-transposed-convolutions/

Increasing Resolution: FCN Skip Connections

bonus!

- Convolutions and pooling lose a lot of information.
- Original FCN paper considered adding skip connections to help upsampling:

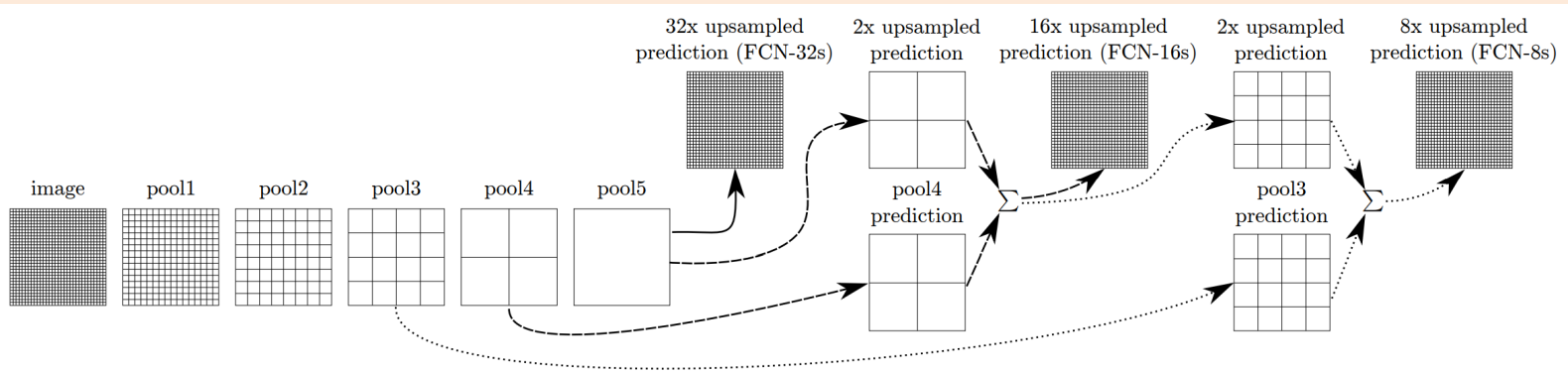
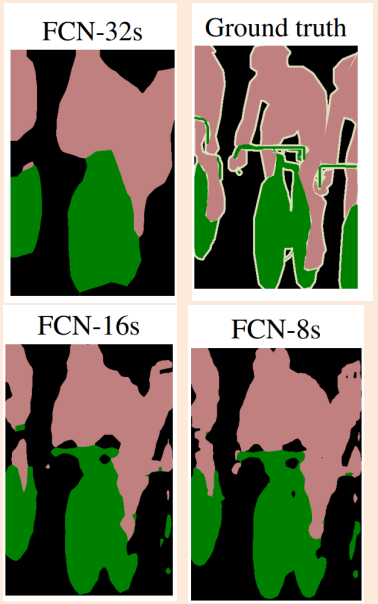


Figure 3. Our DAG nets learn to combine coarse, high layer information with fine, low layer information. Layers are shown as grids that reveal relative spatial coarseness. Only pooling and prediction layers are shown; intermediate convolution layers (including our converted fully connected layers) are omitted. Solid line (FCN-32s): Our single-stream net, described in Section 4.1, upsamples stride 32 predictions back to pixels in a single step. Dashed line (FCN-16s): Combining predictions from both the final layer and the `pool4` layer, at stride 16, lets our net predict finer details, while retaining high-level semantic information. Dotted line (FCN-8s): Additional predictions from `pool3`, at stride 8, provide further precision.



