Deep Forests

Alexander C. Mueller, PhD

May 21, 2018

Agenda

In addition to covering the paper, we are going to provide some context and elaboration...

- What is a neural net?
- What is a decision tree?
- What is representation learning?
- What's in the paper? Why?

It was tough to avoid putting some machine learning jargon in the slides without definition but we can probably verbally coach up the uninitiated along the way.

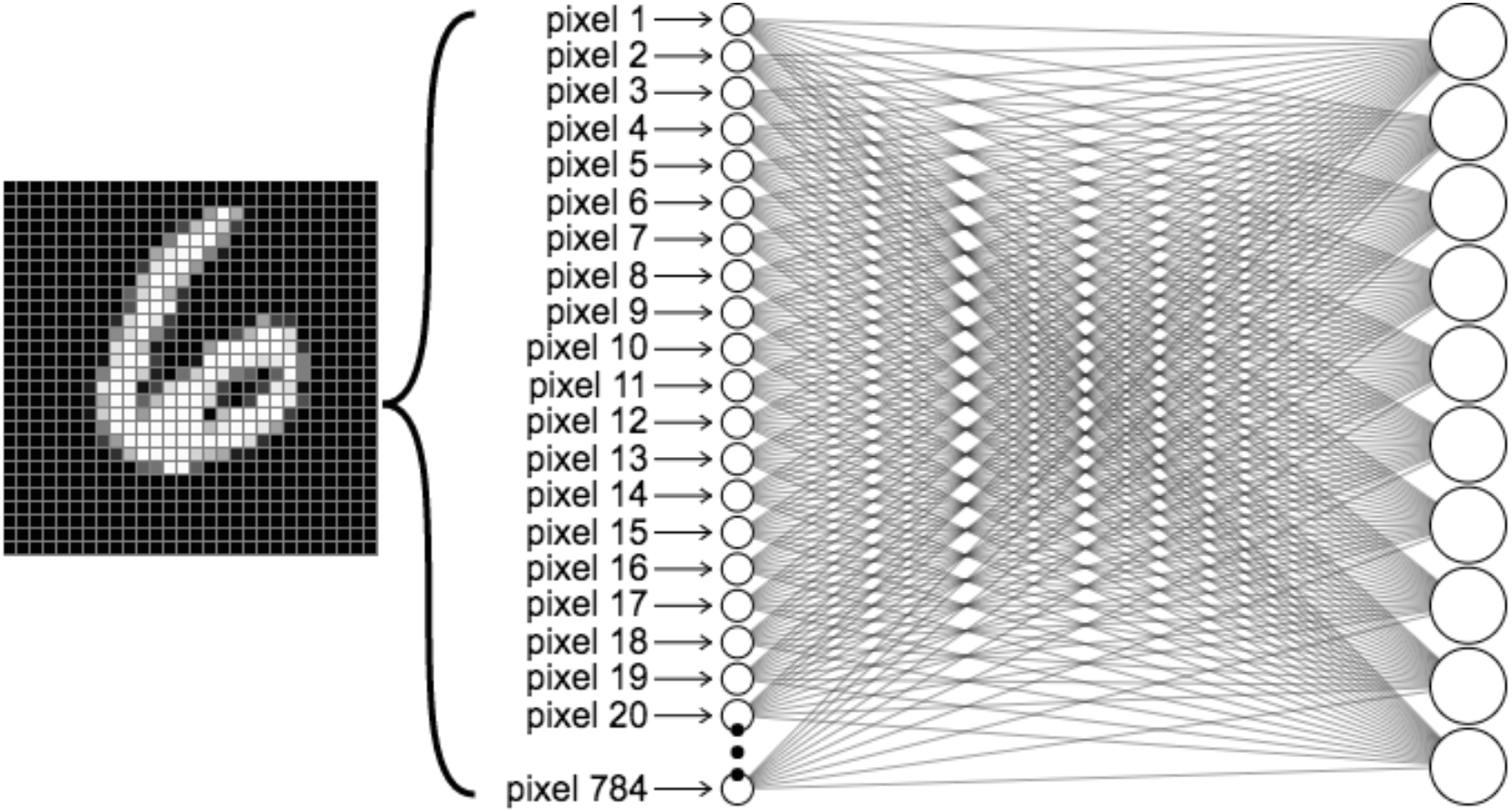
Neural Nets: Basics 1

A neural network ...

