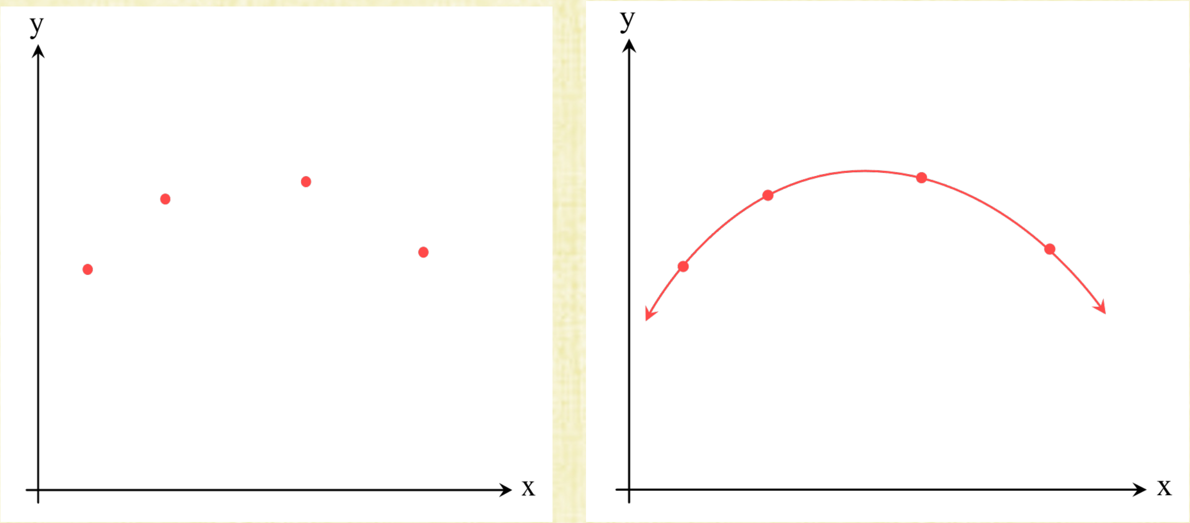
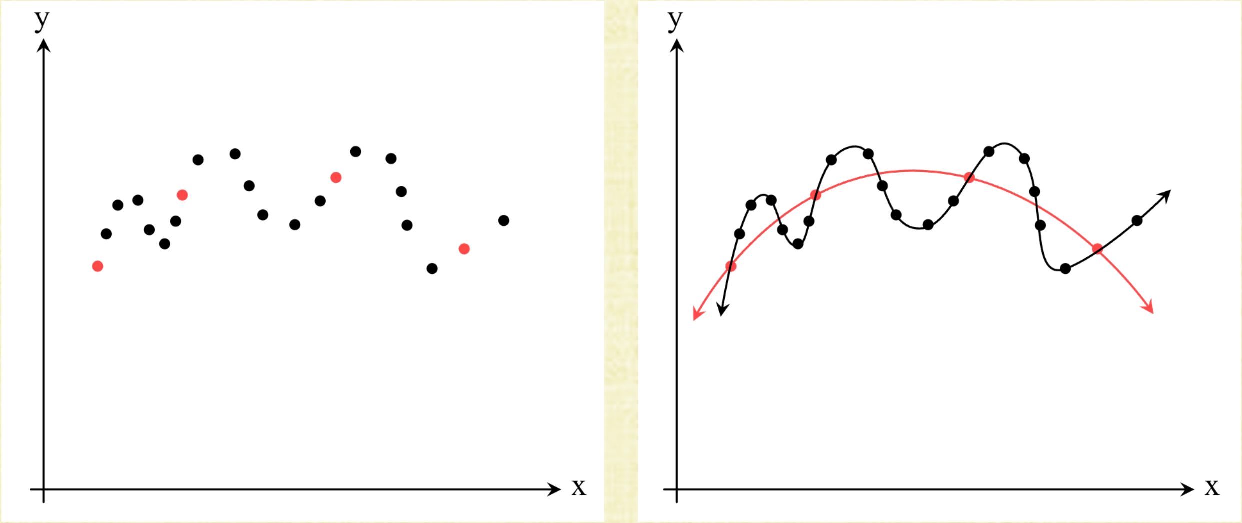
Local Approximations

Sampling

- Accurate approximation of a function is often limited by the amount of available data
- Given too few samples (left), one may ”hallucinate” an incorrect function
- Adding more data allows for better/proper feature resolution (right)
- Given “enough” sample points, a function tends to not vary too much in between them



under-resolved



resolved better with more data

Taylor Expansion

- $f(x+h) = \sum_{p=0}^{\infty} \frac{h^p}{p!} f^{(p)}(x) = \sum_{p=0}^{\hat{p}} \frac{h^p}{p!} f^{(p)}(x) + O(h^{\hat{p}+1})$
- Bounded derivatives would indicate that $O\left(h^{\hat{p}+1}\right) \to 0$ as $h \to 0$

- Examples:
  - $f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(x) + \frac{h^3}{6} f'''(x) + O(h^4)$ looking forward
  - $f(x-h) = f(x) - hf'(x) + \frac{h^2}{2} f''(x) - \frac{h^3}{6} f'''(x) + O(h^4)$ looking backward

- <u>Truncated Taylor expansions</u> become more valid approximations as $h \to 0$
  - $f(x+h) \approx f(x) + hf'(x) + \frac{h^2}{2} f''(x) + \frac{h^3}{6} f'''(x)$ looking forward
  - $f(x-h) \approx f(x) - hf'(x) + \frac{h^2}{2} f''(x) - \frac{h^3}{6} f'''(x)$ looking backward