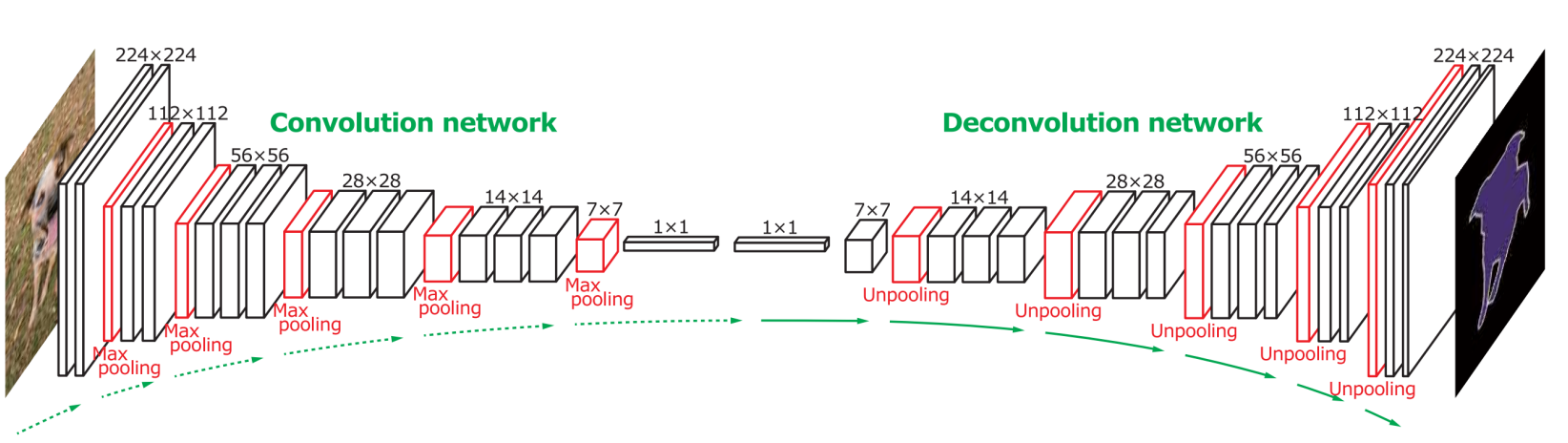
- Allows using high-resolution information from earlier layers.
- They first trained the low-resolution FCN-32, then FCN-16, then FCN-8.
  - "First learn to encode at a low resolution", then slowly increase resolution.
  - Parameters of transposed convolutions initialized to simulate "bilinear interpolation".

https://arxiv.org/pdf/1411.4038.pdf

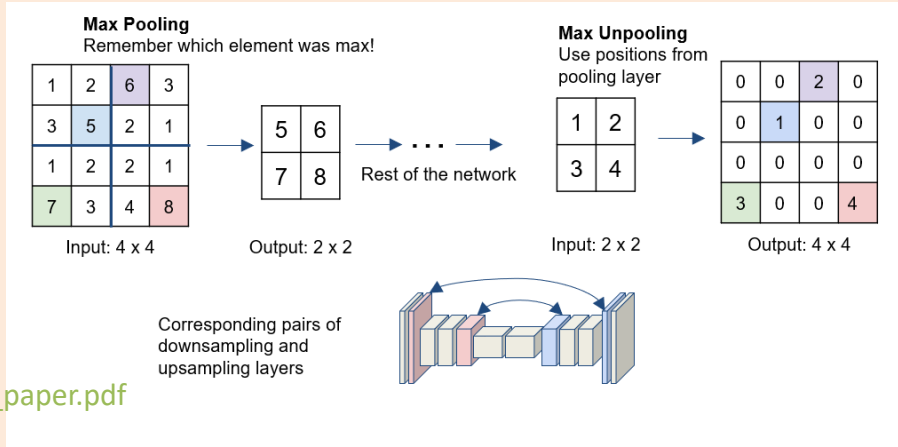
bonus!

Increasing Resolution: Deconvolution Networks

- Alternate resolution-increasing method is deconvolution networks:



- Includes transposed convolution layers and unpooling layers.
  - Store the max pooling argmax values.
  - Restores "where" activation happened.
    - Still loses the "non-argmax" information.

