A brief intro to computer vision and neural networks

28 April 2016

Pamela Toman

At 1:00, you will be able to…

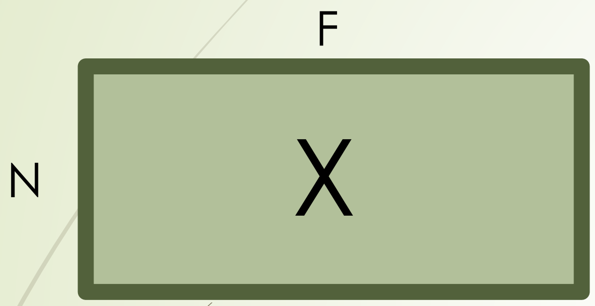
- **Competence** (able to perform on your own, with varying levels of perfection):
  - Understand how neural networks extend linear classification, and have some intuition for how and why they are more powerful
  - Know when a neural network variant might be appropriate for your problem and why
  - Know how to get more help
- **Exposure** (aware):
  - Articulate some of the challenges in computer vision
  - Articulate the broad strokes of gradient descent
  - Recognize the phrase "backpropagation" (it is how we train networks)
  - Recognize the phrase "convolutional neural network" (it's state-of-the-art for vision)
  - Express the history of neural networks and some reasons deep learning has been causing so much excitement in recent years
  - Be familiar with tools that make networks easier to use: transfer learning + software
  - Recall images from a handful of cool recent papers

Let’s chat…

How might we predict a person’s citizenship?

Linear classification

Linear classification

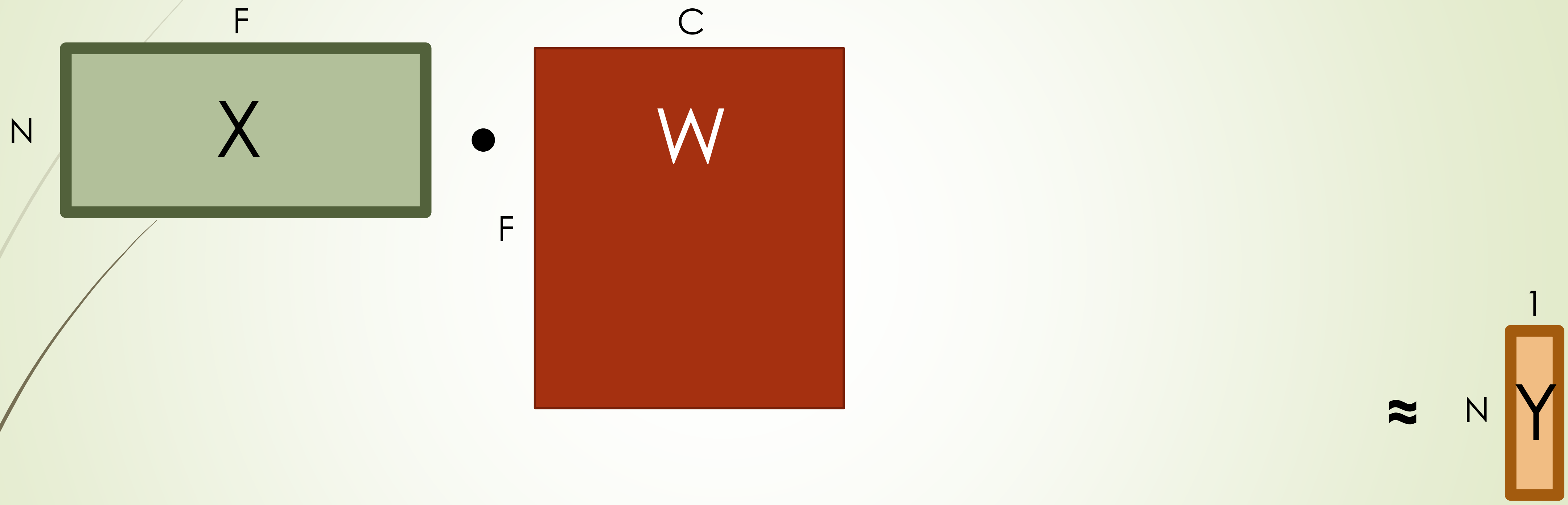


N = number of examples (people: Yeltsin, Swift, Bieber, Obama, you, …)
F = number of features (characteristics: age of first vote, TV watched/year, alcohol consumed/year, family size, years of education, …)

Linear classification

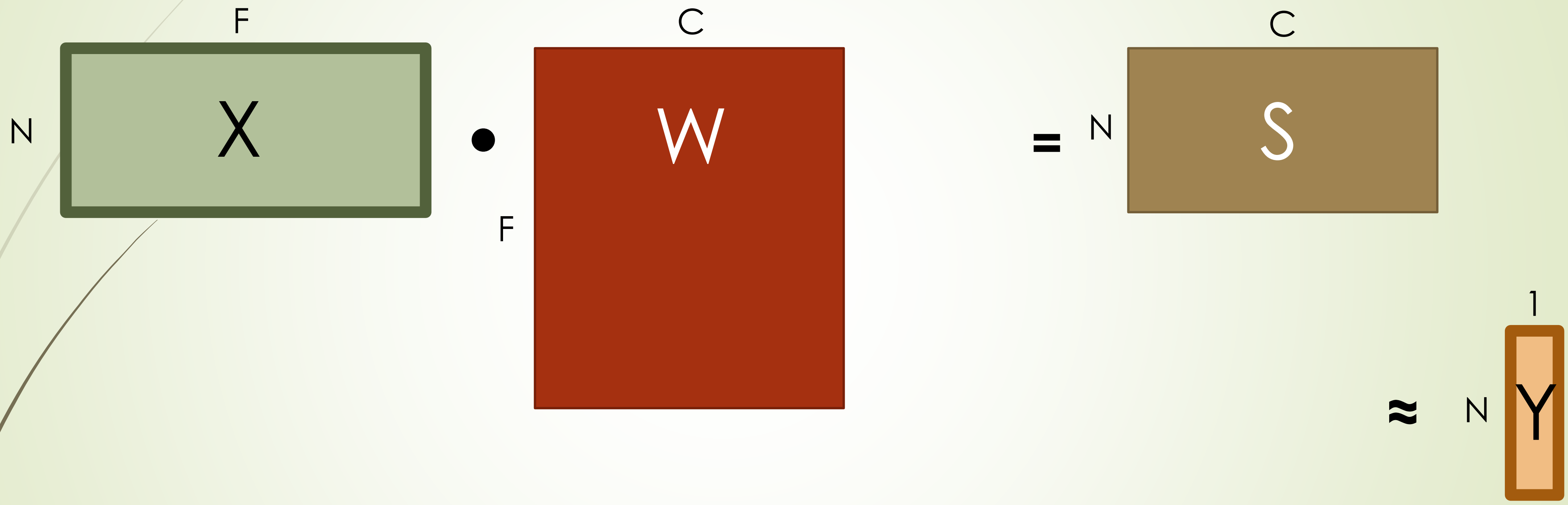
N = number of examples (people: Yeltsin, Swift, Bieber, Obama, you, …)
F = number of features (characteristics: age of first vote, TV watched/year, alcohol consumed/year, family size, years of education, …)