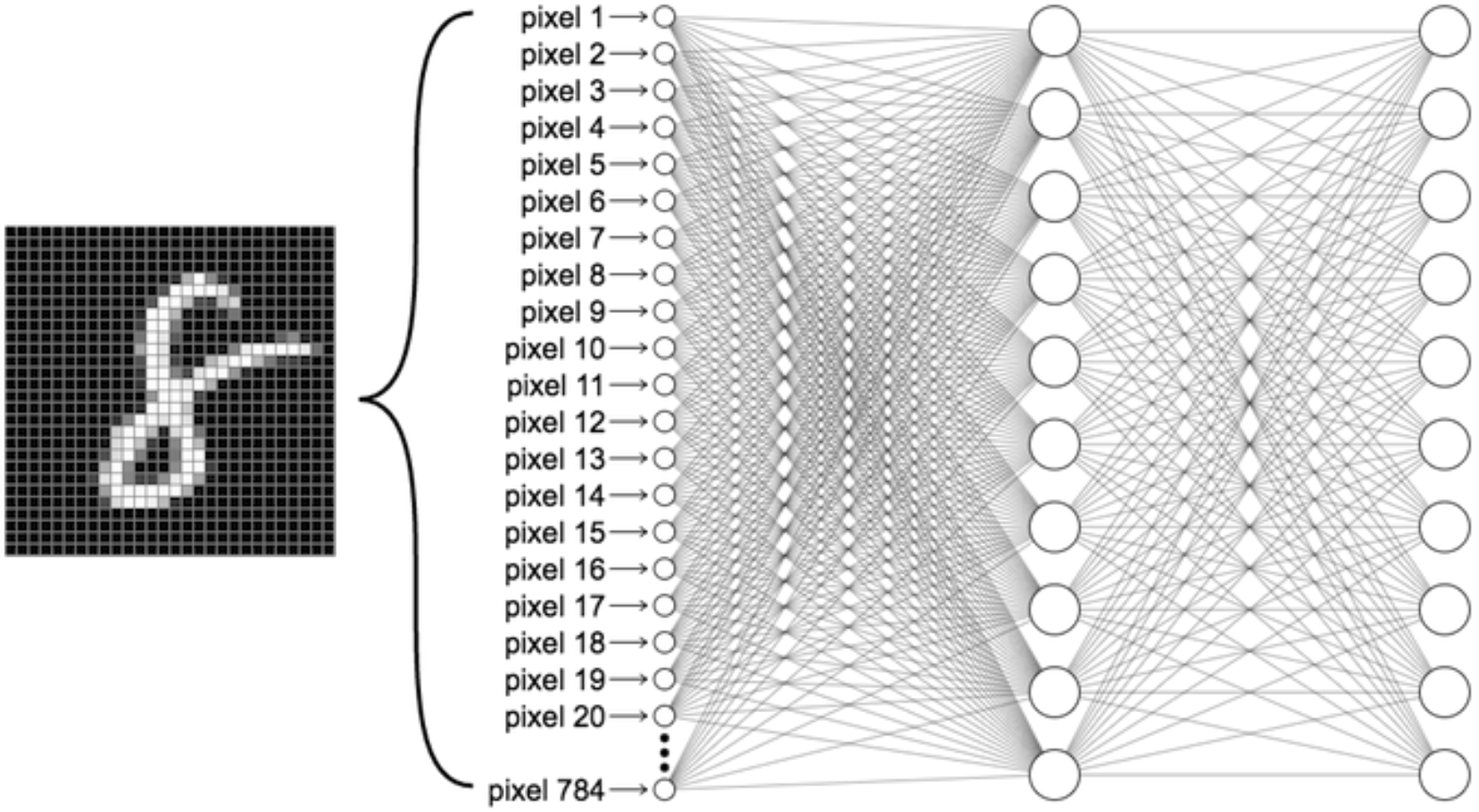
- is a graph, often organized into layers.
- has a real number for each node, often between 0 and 1, called an activation level.
- has a real number for each edge called a weight.
- has an activation function that uses the weights and activation levels in one layer to compute new activation levels for the next layer.

Neural networks are trained by optimizing their weights so that a loss function is minimized over the training data.