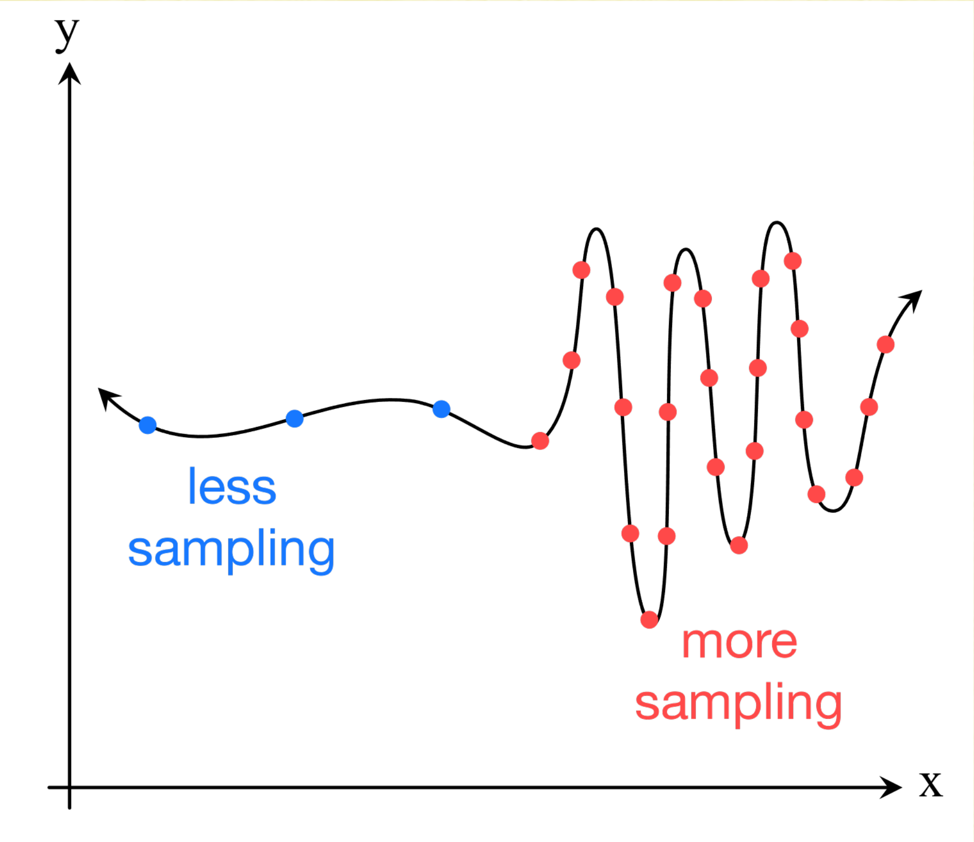
Well-Resolved Functions

- The Taylor expansion approximates a function f at a new location $x + h$ based on known information at a nearby point x
- When the sample points are "closely" spaced, new locations are "close" to known sample points making h "small" enough
- However, large derivatives can overwhelm even a small h
- Thus, functions with more variation need higher sampling rates
  - Similarly, smoother functions can utilize lower sampling rates

- Well-resolved functions have vanishing high order terms in their Taylor expansion making truncated Taylor expansions more valid

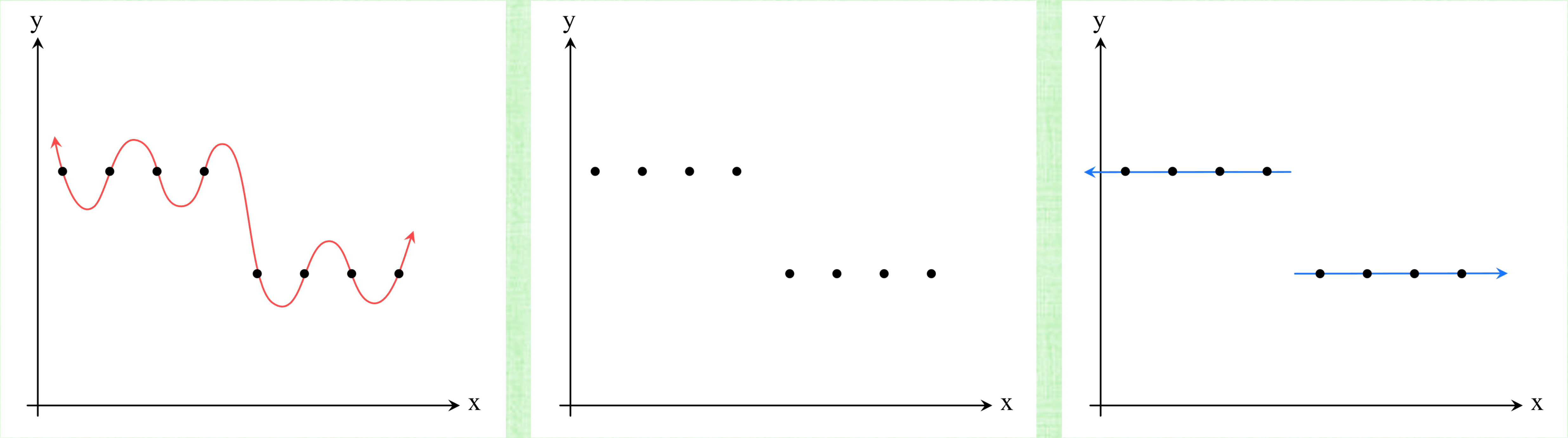
Well-Resolved Functions

- Regions of a function with less/more variation require lower/higher sampling rates