Linear classification



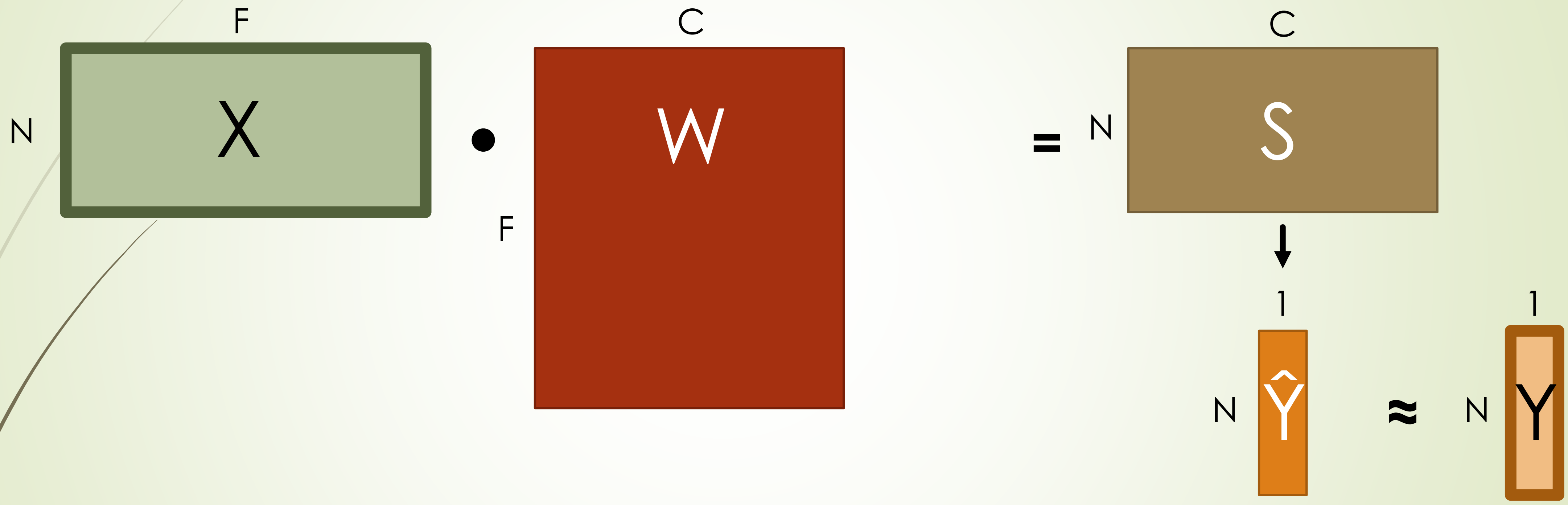
N = number of examples (people: Yeltsin, Swift, Bieber, Obama, you, …)
F = number of features (characteristics: age of first vote, TV watched/year, alcohol consumed/year, family size, years of education, …)
C = number of classes (countries: Russia, Canada, USA, China, …)

Linear classification



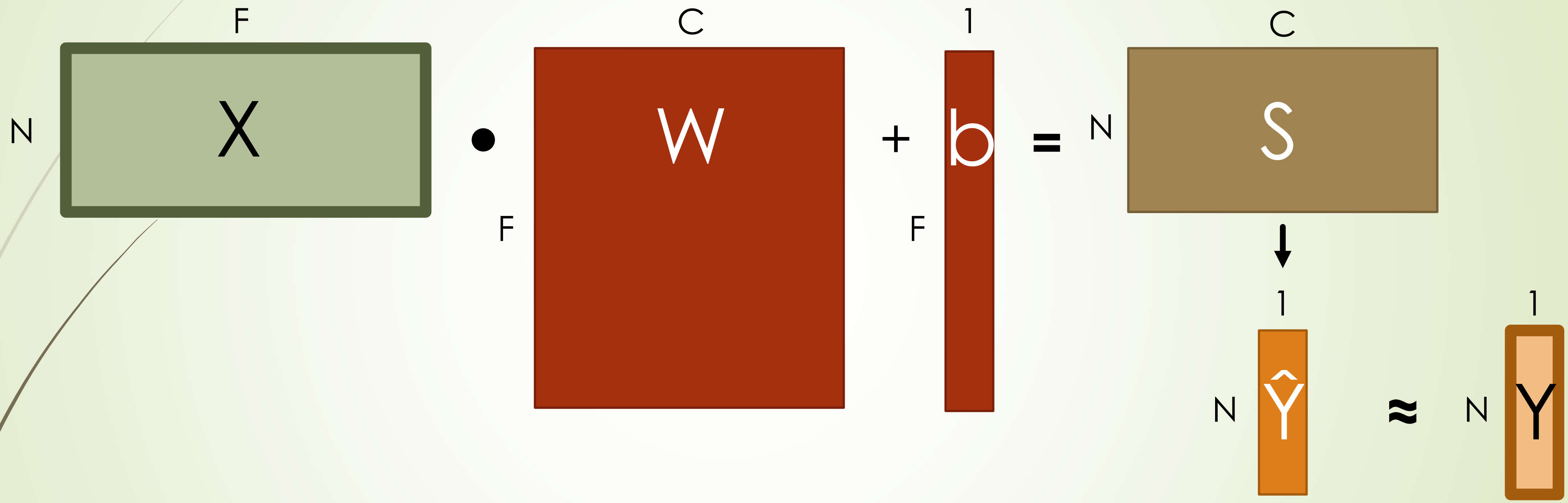
N = number of examples (people: Yeltsin, Swift, Bieber, Obama, you, ...)
F = number of features (characteristics: age of first vote, TV watched/year, alcohol consumed/year, family size, years of education, ...)
C = number of classes (countries: Russia, Canada, USA, China, ...)