Neural Nets: Basics 2



Neural Nets: Basics 3



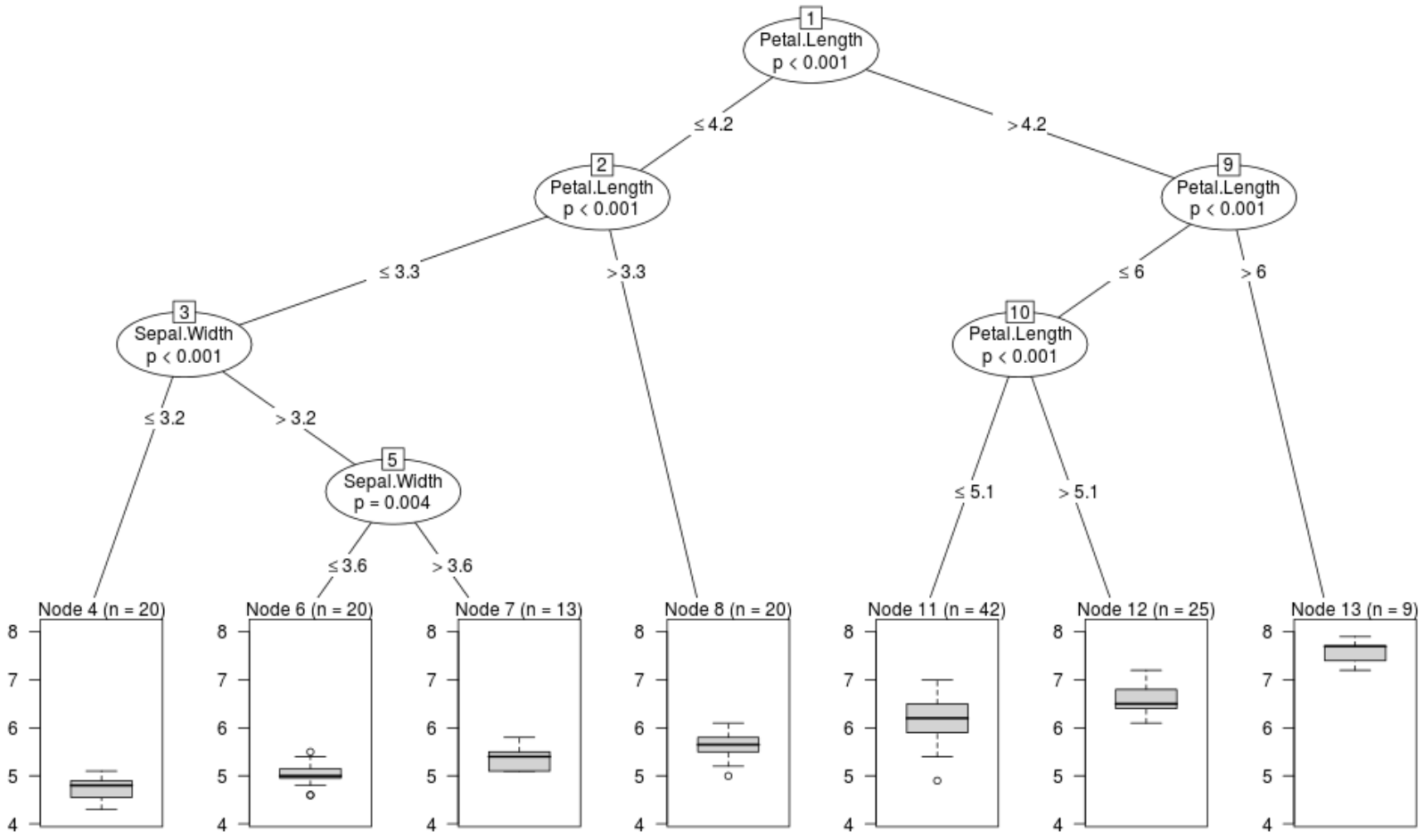
Decision Trees: Basics 1

A decision tree ...

- is an acyclic binary graph and decision flowchart.
- has branching points defined via “splits” of the feature space.
- has leaves that correspond to components in a partition of the feature space.
- has a key trait called depth, roughly how many splits it contains.

Decision trees are trained by a greedy algorithm that chooses splits to minimize variance in the response variable. The algorithm splits one feature at a time, and it will be important later that this amounts to implicit feature selection.

Decision Trees: Basics 2



Contrasts Relevant to the Paper

The output of a neural network can be written as a differentiable function of the weights, and the training process involves taking partial derivatives of this function. **In contrast**, the output of a decision tree is typically not even a continuous function of the inputs. **The paper** discusses this issue via a *(non-)differentiable modules* terminology.

Neural networks require many choices prior to training, relating to how to initialize the weights for example. **In contrast**, decision trees require few choices prior to training. **The paper** discusses this issue via a *hyperparameters* terminology.

Decision trees have very minimal requirements for prior data cleaning and formatting.

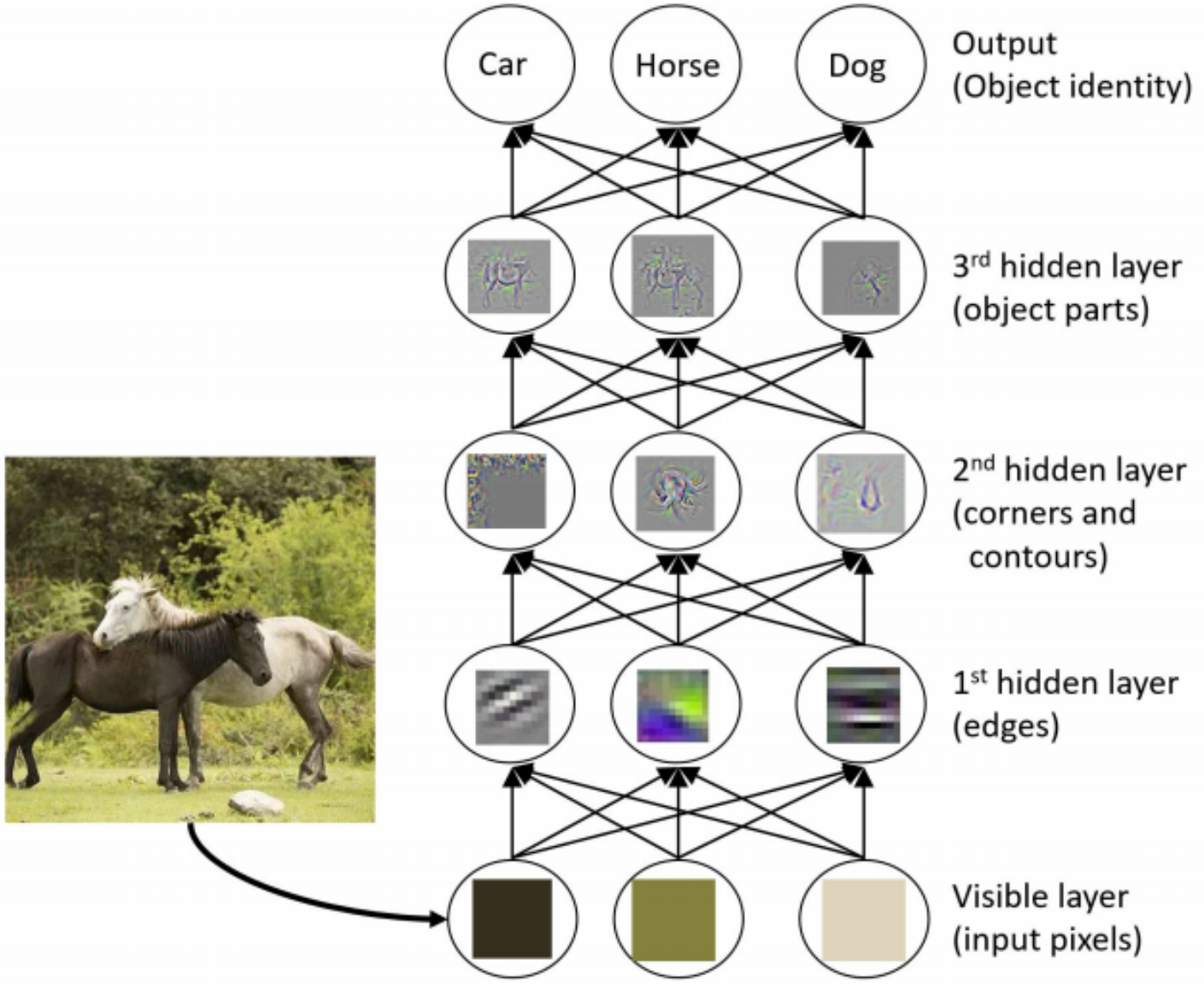
Long ago in the dark ages of computer vision...