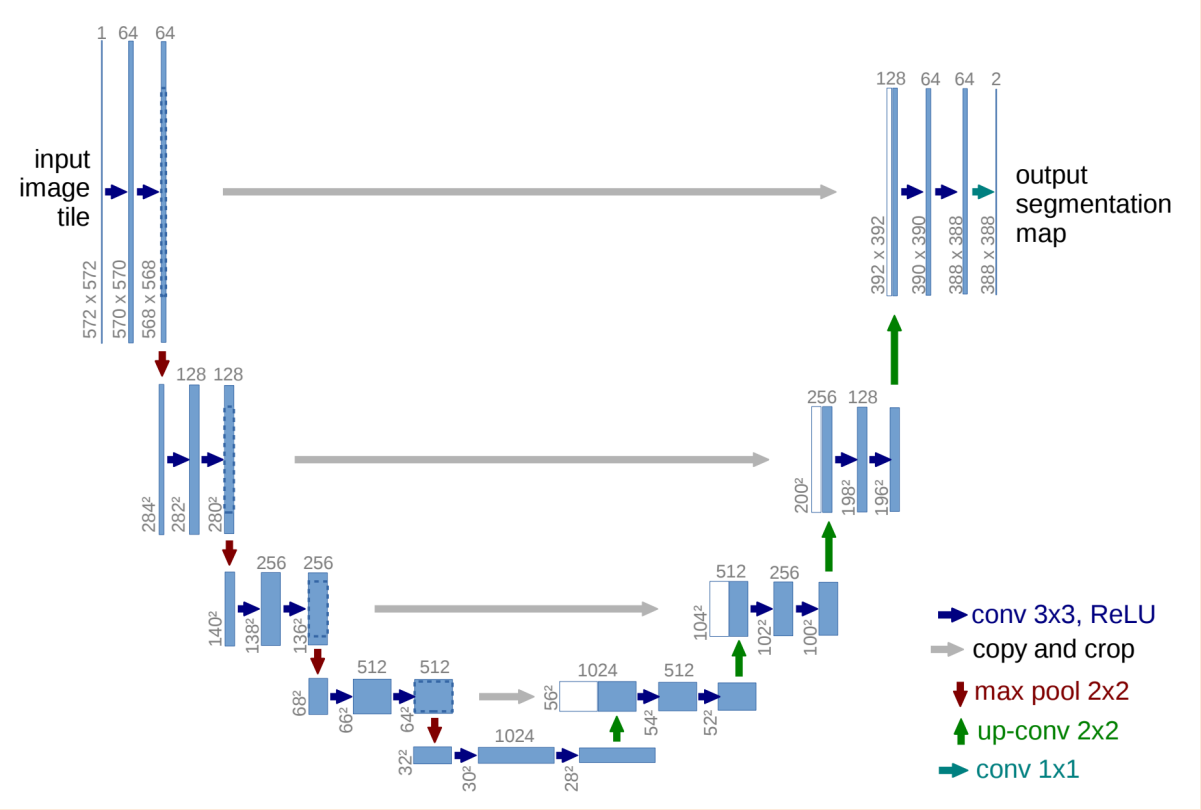


https://www.cv-foundation.org/openaccess/content_iccv_2015/papers/Noh_Learning_Deconvolution_Network_ICCV_2015_paper.pdf
https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba

bonus!

Increasing Resolution: U-Nets

- Another popular variant is u-nets:



- Decoding has connections to same-resolution encoding step.

https://arxiv.org/pdf/1505.04597.pdf

Source of Labels

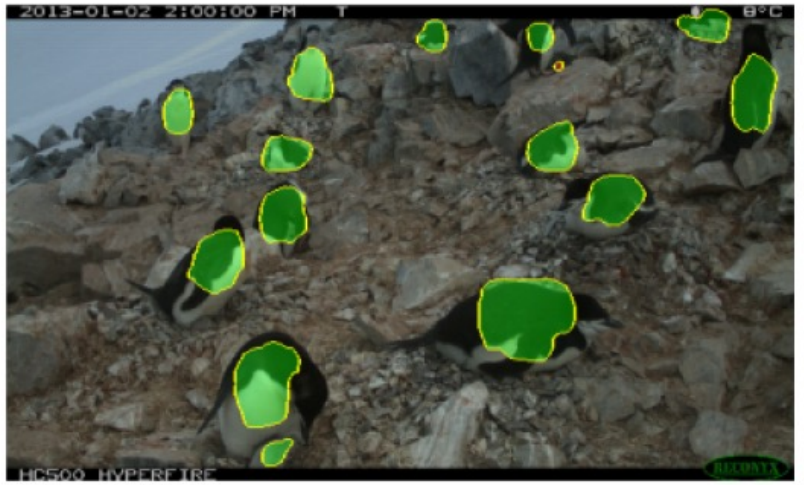
- Labeling every pixel takes a long time.
- Where we get the labels from?
  - Domain expert (medical images).
  - Grad students or paid labelers (ImageNet).
  - Simulated environments.
    - High number of lower-quality examples.
    - Often a net gain with fine-tuning on real images.
    - Can get data at night, in fog, or dangerous situations.

Video game

Google street view

Source of Labels

- Recent works recognize you do not need to label every pixel.
  - You can evaluate loss/gradient on a subset of labeled pixels.
  - Could have labeler click on a few pixels inside objects, and a few outside.
    - Many variations are possible, that let you label a lot of images in a short time.
- Penguin counting based “single pixel” labels in training data:
  - And some tricks to separate objects and remove false positives:



End of Part 1 ("Binary Variables"): Key Concepts

- We discussed binary density estimation.
  - Model the proportion of times a binary event happens.
- We discussed the Bernoulli parameterization.
- We discussed various inference tasks, given the parameter:
  - Compute probabilities, find decoding, generate samples.
- We discuss different learning strategies, given data:
  - Maximum likelihood estimation (MLE), maximum a posteriori (MAP).
  - Beta distribution as a prior gives a beta distribution as posterior ("$\propto$").
- We discussed modeling binary variables conditioned on features:
  - Tabular parameterization is flexible but has too many parameters.
  - Logistic regression is limited but has a linear number of parameters.