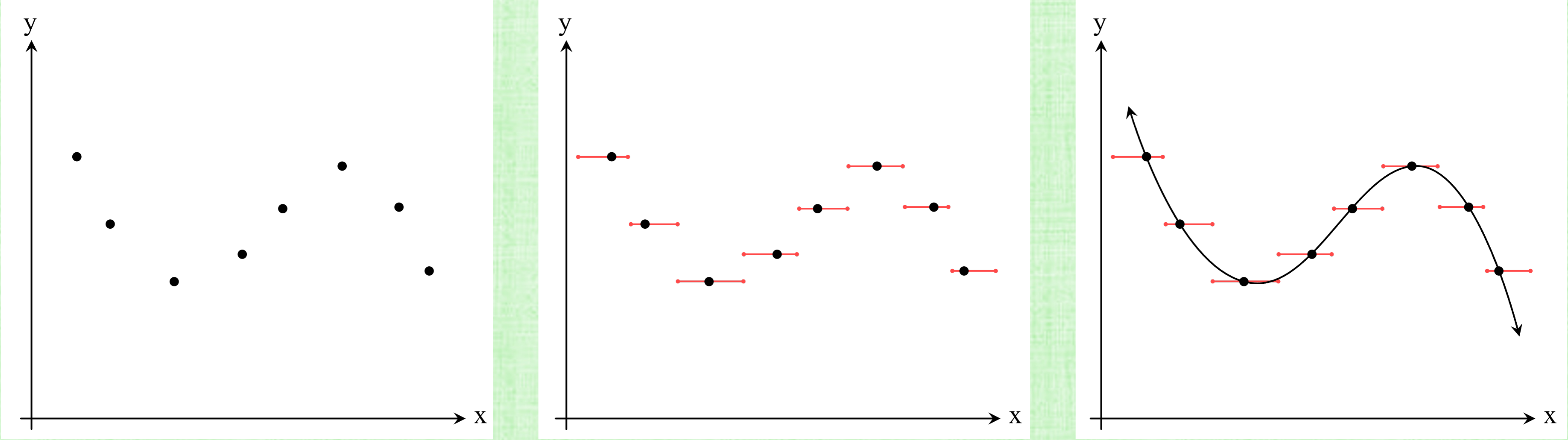
Piecewise Approximation

- Piecewise approximation enables the use of simpler models to approximate (potentially disjoint) subsets of data
  - In ML/DL, “sub-manifold” often refers to a coherent subset



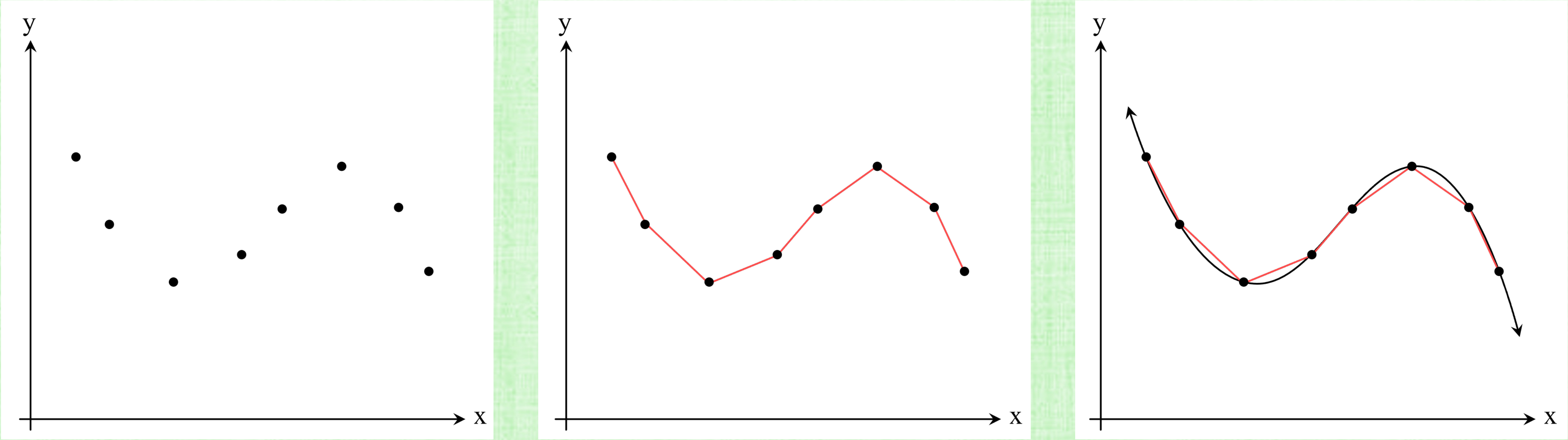
Piecewise Constant Interpolation

- Use the first term in the Taylor expansion (only): $f(x+h) \approx f(x)$
- Errors are $O(h)$, since $f(x+h) = f(x) + O(h)$
- Recall: nearest neighbor is piecewise constant



Piecewise Linear Interpolation

- Use the first two terms in the Taylor expansion: $f(x+h) \approx f(x) + hf'(x)$
- Errors are $O(h^2)$, since $f(x+h) = f(x) + hf'(x) + O(h^2)$