Some historical context is helpful in understanding the “Why?” of representation learning (discussed in the paper and reviewed in the next slide). Imagine it is 2010 and you are trying to write an algorithm that recognizes pictures of couches.

- First, you need an algorithm to recognize discrete objects.
- You might get started with a filter that recognizes edges of objects.
- Hopefully, information about edges is good feature engineering for recognizing couches.

This turned out to be difficult and kind of miserable. It was difficult because it turns one into two problems that interact in an opaque way, and it was miserable because it involved lots of manual, somewhat subjective, and often futile pre-work.

An Example in the Paper



Representation Learning

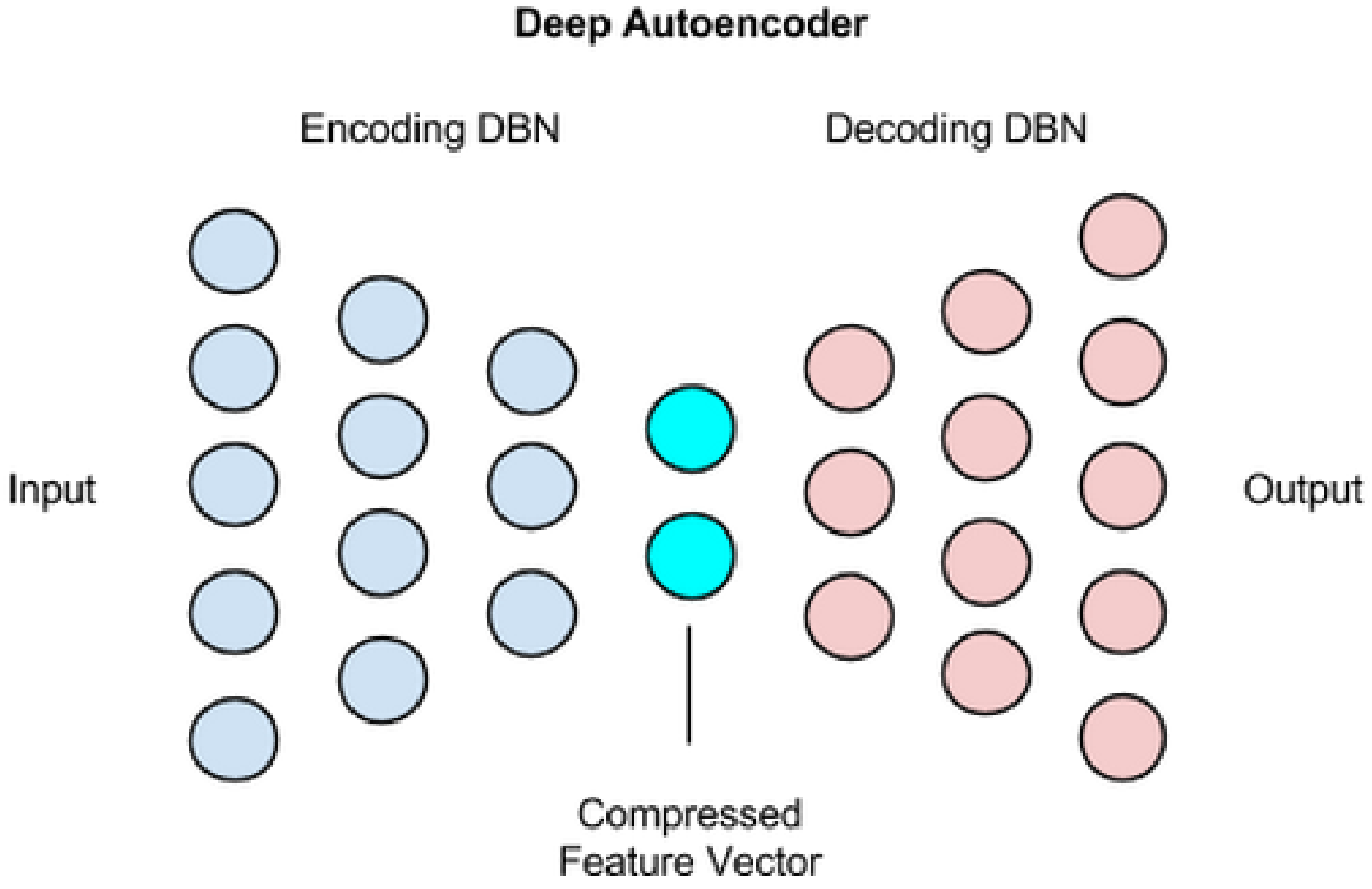
Representation learning, also called feature learning, is where the training algorithm performs (implicit, opaque) feature engineering as part of the training process.

Things called “deep learning” typically apply representation learning concepts via large, many-layered neural networks.