End of Part 1 ("Binary Variables"): Key Concepts

- We discussed multivariate binary density estimation.
  - Refined inference tasks when we have more than one random variable:
    - Joint probability, marginal probability, and conditional probability.
  - Product of Bernoullis assumes variables are independent.
    - Fast inference/learning but a strong assumption.
- We discussed generative classifiers:
  - Build a model of the joint probability of features and labels.
    - Compared to usual discriminative classifiers that model labels given features.
  - Naïve Bayes assumes features are independent given label.
- We discussed neural networks:
  - Model that learns the features and classifier simultaneously.
  - Alternate between linear and non-linear transformations (universal approximator).
  - Training is a non-convex problem, but SGD often works better than expected:
    - For large-enough networks we often find global, and SGD seems to have implicit regularization.

End of Part 1 (“Binary Variables”): Key Concepts

- We discussed deep learning with multiple hidden layers.
  - Biological motivations and efficient representation of some functions.
  - Vanishing gradient problem and modern solutions:
    - ReLU, skip connections, ResNets.
- We discussed automatic differentiation to generate gradient code.
  - Code that generates gradient code for you (using chain rule).
- We discussed convolutional neural networks (CNNs):
  - Include convolution layers that measure image features.
  - Include max pooling layers that highlight top features across space.
  - Reduces number of parameters and gives some spatial invariance.

End of Part 1 (“Binary Variables”): Key Concepts

- We discussed autoencoders:
  - Networks where the output is the input.
  - Encodes input into a bottleneck layer, then decodes back to input.
  - Non-linear dimensionality reduction.
  - Denoising autoencoders learn to enhance images.
- We discussed multi-label classification:
  - Where each training examples can have 0-k correct labels.
  - We discussed an encoding approach where the classes shares hidden layers.
    - Reduces parameters and captures dependencies between labels.
  - We discussed pre-training to learn new tasks with fewer labeled examples.
- We discussed pixel labeling:
  - Fully-convolutional networks maintain spatial information at all layers.
    - Requires upsampling to original image size.
    - Can label images of different sizes.

Next Topic: Categorical Variables

Motivating problem: Political Polling

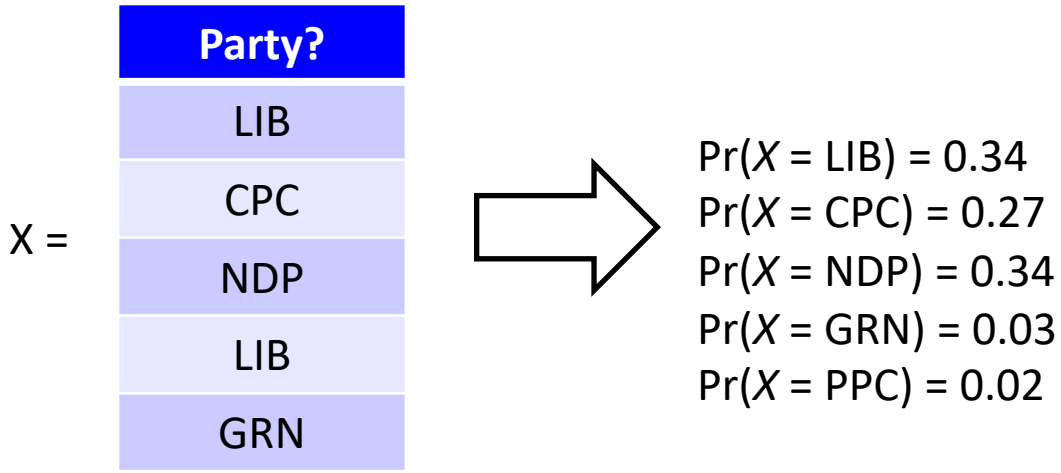
- Want to know support for political parties among a voter group.
  - What percentage will vote the Liberal party? Conservative party? NDP?
    - What to know support for each party, since may want to attract voters?

- Where Mark lives, the last election results were:
  - 34.4% LIB
  - 33.5% NDP
  - 26.8% CPC
  - 2.9% GRN
  - 2.4% PPC

- We want to predict these quantities based on a sample ("poll").

General Problem: Categorical Density Estimation

- This is a special case of density estimation with a categorical variable:
  - Input: *n* IID samples of categorical values x^1, x^2, x^3,…, x^n from a population.
  - Output: model of probability that *X*=1, *X*=2, *X*=3, …, *X*=k.
- Categorical density estimation as a picture:



- We are going to revisit many of our previous concepts in this case.
  - Again building up to more-complicated cases.
    - And introducing some concepts that we skipped in Part 1.

Other Applications of Categorical Density Estimation

- Other applications where categorical density estimation is useful:
  - What portion of my clients use cash, credit, or debit?
  - Prevalence of different blood types.
  - Probability of having different types of cancers.
  - Probability of seeing different words (natural language processing).