Higher Order Piecewise Interpolation

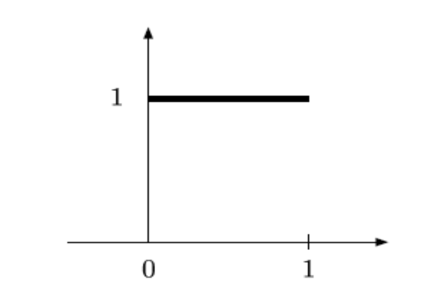
- Piecewise quadratic interpolation uses the first three terms in the Taylor expansion and has $O(h^3)$ errors

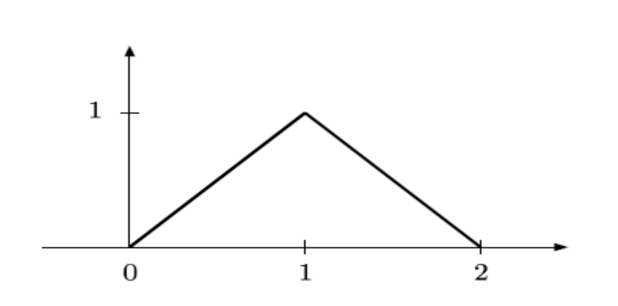
- Piecewise cubic interpolation uses the first four terms in the Taylor expansion and has $O(h^4)$ errors

- Recall: higher order interpolation becomes more oscillatory (i.e. overfitting)
  - These oscillations are sometimes referred to as Gibbs phenomena

Piecewise Cubic Interpolation (B-Splines)

- Piecewise cubic splines are quite popular because of their ability to match derivatives across approximation boundaries
- B-splines – hierarchical family: ϕ_i^p is a piecewise polynomial of degree p
  - Piecewise constant: $\phi_i^0(x) = 1$ for $x \in [x_i, x_{i+1}]$ and 0 otherwise
  - A linear $w_i^p(x) = \frac{x - x_i}{x_{i+p+1} - x_i}$ increases the polynomial degree of ϕ^p to ϕ^{p+1}
  - Recursively: $\phi_i^{p+1}(x) = w_i^p(x)\phi_i^p(x) + \left(1 - w_{i+1}^p(x)\right)\phi_{i+1}^p(x)$
  - Piecewise linear ϕ_i^1, piecewise quadratic ϕ_i^2, piecewise cubic ϕ_i^3, etc.





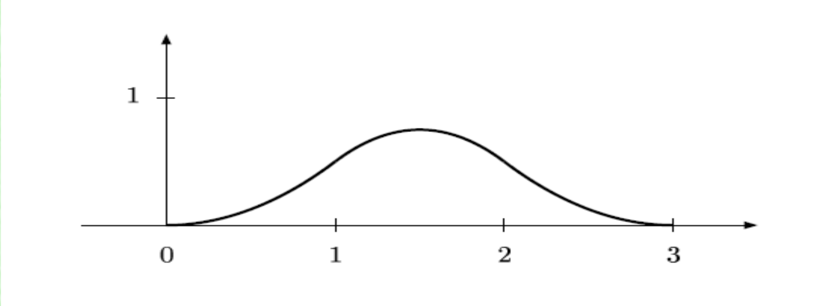


Image Segmentation

- Divide image pixels into separate regions, each representing separate objects or groups of objects
- Before neural networks: various methods relied on clustering in color and/or space, graph-cuts, edge detection, etc.

- Since humans do well on this problem, use neural networks to hopefully mimic <u>human perception/semantics</u>
- Training examples:
  - Input: an <u>image</u> (all the pixel RGB values)
  - Output: <u>labels</u> on all the pixels, indicating what group each pixel is in

Bool Output Labels

- Binary segmentation of an image
- E.g. true = dog, false = not dog

Input

Output

Integer Output Labels

- Multi-object segmentation with an integer for each object
- E.g. 1=cat, 2=dog, 3=human, 4=mug, 5=couch, 6=everything else

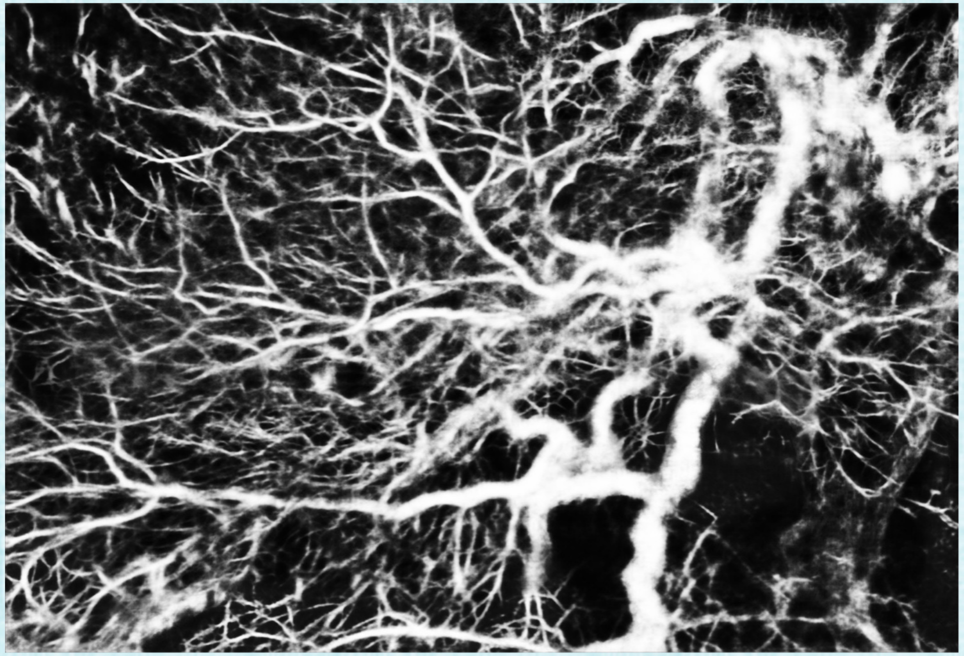
Input

Output

Real Number Output Labels

- Probabilistic segmentation with real number values in $[0,1]$
- E.g. 1=tree branch, .8=probably a branch, .2=probably not a branch, etc.