Linear classification



N = number of examples (people: Yeltsin, Swift, Bieber, Obama, you, …)
F = number of features (characteristics: age of first vote, TV watched/year, alcohol consumed/year, family size, years of education, …)
C = number of classes (countries: Russia, Canada, USA, China, …)

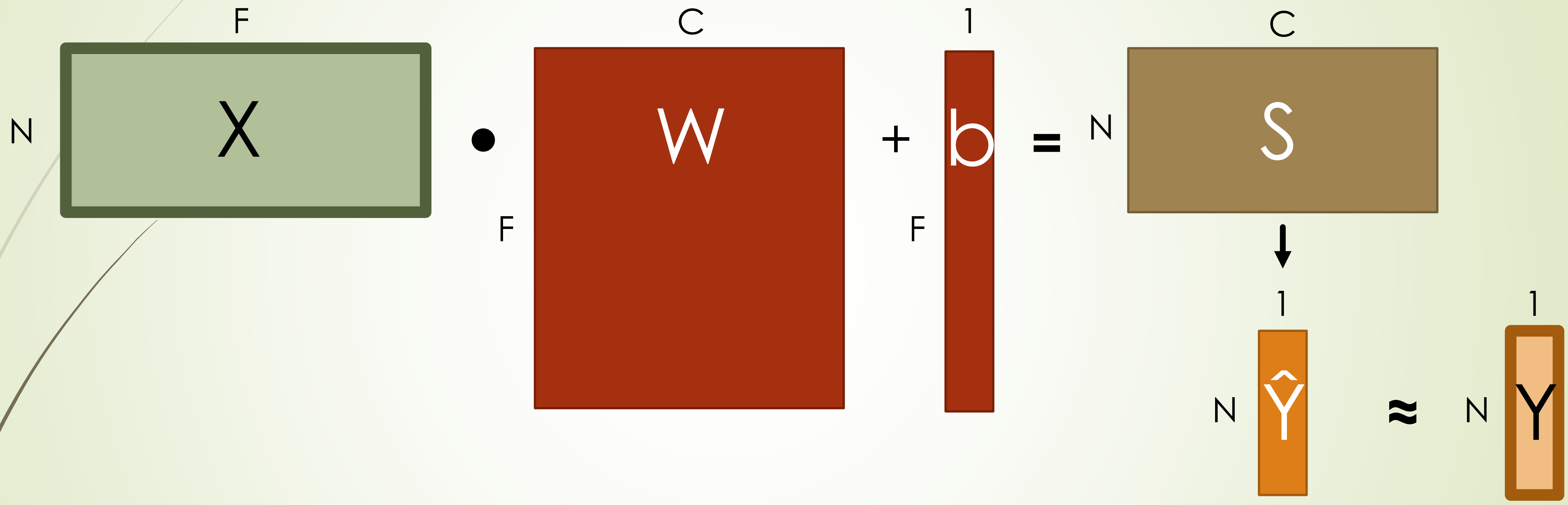
Linear classification



N = number of examples (people: Yeltsin, Swift, Bieber, Obama, you, …)
F = number of features (characteristics: age of first vote, TV watched/year, alcohol consumed/year, family size, years of education, …)
C = number of classes (countries: Russia, Canada, USA, China, …)

$$f(X;W,b) = XW + b$$

Linear classification



N = number of examples (people: Yeltsin, Swift, Bieber, Obama, you, …)
F = number of features (characteristics: age of first vote, TV watched/year, alcohol consumed/year, family size, years of education, …)
C = number of classes (countries: Russia, Canada, USA, China, …)

Now a challenge…

Now a challenge…

Now a challenge…










Now a challenge...







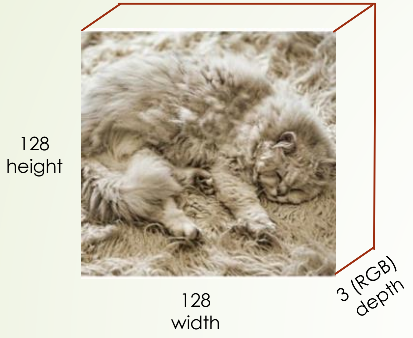


How do we label images?

Images as input examples

Images as input examples

X



Y

"cat"