- For categorical variables, we do not assume there is an ordering.
  - Category 4 is not “closer” to category 3 than it is to category 1.

bonus!

Ordinal Variables

- Categorical variables with an ordering are called ordinal:
  - Dice (1, 2, 3, 4, 5, 6).
    - Though I may use dice to illustrate categorical ideas.
  - Survey results (“strongly disagree”, “disagree”, “neutral”,…).
  - Ratings (1 star, 2 star,…).
  - Tumour grading (Grade I, Grade II, Grade III, Grad IV).

- We will not cover ordinal variables (for now), but several methods exist.
  - Such as “ordinal logistic regression”.
    - A loss function that reflects that “2 stars” is closer to “3 stars” than “4 stars”.
      - But the distances between adjacent “stars” may not be the same.
    - That loss function can be used in place of “softmax” in neural nets with ordinal labels.

Parameterization of Categorical Probabilities

- We typically parameterize using the categorical distribution:
  - Sometimes called "multinoulli".
  - Has parameters $\theta_1, \theta_2, \ldots, \theta_k$ when we have *k* categories.
  - Defines probabilities using:

$$p(x=1 \mid \theta_1, \theta_2, \ldots, \theta_k) = \theta_1 \quad p(x=2 \mid \theta_1, \theta_2, \ldots, \theta_k) = \theta_2 \quad \cdots \quad p(x=k \mid \theta_1, \theta_2, \ldots, \theta_k) = \theta_k$$

  - Because probabilities sum to 1, we require: $\sum_{c=1}^{K} \theta_c = 1$

- One way to write this for a generic *x*:

$$p(x \mid \Theta) = \theta_1^{\mathbb{1}(x=1)} \theta_2^{\mathbb{1}(x=2)} \cdots \theta_K^{\mathbb{1}(x=k)}$$

$$\Theta = \{\theta_1, \theta_2, \ldots, \theta_k\}$$

Inference Task: Union

- Inference task : given θ, compute probabilities of unions:
  - For example, Pr(X = LIB ∪ X = NDP | Θ).
    - What fraction of votes would these parties their supporters voted together?

- We assume the categories are mutually exclusive.
  - "You can only pick one."
  - This allows us to compute unions with addition:
    - Pr(X = 2 ∪ X = 3 ∪ X = 4 | Θ) = $\theta_2 + \theta_3 + \theta_4$.

- A variation on this task is computing Pr($X \leq c$) for some value c.
  - Easy to do: Pr($X \leq 4$) = $\theta_1 + \theta_2 + \theta_3 + \theta_4$.
  - We often want to do this even though the categories are unordered.
  - We call Pr($X \leq$ c) the cumulative distribution function (CDF).

Inference Task: Mode / Decoding

- Inference task: given θ, find *x* that maximizes p(x | Θ) (decoding).

- Probably the most relevant inference for the elections example:
  - The mode is "who wins the election".

- Computing the decoding using "argmax" notation:

$$x_{decode} \in \underset{c}{argmax}\{p(x = c \mid \Theta)\}$$
$$\equiv \underset{c}{argmax}\{\theta_c\}$$

  - So the decoding is the category 'c' where θ_c is the largest.

Inference Task: Computing Dataset Probabilities

- Inference task : given Θ and IID data, compute $p(x^1, x^2, \ldots, x^n \mid \Theta)$.
  - The likelihood of training/validation/testing data.

- Assuming "independence of IID data given parameters", we have:

$$p(x^1, x^2, \ldots, x^n \mid \Theta) = \prod_{i=1}^{n} p(x^i \mid \Theta) \qquad \text{(this is the conditional independence assumption)}$$

$$= \prod_{i=1}^{n} \theta_1^{I(x^i=1)} \theta_2^{I(x^i=2)} \cdots \theta_k^{I(x^i=k)} \qquad \text{(definition of categorical distribution)}$$

$$= \theta_1^{\sum_{i=1}^{n} I(x^i=1)} \theta_2^{\sum_{i=1}^{n} I(x^i=2)} \cdots \theta_k^{\sum_{i=1}^{n} I(x^i=k)} \qquad (\theta_1 \theta_1 \theta_1 \theta_2 \theta_1 \theta_2 = \theta_1^4 \theta_2^2)$$

$$= \theta_1^{n_1} \theta_2^{n_2} \cdots \theta_k^{n_k} \qquad \left(n_c = \sum_{i=1}^{n} I(x^i = c)\right)$$

- Where n_1 is "number of 1s", n_2 is "number 2s", and so on.
  - Similar to the Bernoulli, the likelihood only depends on the counts.