Input



Output

Segmenting Botanical Trees

Difficult Problem:

- Trees are large-scale and geometrically-complex structures
- Branches severely occlude each other
- The images have limited pixel resolution of individual branches

- Even humans have a hard time ascertaining the correct topological structure from a single image/view
- Can we train a neural network to help?

Constructing Training Data

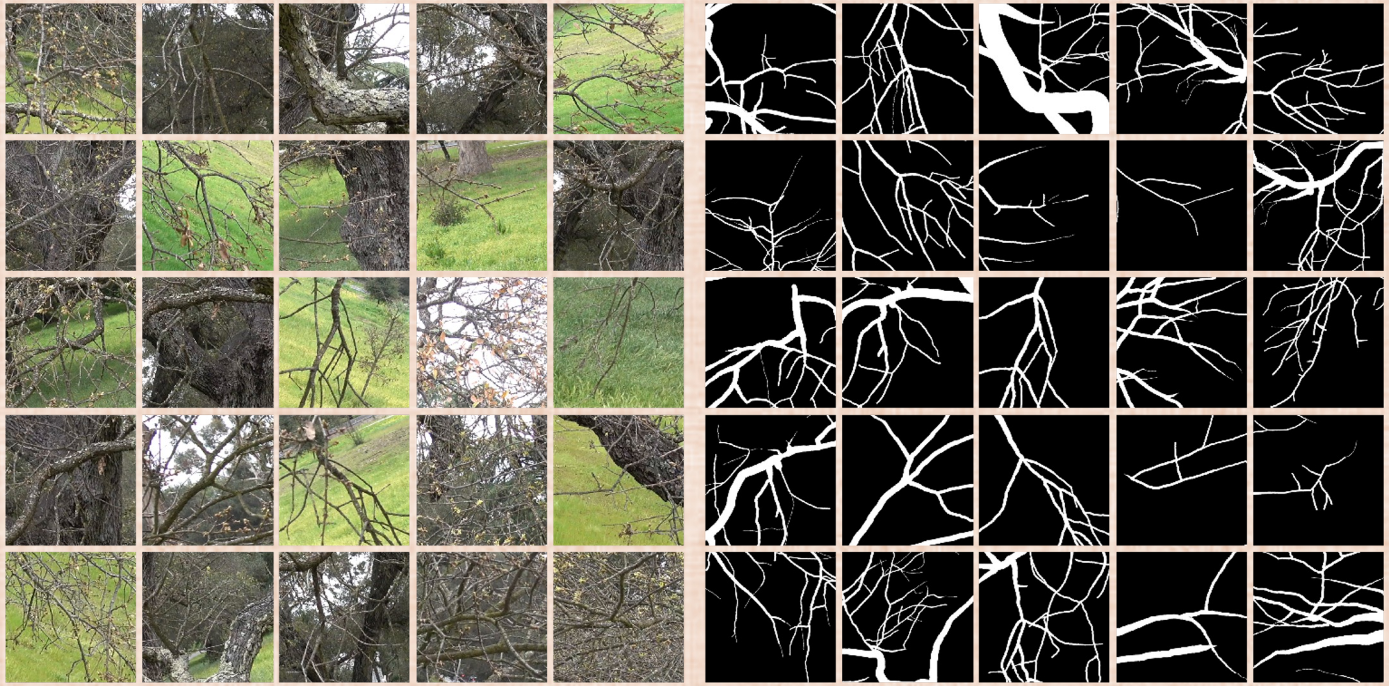
- Begin with a dataset of labels (tediously) created by hand
- Draw lines and thicknesses on top of branches; then, use this information to create a <u>binary mask</u> for the image



Constructing More Training Data

- Artificially increase the amount of training data by taking various image subsets
- This also helps to avoid down-sampling (networks use low-resolution images)