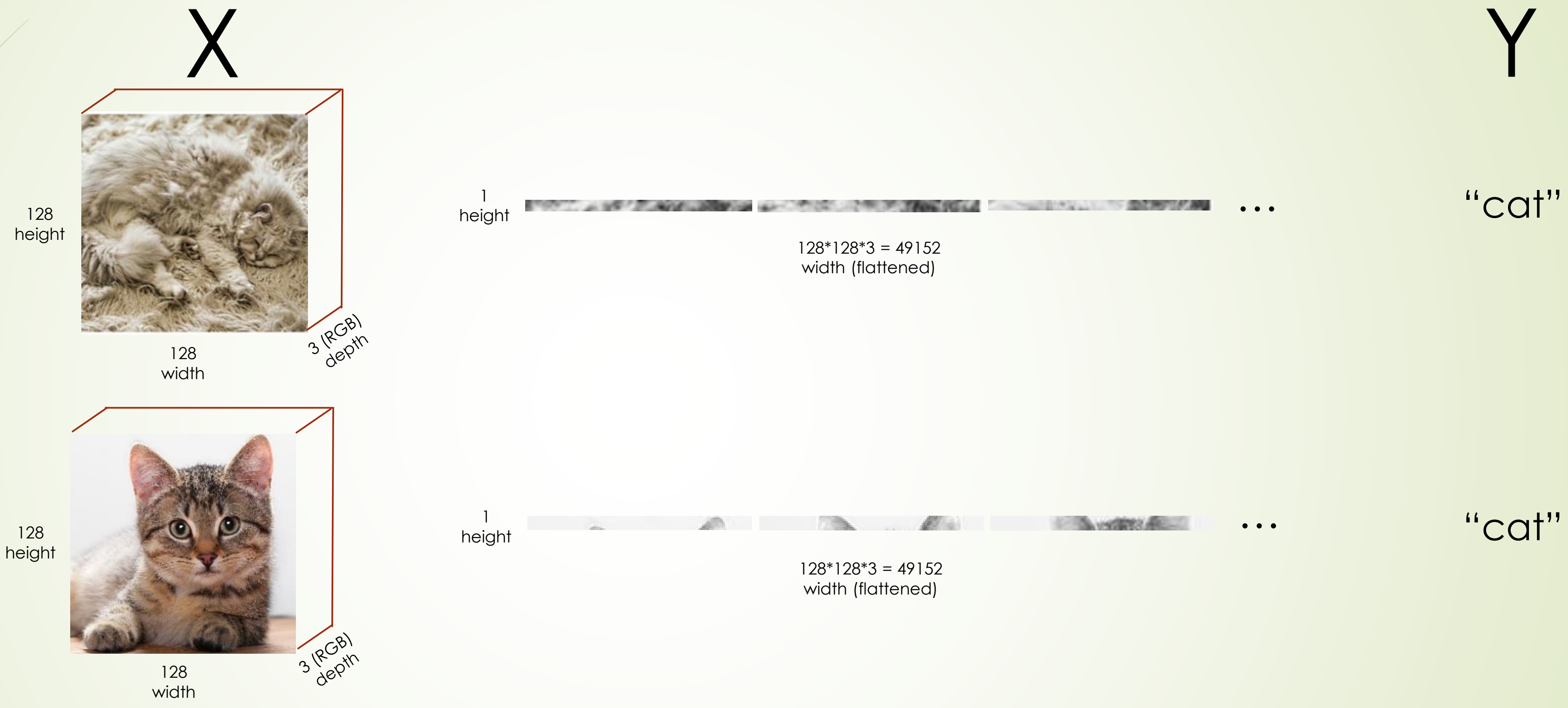
Images as input examples

X

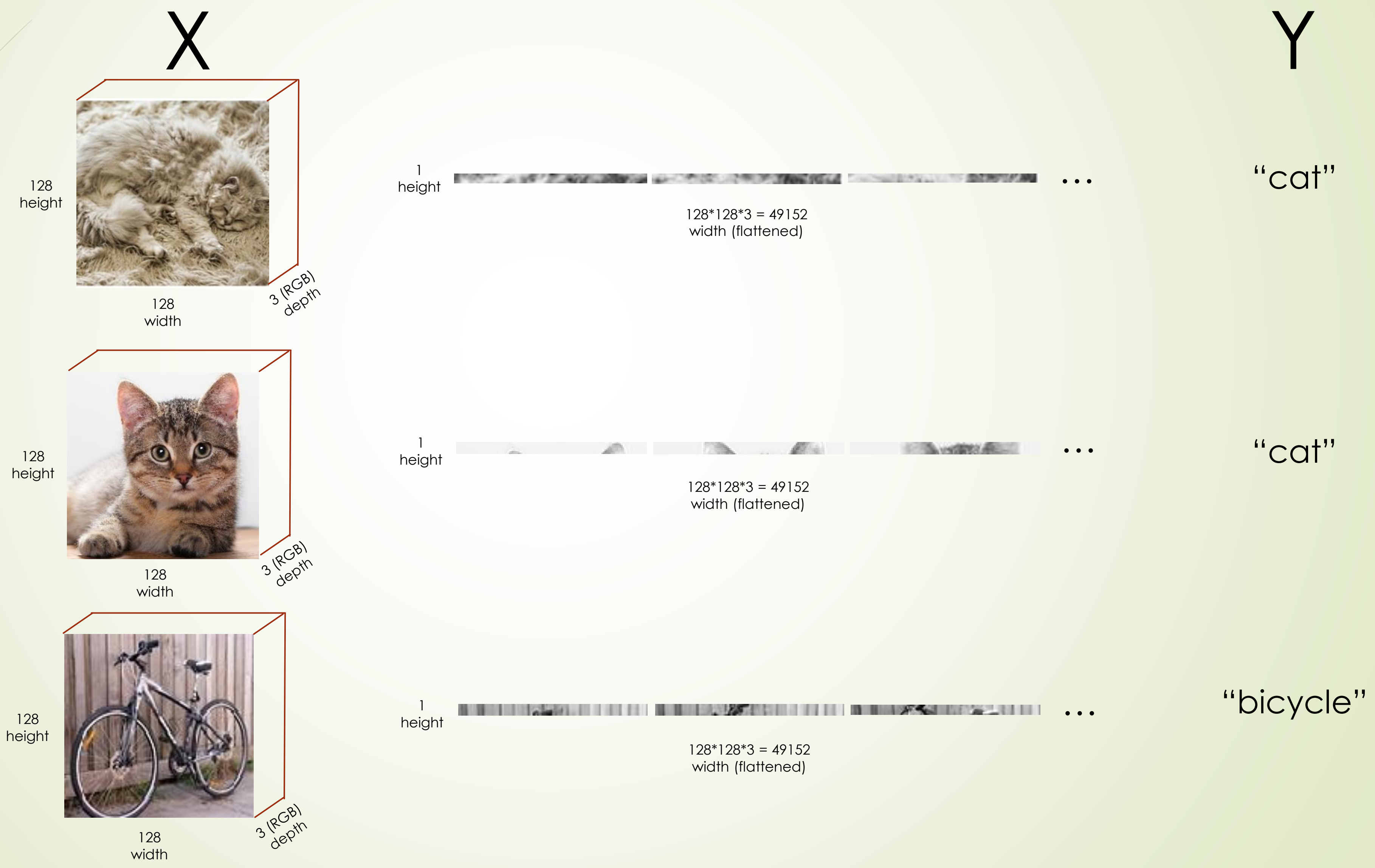
Y

“cat”