Code for Categorical Likelihood

- We just showed that the categorical likelihood can be written:

$$p(x^1, x^2, \ldots, x^n \mid \theta) = \theta_1^{n_1} \theta_2^{n_2} \cdots \theta_k^{n_k}$$

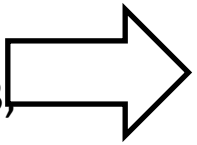
```
nc = zeros(k)
for i in 1:n
    nc[X[k]] += 1
end
logLik = 0
for c in 1:k
    logLik += nc[c] * log(theta[c])
end
```

- Will be very small for large n.
  - Compute the log-likelihood in practice.
- Runtime: $O(n + k)$.
  - If $n >> k$ (many samples, few categories), this is $O(n)$.
  - If $k >> n$ (many categories, few samples), you could also get $O(n)$.
    - Only track the classes with non-zero counts.

Inference Task: Sampling

- Inference task: given Θ,
  generate samples of '*X*' distributed according to p(x | Θ).

Pr(*X* = LIB) = 0.34, Pr(*X*=NDP) = 0.34,
Pr(*X* = CPC) = 0.27, Pr(*X*=GRN) = 0.03,
Pr(*X* = PPC) = 0.02.



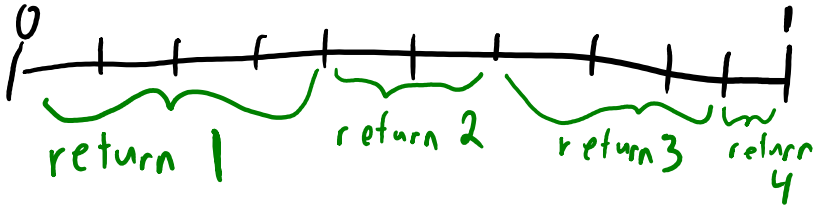
Party?
LIB
CPC
NDP
LIB
GRN

- Notice that we are not "sampling a value for each of the classes".
  - Each sample will belong to one category.
    - About 34% of the samples should be LIB, 27% will be CPC, and so on.

Inference Task: Sampling

- Recall we assume we can sample uniformly between 0 and 1.
  - And we want to turn this into a sample over the *k* categories.

- Sampling from categorical distribution with $\Theta = \{0.4, 0.2, 0.3, 0.1\}$:
  - Generate a uniform sample *u*.
  - If u < 0.4, return 1 (like sampling from a Bernoulli with $\theta = 0.4$).
  - If u > 0.4 but less than 0.6, return 2 (like sampling Bernoulli with $\theta = 0.2$).
  - If u > 0.6 but less than 0.9, return 3 (like sampling Bernoulli with $\theta = 0.3$).
  - If u > 0.9, return 0 (like sampling Bernoulli with θ = 0.1).



Inference Task: Sampling

- Formally, the sampler "returns the *c* such that $p(x \leq c-1) < u < p(x \leq c)$".
  - Where the CDF for categorical is $p(x \leq c) = \theta_1 + \theta_2 + \cdots + \theta_c$.

- Sampling from a categorical distribution with 'k' categories:
  1. Generate 'u' uniformly on the interval between 0 and 1.
  2. If $u \leq p(x \leq 1)$, return 1.
  3. If $u \leq p(x \leq 2)$, return 2.
  4. If $u \leq p(x \leq 3)$, return 3.
  5. …

- Runtime:
  - If you compute $Pr(x \leq c)$ from scratch at each step, costs $O(k^2)$.
  - If you use that $Pr(x \leq c) = Pr(x \leq c-1) + \theta_c$, costs $O(k)$.

Inference Task: Sampling

- In code:

```
u = rand()
CDF = 0
For c in 1:K
    CDF += theta[c]
    if u < CDF
        return c
    end
end
```

```julia
### For vector p giving discrete probabilities, generates a random sample
function sampleDiscrete(p)
    findfirst(cumsum(p[:]).> rand())
end
```

- Calling this function each time costs O(k).

- You can go faster if you have CDF values stored.
  - In this case, do a binary search for the *c* such that $Pr(x \leq c-1) < u < Pr(x \leq c)$.
- Using this faster procedure, it costs O(k + t log k) to generate 't' samples.
  - O(k) to compute all the CDFs.
  - O(log k) to do a binary search to generate each sample.

bonus!

Even faster sampling: Alias method

- Previous method (sometimes called “roulette wheel sampling”): O(k) preprocessing time, O(log k) time per sample
- Vose’s alias method: O(k) preprocessing, O(1) time per sample

- Nice (long) article building up to it by Keith Schwarz (Stanford CS): Darts, Dice, and Coins: Sampling from a Discrete Distribution

Next Topic: Monte Carlo Approximation

Motivation: Probabilistic Inference

- Given a probabilistic model, we often want to make inferences:
  - Marginals: what is the probability that $X_j = c$?
  - Conditionals: what is the probability that $X_j = c$ given $X_{j'} = c'$?