3840 pixels wide, 2160 pixels tall



each image: 512 pixels wide, 512 pixels tall

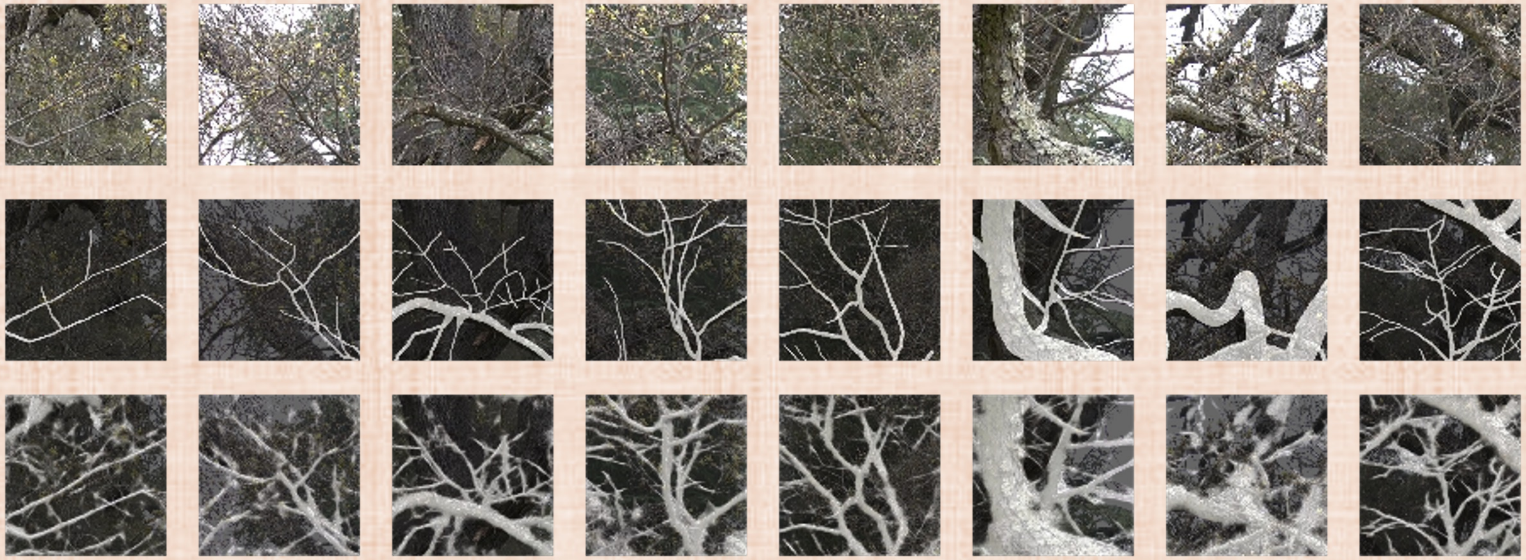
Training the Neural Network

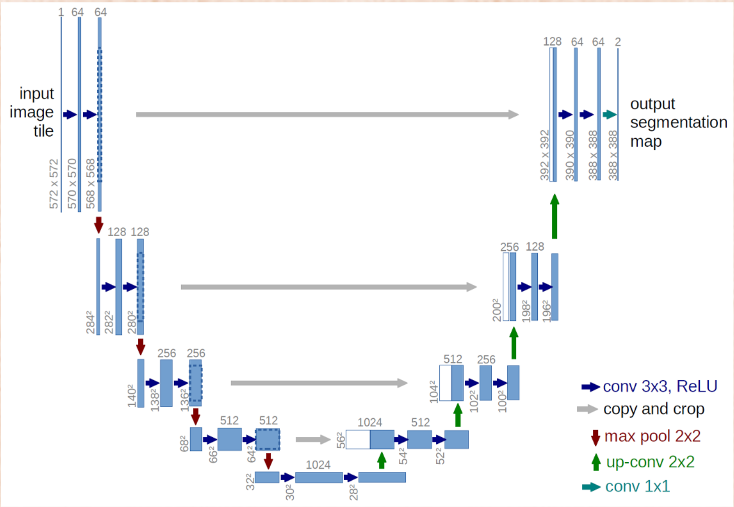
- Find function parameters c such that the network function $f_c(x)$ gives minimal error on the training data (i.e. minimize network "loss")
- The network should predict the known target labels (or close to it) from the input images