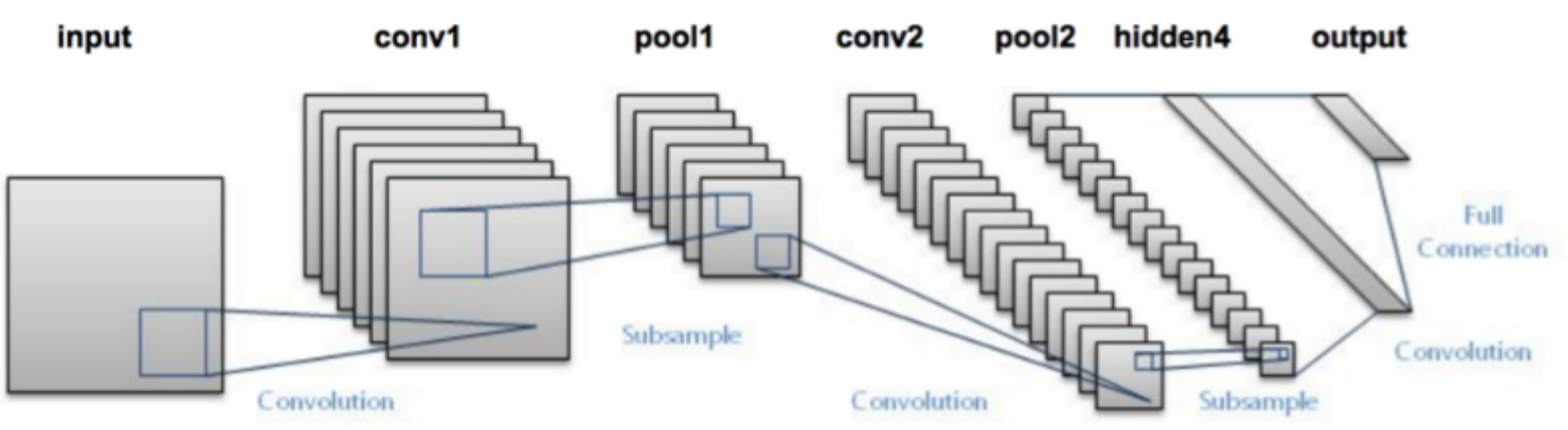
Autoencoders are a very pure example of representation learning because the representation is itself the goal.

Convolutional neural nets are structured so that particular sub-graphs are induced to compute, automatically and typically without any human visibility, features that will have a maximum positive impact on performance.

Autoencoders



Convolutional Neural Nets



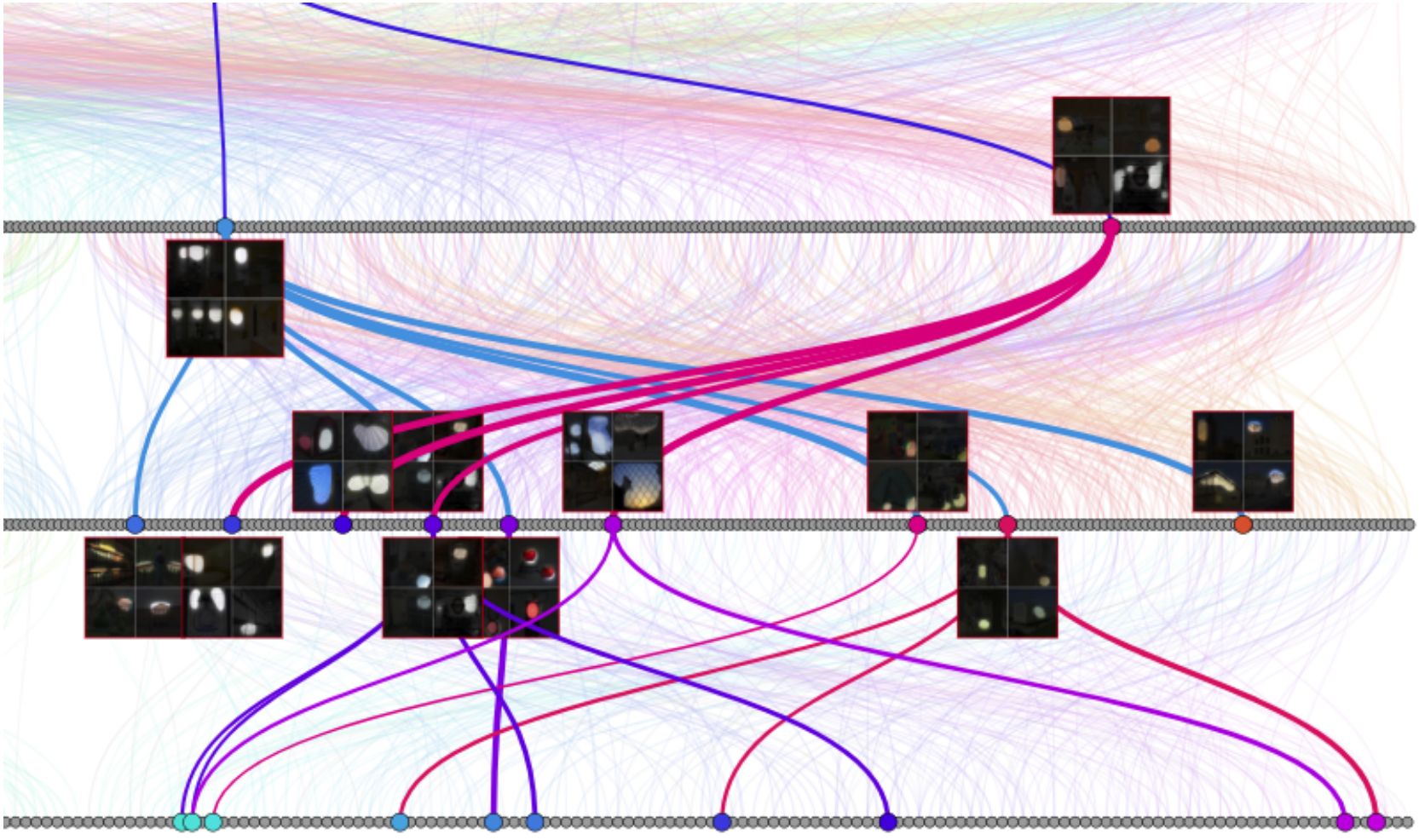
Convolution Filters

The first phase in the preceding diagram was the application of **convolution filters**. Their purpose is to **isolate data from smaller sub-frames** in the image and guarantee a subgraph of the neural net considers them in isolation. This enables representation learning.