- This is simple for the models we have seen so far.
  - For Bernoulli/categorical, computing probabilities is straightforward.
  - For multivariate models, we assumed everything was independent.
    - Perhaps conditioned on a label or the final hidden layer of a neural network.

- For many models, inference has no closed form or is NP-hard.
  - For these problems, we often use Monte Carlo approximations.

Monte Carlo: Marginalization by Sampling

- A basic Monte Carlo method for estimating probabilities of events:
  - Generate a large number of samples x^i from the model:

$$X = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

  - Compute frequency that the event happened in the samples:

$$\Pr(X_2=1) \approx \frac{3}{4} \qquad \Pr(X_3=0) \approx 0$$

- Monte Carlo methods are the second most important type of ML algorithms.
  - Modern versions originally developed to build better atomic bombs ☹
    - Runs physics simulator to "sample", then see if it leads to a chain reaction.

Monte Carlo for Approximating Probabilities

- Monte Carlo estimate of the probability of an event A:

$$\frac{\text{number of samples where } A \text{ happened}}{\text{number of samples}}$$

- You can think of this as the MLE for a binary variable:
  - The binary variable is 1 for samples where *A* happened, 0 otherwise.
- Approximating probability of a pair of independent dice rolling a sum of 7:
  - Roll two dice, check if the sum is 7.
  - Roll two dice, check if the sum is 7.
  - Roll two dice, check if the sum is 7.
  - Roll two dice, check if the sum is 7.
  - Roll two dice, check if the sum is 7.
  - …
  - Monte Carlo estimate: fraction of samples where sum is 7.

Monte Carlo for Approximating Probabilities

- Recall our motivating problem:
  - Building a model of voters among categories (LIB, CPC, NDP, GRN, PPC).
- You might consider the following inference problem:
  - In 100 samples, what the probability that $n_{LIB} > \max\{n_{CPC}, n_{NDP}, n_{GRN}, n_{PPC}\}$?
    - "What is the probability that LIBs win the election again?"

- You can do some math to figure out the answer, or do Monte Carlo:
  - Generate 100 samples, check who won.
  - Generate 100 samples, check who won.
  - …
  - Approximate probability by fraction of times they won.

Monte Carlo for Inequalities

- Consider probability that a variable is above a threshold.
  - Probability that a beta variable is above 0.7.
  - Probability that a standard normal variable is above -1.2.
- Monte Carlo estimate for $\Pr(X \leq \tau)$ for some threshold τ:
  - Fraction of samples that are above the threshold.

Monte Carlo Method for the Mean

- A Monte Carlo approximation of the mean:
  - Approximate the mean by the mean of the samples.

$$\mathbb{E}[X] \approx \frac{1}{n} \sum_{i=1}^{n} x^i$$

- A Monte Carlo approximation of expected value of X^2:
  - Approximate $\mathbb{E}[X^2]$ by the average value of $(x)^2$ on the samples.

- A Monte Carlo approximation of the expected value of function g.
  - Approximate $\mathbb{E}[g(X)]$ by the average value of g(x) on the samples.

Monte Carlo Method: Definition

- Monte Carlo approximates expectation of random functions:

$$\mathbb{E}[g(X)] = \sum_{x \in \mathcal{X}} g(x)\, p(x)$$

pmf of discrete variable X

$$\mathbb{E}[g(X)] = \int_{x \in \mathcal{X}} g(x)\, p(x)\, dx$$

pdf of continuous X

- Computing mean is a special case: use g(x) = x.
- Computing probability of an event *A* is also a special case:
  - Set g(x) = 𝟙["A happened in sample x"], indicator function for event *A*.

- Monte Carlo methods generate n samples x^i from X, then use:

$$\mathbb{E}[g(X)] \approx \frac{1}{n} \sum_{i=1}^{n} g(x^i)$$

Summary of Monte Carlo Theory

- Let $\mu = \mathbb{E}[g(X)]$, the value we want to compute.
  - And assume variance of g(X), σ^2, exists and is bounded ("not infinite").
- With IID samples, Monte Carlo gives an unbiased approximation of μ.
  - Expected value of Monte Carlo estimate, averaged over samplings, is μ.
- Monte Carlo estimate "converges" to μ as sample size *n* goes to ∞.
  - Estimate get arbitrarily close to μ as your number of samples gets large.
- Expected squared error between estimate and μ is σ^2/n with *n* samples.
  - This is the speed at which you converge to μ (in squared error) as you increase *n*.
- Monte Carlo can be written as a special case of SGD.
  - See the post-lecture bonus slides for some details on all of the above.