Input images

Target labels

Network outputs

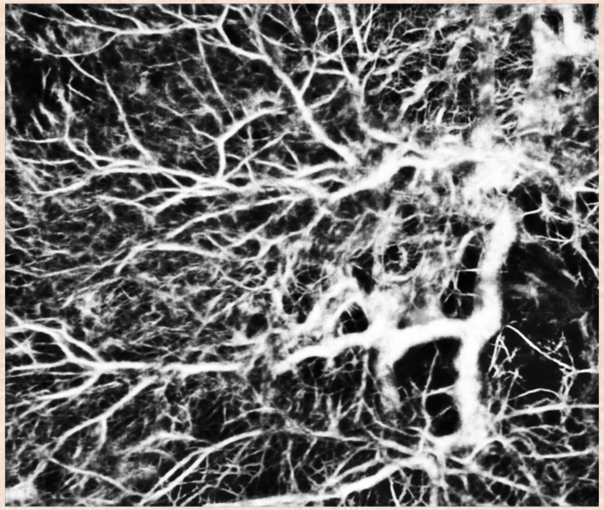


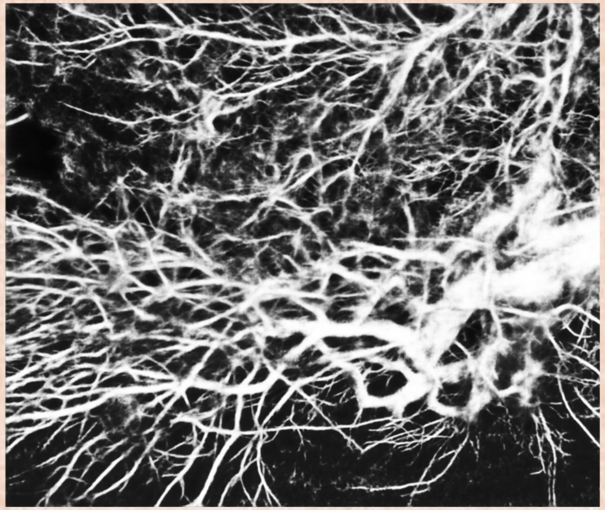


architecture

Network Inference/Prediction

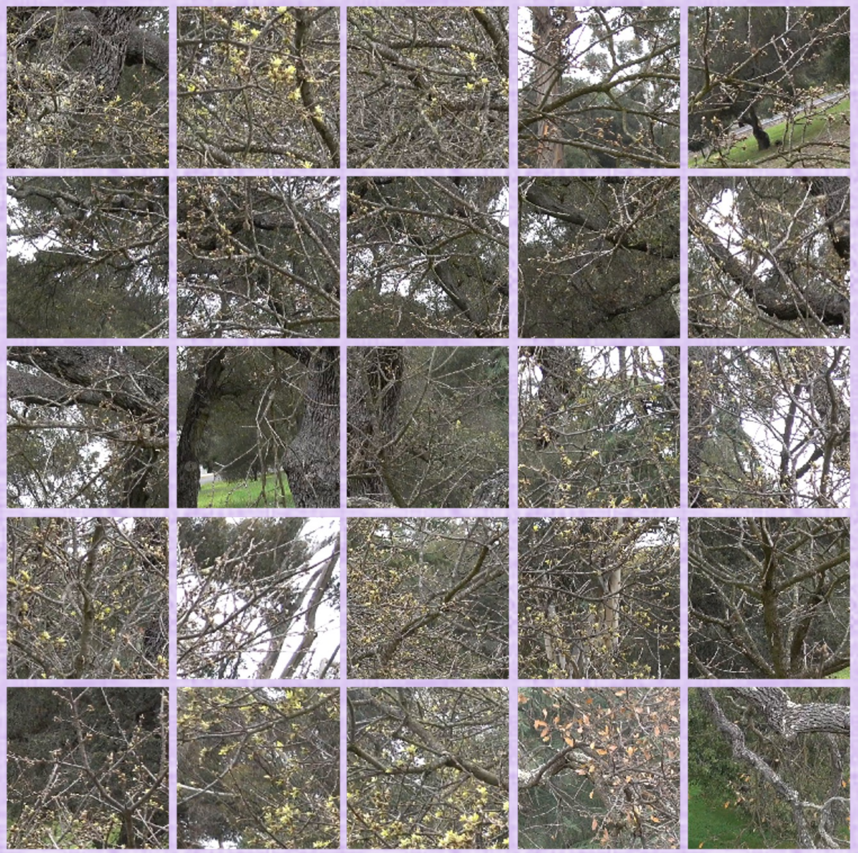
- After training, use the resulting network function $f_{c_{trained}}(x)$ to infer/predict labels for new images (not previously hand-labeled)



Local Approximations

- Roughly speaking, input images mostly seem to be of two different types: either (1) branches over grass or (2) clusters of branches



Train 2 Neural Networks

- Divide the training data into these two disparate groups
- Train a separate network on each group: separate architecture, separate trainable parameters, etc.

- k-means clustering on hue/saturation was used to divide the training images into 2 separate clusters
- Then, each cluster was used to train a network

Combining Inference Outputs

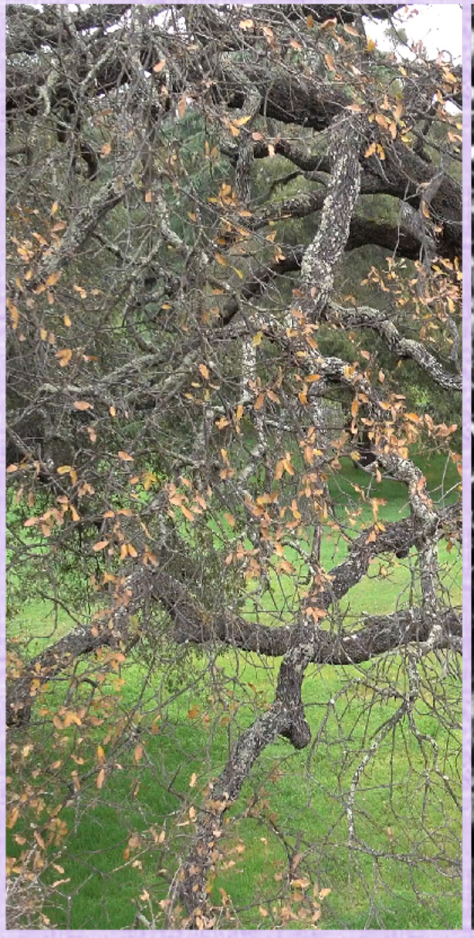
- Given an input image, inference it (separately) on both networks
- Then combine the two predictions, using the network that makes the most sense locally in each part of the image (blending predictions when appropriate)

To inference each pixel:

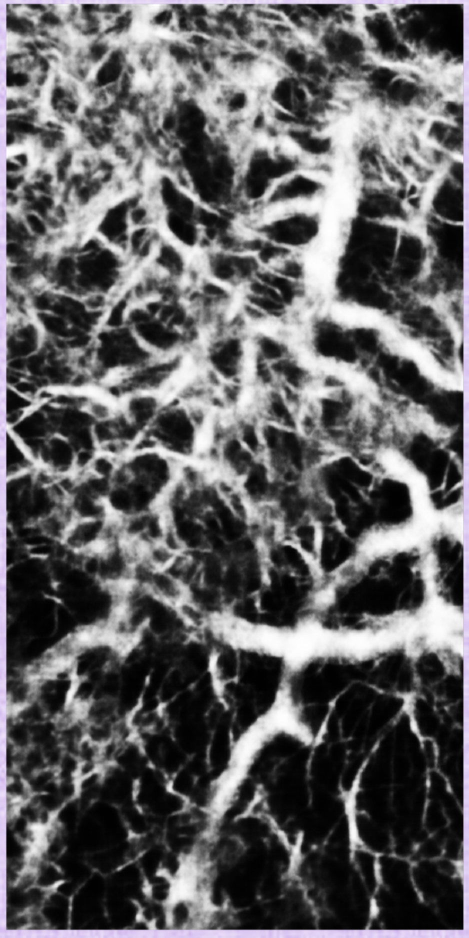
- Compute hue/saturation values on a small patch around the pixel
- Find the distances from the patch hue/saturation values to the 2 cluster centers
- Interpolate the outputs from the 2 networks using those distances

- The closer a pixel is to a k-means cluster, the more weight is given to that cluster's network inference/prediction

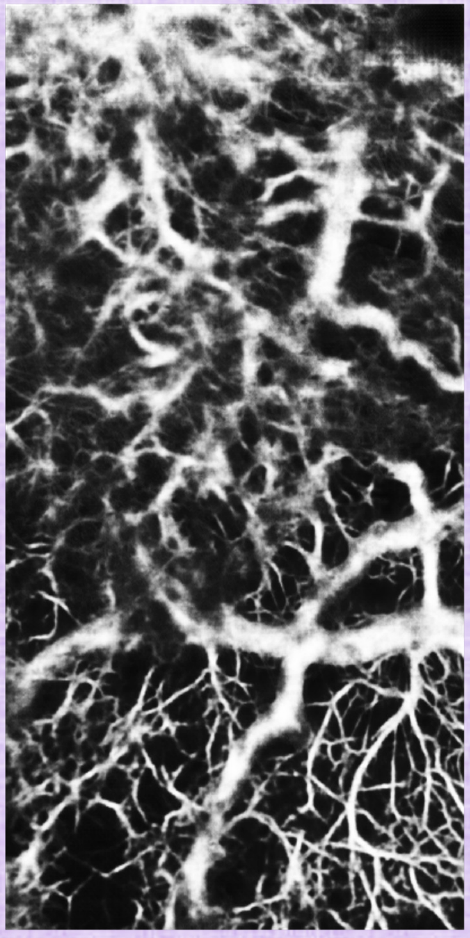
Example



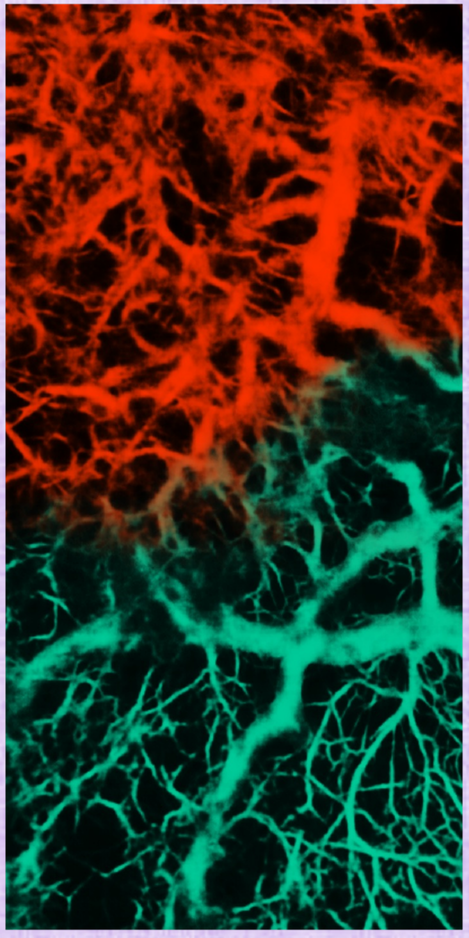
Input



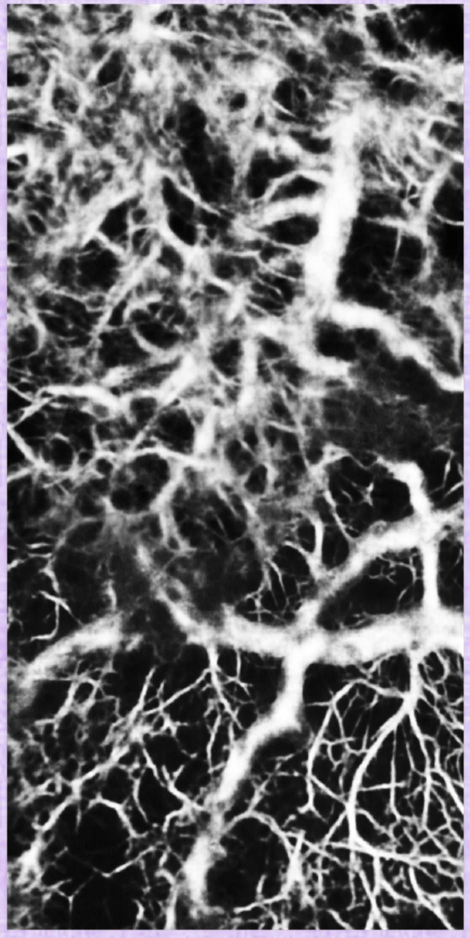
Network 1



Network 2



Combine



Final Result

Branch Estimation