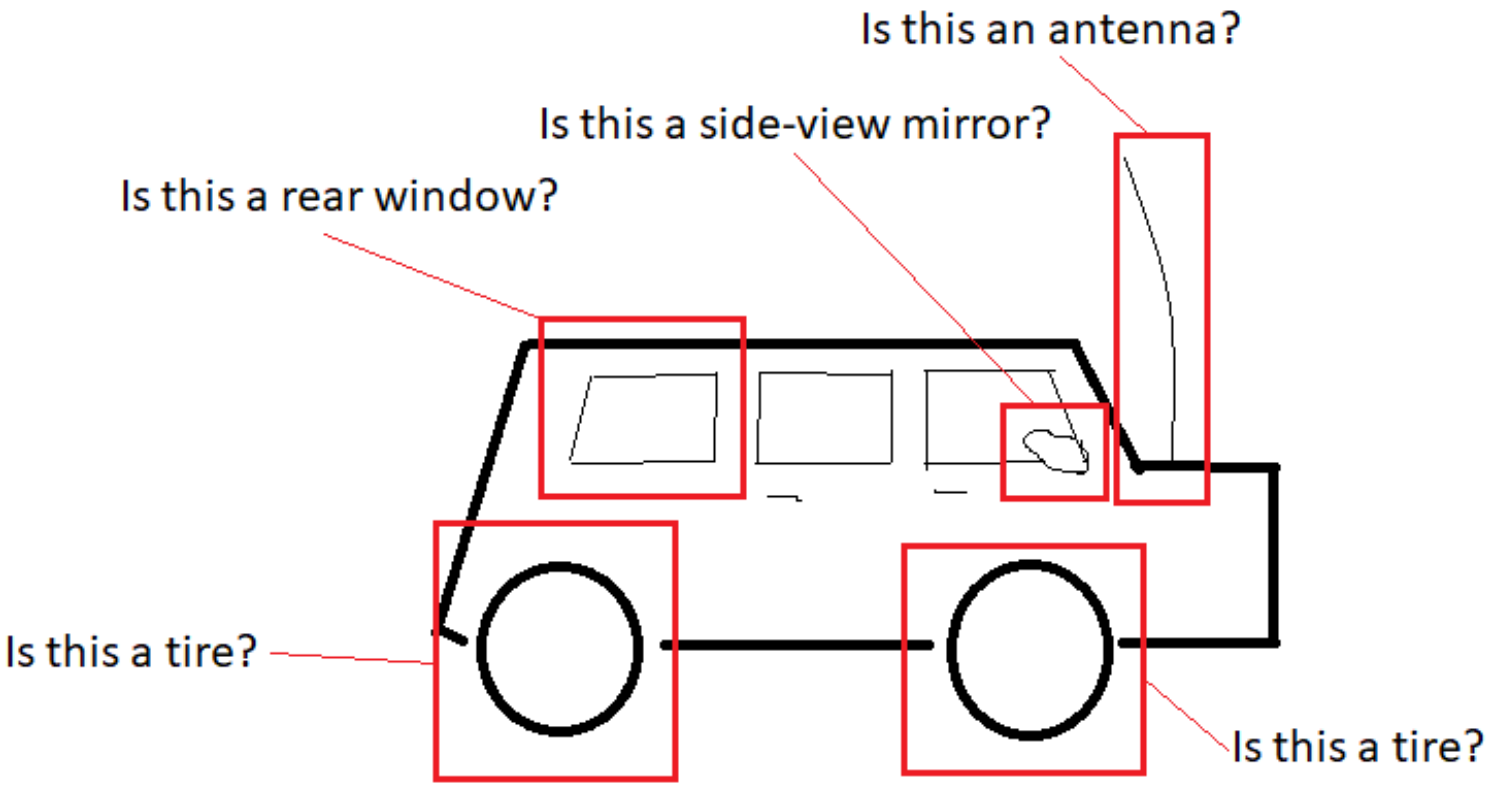
The paper also uses convolution filters in this way in their main examples. As above, they guarantee that **some trees in each layer consider sub-frames in isolation**. Convolution filters are arguably an **independent technique** that enables representation learning and not a representation learning idea themselves.

CNN Visualization

http://people.csail.mit.edu/torralba/...



Simplified Fake Example



Random Forest as Simple Representation Learning

Random forest is a common decision tree ensemble method. It works by training a large number of trees and aggregating their opinions by voting. It has a couple subtle wrinkles...

- Each tree randomly restricts which variables are eligible for splitting at each step.
- Trees are trained deep and hopefully the fact that we have many trees prevents overtraining.

These wrinkles make the individual trees obsessive experts on particular features and you could **think of them as features themselves**. From this perspective, the implicit feature engineering is relatively complex while the application of these features is very simple.