Summary

- Pixel classification:
  - Assigning a label to every pixel in an image.
- Fully-convolutional networks (FCNs):
  - CNNs where every layer maintains spatial information.
  - Useful for handling images of different sizes.
  - Requires upsampling to be used for pixel classification.
  - Transpose convolutions learn upsampling operators.

- Categorical distribution:
  - Probability over unordered categories, where p(x = c | Θ) = θ_c.
- Inference in categorical models:
  - CDF, decoding, likelihood, sampling.
- Monte Carlo methods:
  - Approximate expectations of random functions.
  - Generate a set of IID samples.
  - Take average value of function value applied to each sample.

- Next time: why we're doing everything wrong to make decisions.

bonus!

Law of the Unconscious Statistician

$$\mathbb{E}[g(X)] = \sum_{x \in X} g(x)\, p(x)$$

p: pmf of discrete variable X

$$\mathbb{E}[g(X)] = \int_{x \in X} g(x)\, p(x)\, dx$$

p: pdf of continuous X

- These equalities sometimes called "Law of the Unconscious Statistician"
  - ("LOTUS", when I took intro stats)
  - "Unconscious" because people don't realize this is actually a theorem to prove

$Y = g(X)$

$$\mathbb{E}[Y] = \sum_{y} y \Pr(Y=y) = \sum_{y} y \sum_{x: g(x)=y} p(x) = \sum_{x} g(x)\, p(x)$$

each x only maps to a single y

Unbiasedness of Monte Carlo Methods

bonus!

- Let $\mu = \mathbb{E}[g(x)]$ be the value we want to approximate (not necessarily mean).

- The Monte Carlo estimate is an unbiased approximation of μ,

$$\begin{aligned}
\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} g(x^i)\right] &= \frac{1}{n}\mathbb{E}\left[\sum_{i=1}^{n} g(x^i)\right] && \text{(linearity of } \mathbb{E}\text{)} \\
&= \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}[g(x^i)] && \text{(linearity of } \mathbb{E}\text{)} \\
&= \frac{1}{n}\sum_{i=1}^{n} \mu && (x^i \text{ is IID with mean } \mu) \\
&= \mu.
\end{aligned}$$

- The law of large numbers says that:
  - Unbiased approximators "converge" (probabilistically) to expectation as $n \to \infty$.
  - So the more samples you get, the closer to the true value you expect to get.

Rate of Convergence of Monte Carlo Methods

bonus!

- Let f be the squared error in a 1D Monte Carlo approximation,

$$f(x^1, x^2, \ldots, x^n) = \left(\frac{1}{n} \sum_{i=1}^{n} g(x^i) - \mu \right)^2 .$$

- If variance is bounded, error with n samples is $O(1/n)$,

$$\begin{aligned}
\mathbb{E}\left[\left(\frac{1}{n} \sum_{i=1}^{n} g(x^i) - \mu \right)^2 \right] &= \mathsf{Var}\left[\frac{1}{n} \sum_{i=1}^{n} g(x^i) \right] && \text{(unbiased and def'n of variance)} \\
&= \frac{1}{n^2} \mathsf{Var}\left[\sum_{i=1}^{n} g(x^i) \right] && (\mathsf{Var}(\alpha x) = \alpha^2 \mathsf{Var}(x)) \\
&= \frac{1}{n^2} \sum_{i=1}^{n} \mathsf{Var}[g(x^i)] && \text{(IID)} \\
&= \frac{1}{n^2} \sum_{i=1}^{n} \sigma^2 = \frac{\sigma^2}{n}. && (x^i \text{ is IID with var } \sigma^2)
\end{aligned}$$

- Similar $O(1/n)$ argument holds for $d > 1$ (notice that faster for small σ^2).

Monte Carlo as a Stochastic Gradient Method

bonus!

- Monte Carlo approximation as a stochastic gradient method with $\alpha_i = 1/(i+1)$,

$$
\begin{aligned}
w^n &= w^{n-1} - \alpha_{n-1}(w^{n-1} - x^i) \\
&= (1 - \alpha_{n-1})w^{n-1} + \alpha_{n-1}x^i \\
&= \frac{n-1}{n}w^{n-1} + \frac{1}{n}x^i \\
&= \frac{n-1}{n}\left(\frac{n-2}{n-1}w^{n-2} + \frac{1}{n-1}x^{i-1}\right) + \frac{1}{n}x^i \\
&= \frac{n-2}{n}w^{n-2} + \frac{1}{n}\left(x^{i-1} + x^i\right) \\
&= \frac{n-3}{n}w^{n-3} + \frac{1}{n}\left(x^{i-2} + x^{i-1} + x^i\right) \\
&= \frac{1}{n}\sum_{i=1}^{n} x^i.
\end{aligned}
$$