Images as input examples

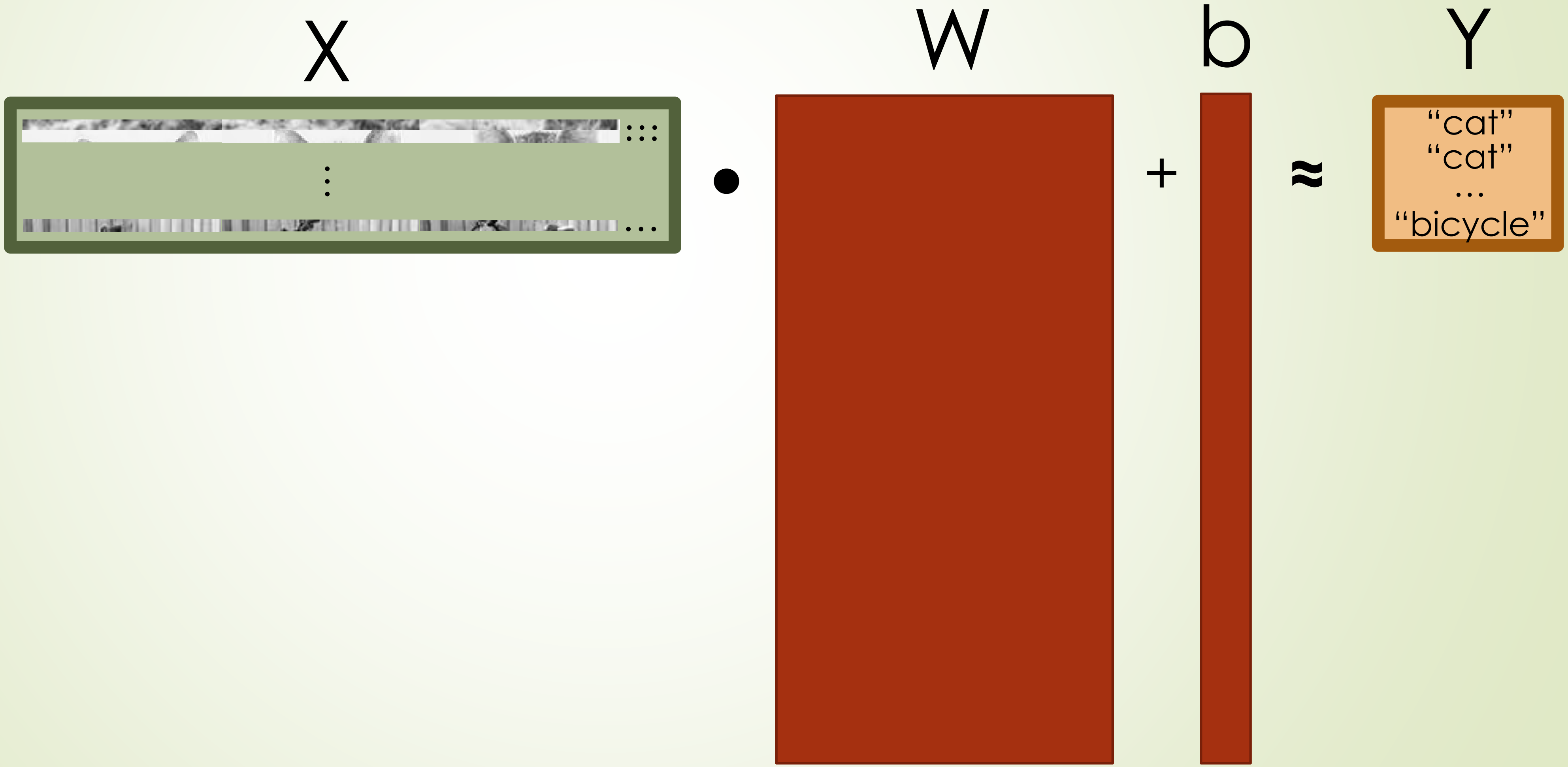


Images as input examples



Images as input examples

How do we figure out the weights *W* and offsets *b*?



Gradient descent to the rescue!

- Define a "loss function"
- Guess at all the weights *W* and bias offsets *b*
- Seek iterative improvement

Gradient descent to the rescue!

- Define a "loss function"
- Guess at all the weights *W* and bias offsets *b*
- Seek iterative improvement

Gradient descent to the rescue!

- Define a “loss function”
- Guess at all the weights *W* and bias offsets *b*
- Seek iterative improvement

Cargo Collective

Gradient descent to the rescue!

- Define a "loss function"
- Guess at all the weights *W* and bias offsets *b*
- Seek iterative improvement

Cargo Collective

Gradient descent to the rescue!

- Define a "loss function"
- Guess at all the weights W and bias offsets b
- Seek iterative improvement

1. Apply f to (X, W, b) to produce the current loss
2. For each parameter (value in W or b),
   calculate how the loss responds to change in that param
   (this is the gradient ≈ derivative)
3. Update each param with a tiny step *away* from higher loss
4. Repeat

Gradient descent to the rescue!

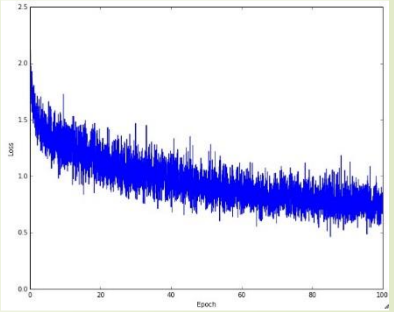
- Define a "loss function"
- Guess at all the weights *W* and bias offsets *b*
- Seek iterative improvement

1. Apply *f* to *(X, W, b)* to produce the current loss
2. For each parameter (value in *W* or *b*), calculate how the loss responds to change in that param (this is the gradient ≈ derivative)
3. Update each param with a tiny step *away* from higher loss
4. Repeat

Most efficient to use calculus

Gradient descent to the rescue!

- Define a “loss function”
- Guess at all the weights *W* and bias offsets *b*
- Seek iterative improvement



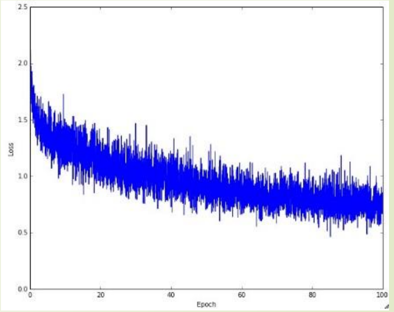
1. Apply *ƒ* to *(X, W, b)* to produce the current loss
2. For each parameter (value in *W* or *b*),
   calculate how the loss responds to change in that param
   (this is the gradient ≈ derivative)
3. Update each param with a tiny step *away* from higher loss
4. Repeat

Most efficient to use calculus

Gradient descent to the rescue!

* Not actually the best method for convex problems like linear regression

- Define a "loss function"
- Guess at all the weights *W* and bias offsets *b*
- Seek iterative improvement