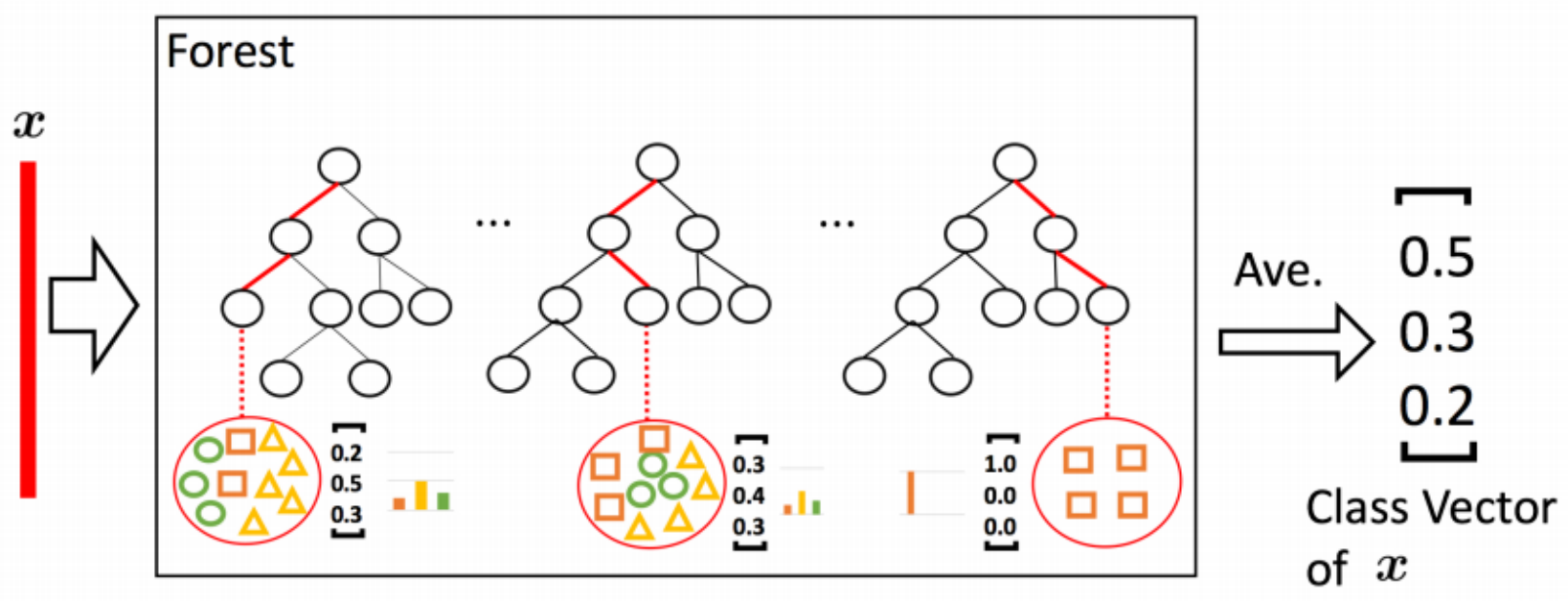
Prereq Concepts for gcForest

The gcForest algorithm uses...

- random forests as described on the previous slide
- **completely random tree forests**, which are composed of very deep decision trees with splits selected totally randomly.
- **class vectors**, which are roughly (and a little dishonestly) vectors of predictions.

These three concepts are used to apply decision trees in layers **with a view towards representation learning** in analogy with convolutional neural nets.