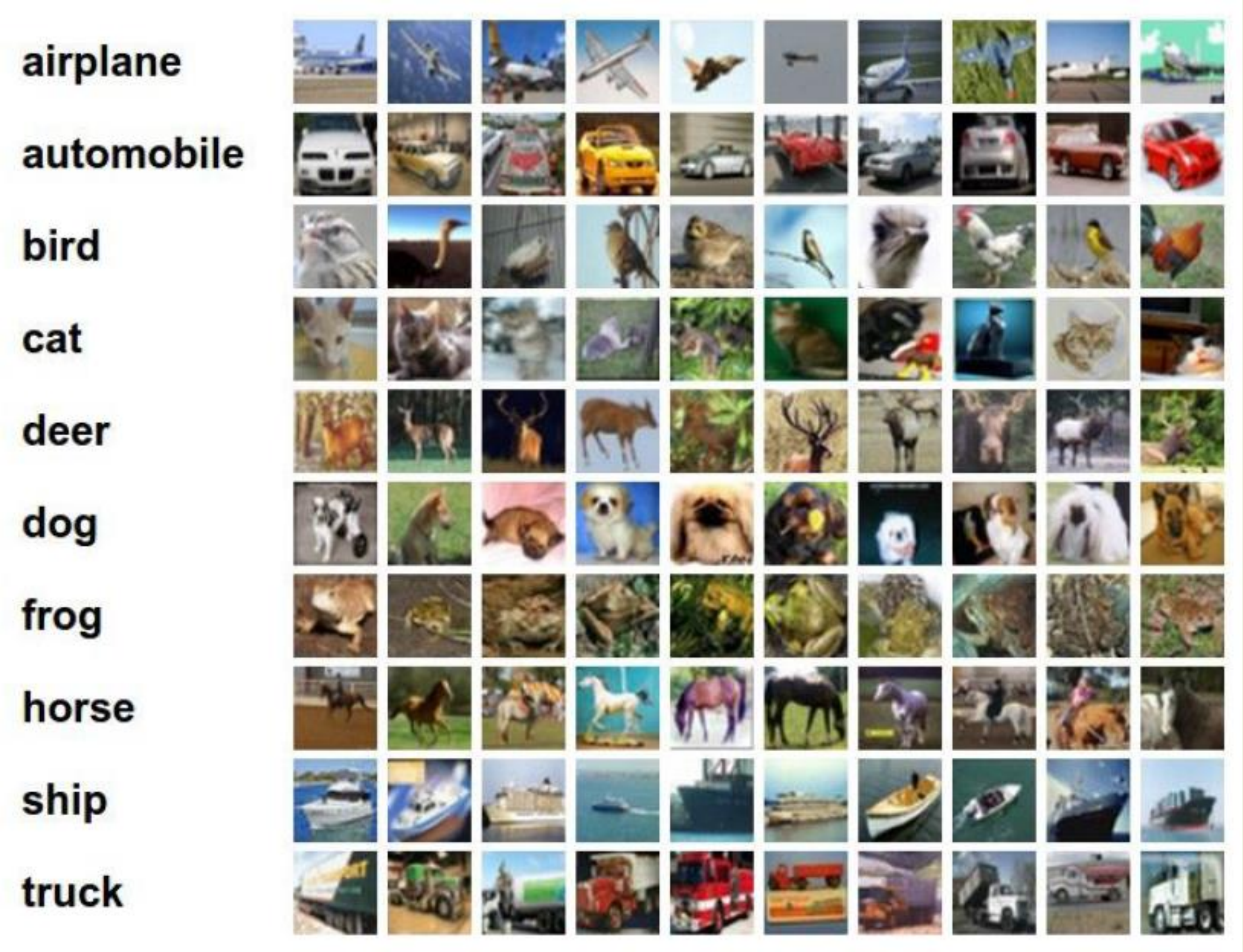
1. Apply f to *(X, W, b)* to produce the current loss
2. For each parameter (value in *W* or *b*),
   calculate how the loss responds to change in that param
   (this is the gradient ≈ derivative)
3. Update each param with a tiny step *away* from higher loss
4. Repeat

Most efficient to use calculus

Example:

Sample dataset

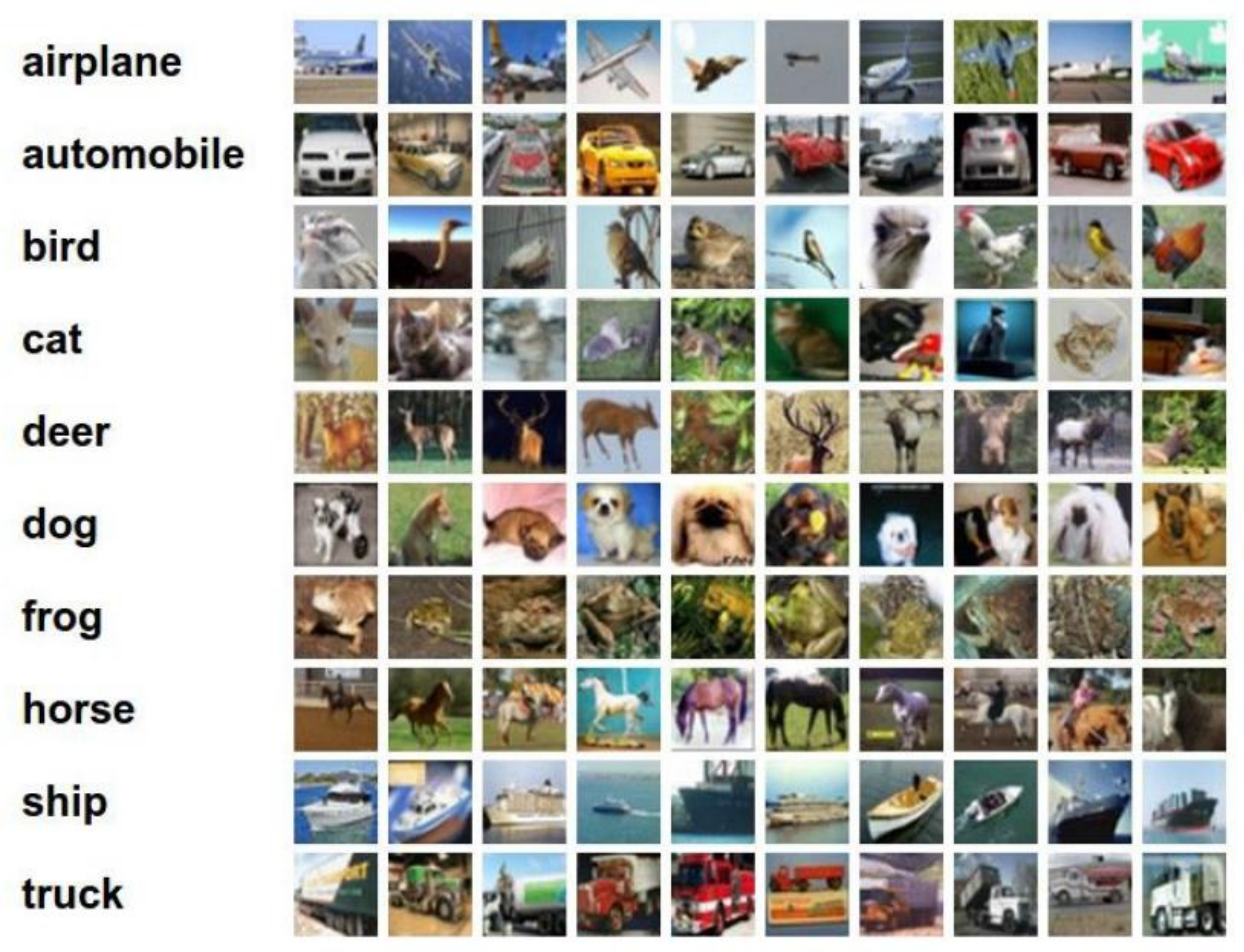
Data:



Example:

Sample dataset
& results of training

Data:



Learned weights (templates):

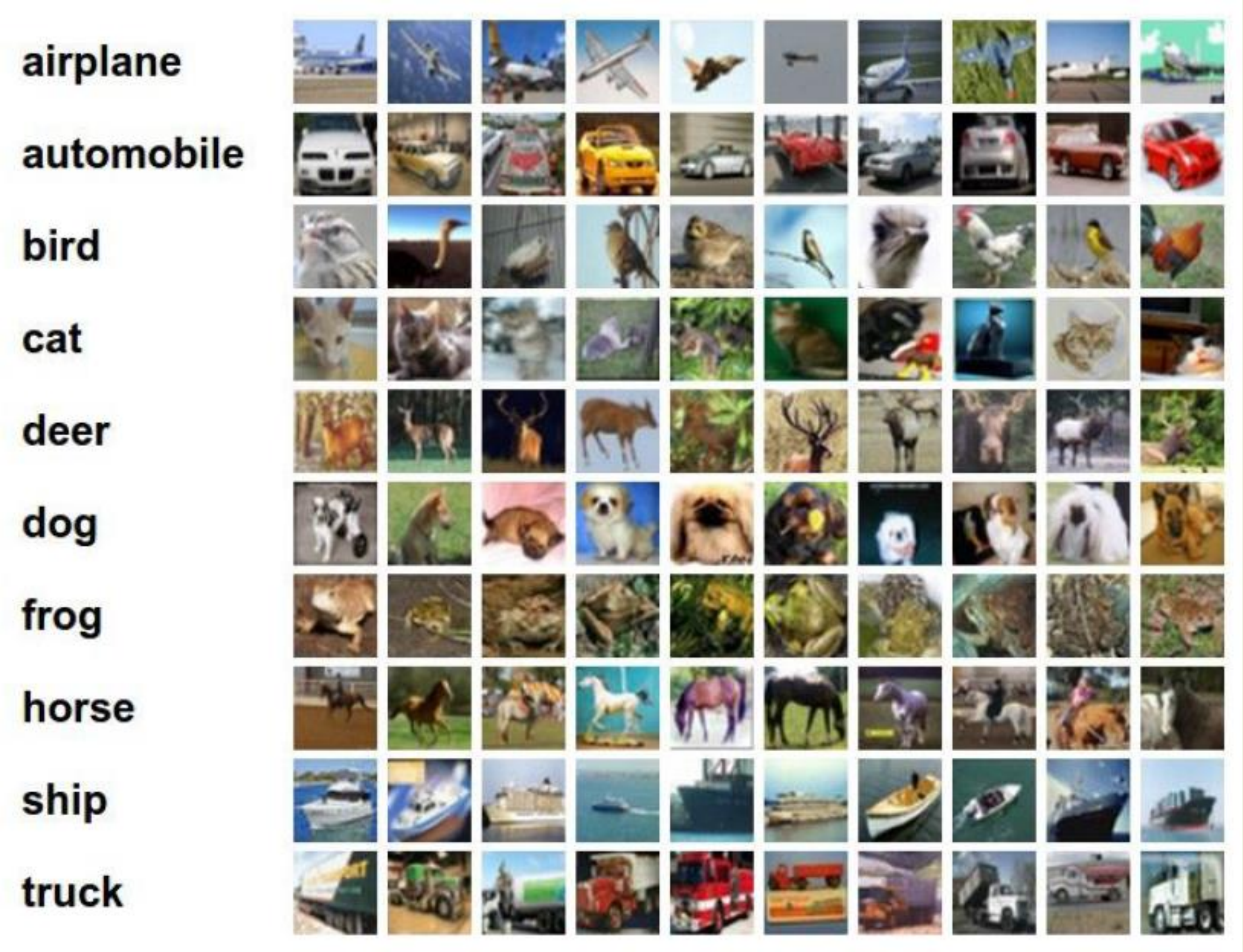
Stanford CS231N

Example:

Sample dataset
& results of training

Achieves ~40% accuracy

Data:



Learned weights (templates):

Stanford CS231N

Neural networks let us do better

- More expressive power than linear classification in two major ways:
  - Hierarchical decisions: composition into deeper (not "deep") networks
  - Non-linear relationships: threshold at 0 with ReLUs (rectified linear units)
- Trade-off: estimating more parameters requires more data, memory, time

Neural networks let us do better

- More expressive power than linear classification in two major ways:
  - Hierarchical decisions: composition into deeper (not "deep") networks
  - Non-linear relationships: threshold at 0 with ReLUs (rectified linear units)
- Trade-off: estimating more parameters requires more data, memory, time

inputs $\longrightarrow$ $\text{weights}_{\text{classes}}$ $\longrightarrow$ classes

Neural networks let us do better

- More expressive power than linear classification in two major ways:
  - Hierarchical decisions: composition into deeper (not "deep") networks
  - Non-linear relationships: threshold at 0 with ReLUs (rectified linear units)
- Trade-off: estimating more parameters requires more data, memory, time

inputs → weights$_{\text{classes}}$ → classes

inputs → weights$_{\text{hidden_1}}$ → threshold at 0
→ weights$_{\text{hidden_2}}$ → threshold at 0
→ weights$_{\text{hidden_3}}$ → threshold at 0 → weights$_{\text{classes}}$ → classes

$f(X; \ldots) = g \circ h \circ i \circ j \circ \cdots$

Neural networks let us do better

- More expressive power than linear classification in two major ways:
  - Hierarchical decisions: composition into deeper (not "deep") networks
  - Non-linear relationships: threshold at 0 with ReLUs (rectified linear units)
- Trade-off: estimating more parameters requires more data, memory, time

inputs → $\text{weights}_{\text{classes}}$ → classes

inputs → $\text{weights}_{\text{hidden\_1}}$ → threshold at 0
→ $\text{weights}_{\text{hidden\_2}}$ → threshold at 0
→ $\text{weights}_{\text{hidden\_3}}$ → threshold at 0 → $\text{weights}_{\text{classes}}$ → classes

$$f(X; \ldots) = g \circ h \circ i \circ j \circ \cdots$$

Neural networks let us do better

- More expressive power than linear classification in two major ways:
  - Hierarchical decisions: composition into deeper (not "deep") networks
  - Non-linear relationships: threshold at 0 with ReLUs (rectified linear units)
- Trade-off: estimating more parameters requires more data, memory, time

inputs → $\text{weights}_{\text{classes}}$ → classes

inputs → $\text{weights}_{\text{hidden\_1}}$ → threshold at 0 →
$\text{weights}_{\text{hidden\_2}}$ → threshold at 0 →
$\text{weights}_{\text{hidden\_3}}$ → threshold at 0 → $\text{weights}_{\text{classes}}$ → classes