Class Vectors

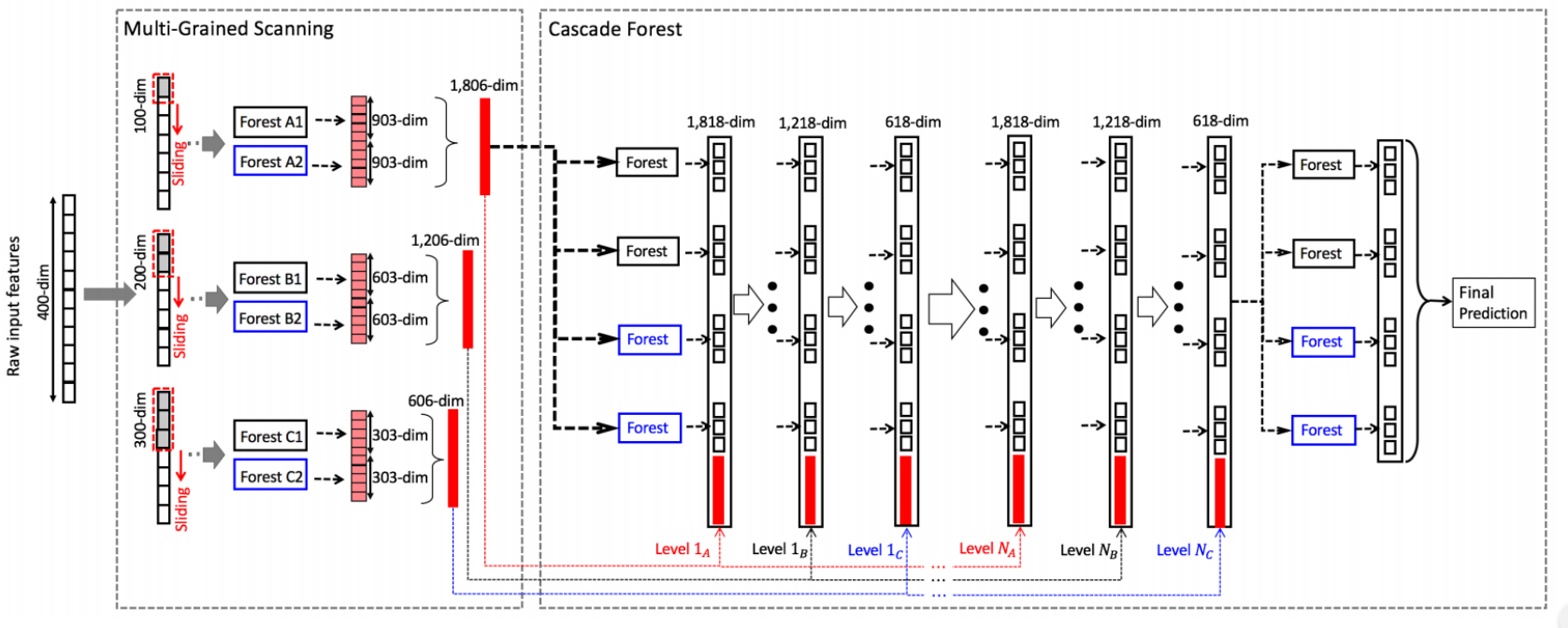


Aerial View on gcForest 1

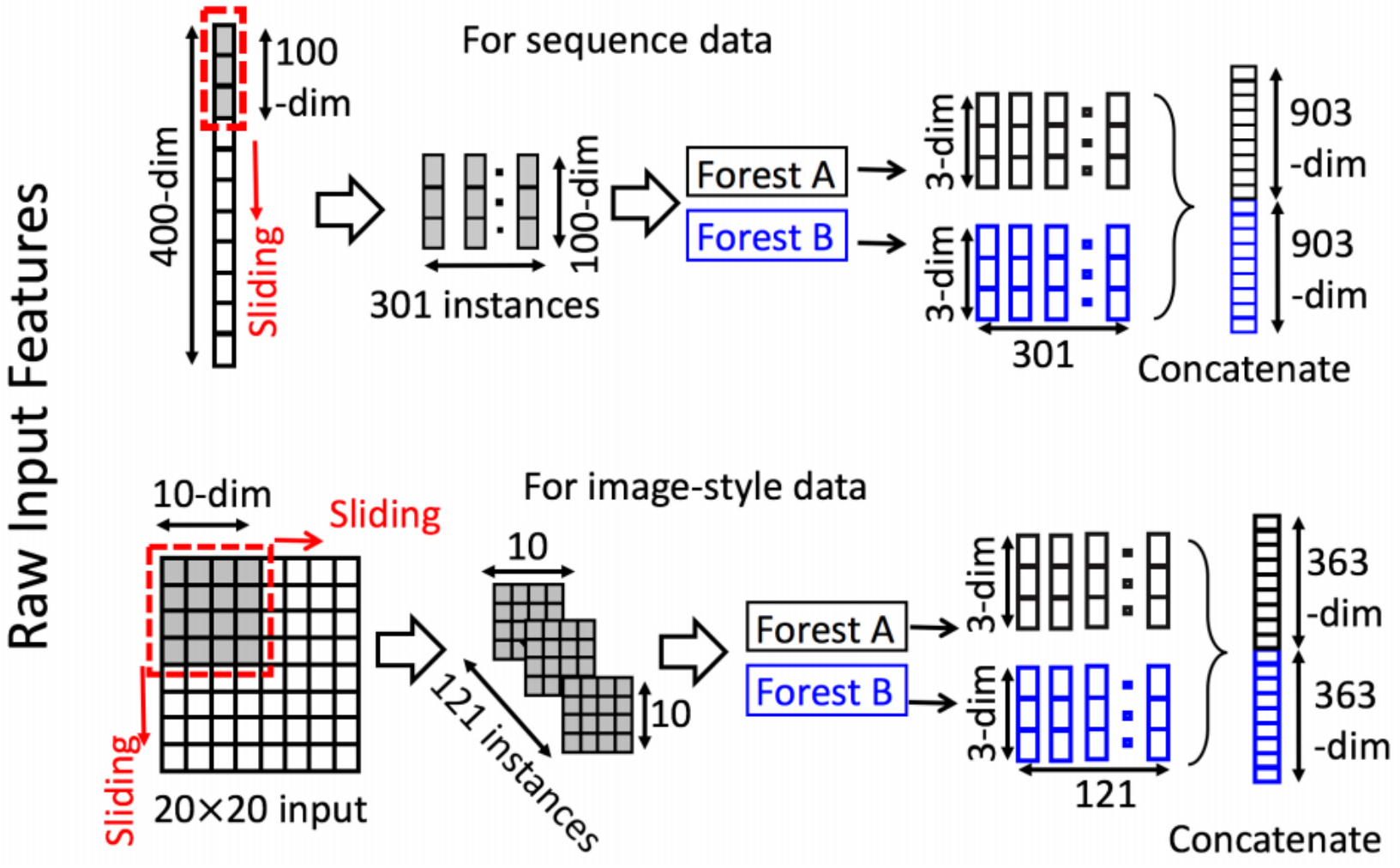
You begin with a vector of features and then...

- a. do whatever filtering (convolution, etc.) you need,
- b. input your modified features to a layer of random forests and completely random tree forests,
- c. extract class vectors and append these to your original vector of modified features,
- d. repeat b-c as needed,
- e. and finally extract honest predictions from the final layer of forests and aggregate by voting.

Aerial View on gcForest 2



Aerial View on gcForest 3



Aerial View on gcForest 4

Why does it work?

As explained previously, we can think of the early-layer forests as **producing new features**, and the **feature selection intrinsic to decision trees** means the features with predictive power are the ones that impact the later layers of forests.

At the final prediction step, we are essentially **running random forest on an augmented feature space** with a number of additional, automatically created features. These **features had their predictive power validated** by a method analogous to their final use - they made it through to the end because they were useful to a random forest.

The End

That's all she wrote.