To estimate the weights, we use ***backpropagation.*** *It simplifies the loss gradient calculation: we can train through a series of highly local, chain-rule based transformations. Backprop is clever and neat, but too mathy for this talk.*

$$f(X; \ldots) = g \circ h \circ i \circ j \circ \cdots$$

Neural networks let us do better

- More expressive power than linear classification in two major ways:
  - Hierarchical decisions: composition into deeper (not "deep") networks
  - Non-linear relationships: threshold at 0 with ReLUs (rectified linear units)
- Trade-off: estimating more parameters requires more data, memory, time

inputs → $\text{weights}_{\text{classes}}$ → classes

inputs → $\text{weights}_{\text{hidden\_1}}$ → threshold at 0

→ $\text{weights}_{\text{hidden\_2}}$ → threshold at 0

→ $\text{weights}_{\text{hidden\_3}}$ → threshold at 0 → $\text{weights}_{\text{classes}}$ → classes

Achieves ~55% accuracy

To estimate the weights, we use **backpropagation.** *It simplifies the loss gradient calculation: we can train through a series of highly local, chain-rule based transformations. Backprop is clever and neat, but too mathy for this talk.*

Convolutional neural networks let us do even better

Convolutional neural networks let us do even better

- By assuming image data, we get more powerful models for the same cost

Convolutional neural networks let us do even better

- By assuming image data, we get more powerful models for the same cost
- Instead of flattening each training image, we keep its original 3-D form

Convolutional neural networks let us do even better

- By assuming image data, we get more powerful models for the same cost

- Instead of flattening each training image, we keep its original 3-D form
- Instead of 1 huge weight matrix at each layer, instead we have sets of 3-D *filters*

Convolutional neural networks let us do even better

- By assuming image data, we get more powerful models for the same cost
- Instead of flattening each training image, we keep its original 3-D form
- Instead of 1 huge weight matrix at each layer, instead we have sets of 3-D *filters*
  - Each filter looks at a small spatial region (e.g., 3x3 pixels) and the entire input depth (e.g., 3 RGB channels at the top layer)

Convolutional neural networks let us do even better

- By assuming image data, we get more powerful models for the same cost

- Instead of flattening each training image, we keep its original 3-D form
- Instead of 1 huge weight matrix at each layer, instead we have sets of 3-D *filters*
  - Each filter looks at a small spatial region (e.g., 3x3 pixels) and the entire input depth (e.g., 3 RGB channels at the top layer)
  - We "convolve" (slide) each filter across the entire image in discrete jumps
  - At each place we pause, we calculate a single "activation" value

Convolutional neural networks let us do even better

- By assuming image data, we get more powerful models for the same cost

- Instead of flattening each training image, we keep its original 3-D form
- Instead of 1 huge weight matrix at each layer, instead we have sets of 3-D *filters*
  - Each filter looks at a small spatial region (e.g., 3x3 pixels) and the entire input depth (e.g., 3 RGB channels at the top layer)
  - We “convolve” (slide) each filter across the entire image in discrete jumps
  - At each place we pause, we calculate a single “activation” value
  - We compile the activation values into the output of the layer

Convolutional neural networks let us do even better

- By assuming image data, we get more powerful models for the same cost
- Instead of flattening each training image, we keep its original 3-D form
- Instead of 1 huge weight matrix at each layer, instead we have sets of 3-D *filters*
  - Each filter looks at a small spatial region (e.g., 3x3 pixels) and the entire input depth (e.g., 3 RGB channels at the top layer)
  - We "convolve" (slide) each filter across the entire image in discrete jumps
  - At each place we pause, we calculate a single "activation" value
  - We compile the activation values into the output of the layer
- Training learns the weights for each filter that best transform input pixels into class scores

Convolutional neural networks let us do even better

- By assuming image data, we get more powerful models for the same cost

- Instead of flattening each training image, we keep its original 3-D form
- Instead of 1 huge weight matrix at each layer, instead we have sets of 3-D *filters*
  - Each filter looks at a small spatial region (e.g., 3x3 pixels) and the entire input depth (e.g., 3 RGB channels at the top layer)
  - We "convolve" (slide) each filter across the entire image in discrete jumps
  - At each place we pause, we calculate a single "activation" value
  - We compile the activation values into the output of the layer
- Training learns the weights for each filter that best transform input pixels into class scores

Diagrams of tensors are hard
(blame physics!)

Convolutional neural networks let us do even better

- By assuming image data, we get more powerful models for the same cost

- Instead of flattening each training image, we keep its original 3-D form
- Instead of 1 huge weight matrix at each layer, instead we have sets of 3-D *filters*
  - Each filter looks at a small spatial region (e.g., 3x3 pixels) and the entire input depth (e.g., 3 RGB channels at the top layer)
  - We "convolve" (slide) each filter across the entire image in discrete jumps
  - At each place we pause, we calculate a single "activation" value
  - We compile the activation values into the output of the layer
- Training learns the weights for each filter that best transform input pixels into class scores

Diagrams of tensors are hard (blame physics!)

Achieves ~80% accuracy

$$f(X; \ldots) = g' \circ h' \circ i' \circ j' \circ \cdots$$

Convolutional neural networks let us do even better

- By assuming image data, we get more powerful models for the same cost
- Instead of flattening each training image, we keep its original 3-D form
- Instead of 1 huge weight matrix at each layer, instead we have sets of 3-D *filters*
  - Each filter looks at a small spatial region (e.g., 3x3 pixels) and the entire input depth (e.g., 3 RGB channels at the top layer)
  - We "convolve" (slide) each filter across the entire image in discrete jumps
  - At each place we pause, we calculate a single "activation" value
  - We compile the activation values into the output of the layer
- Training learns the weights for each filter that best transform input pixels into class scores

Diagrams of tensors are hard (blame physics!)

Achieves ~80% accuracy

Why do convolutional neural networks perform better?

- Performs even better than vanilla neural networks:
  - Non-linearities provide more freedom to the learning algorithm
  - Deeper compositions provide hierarchical recognition

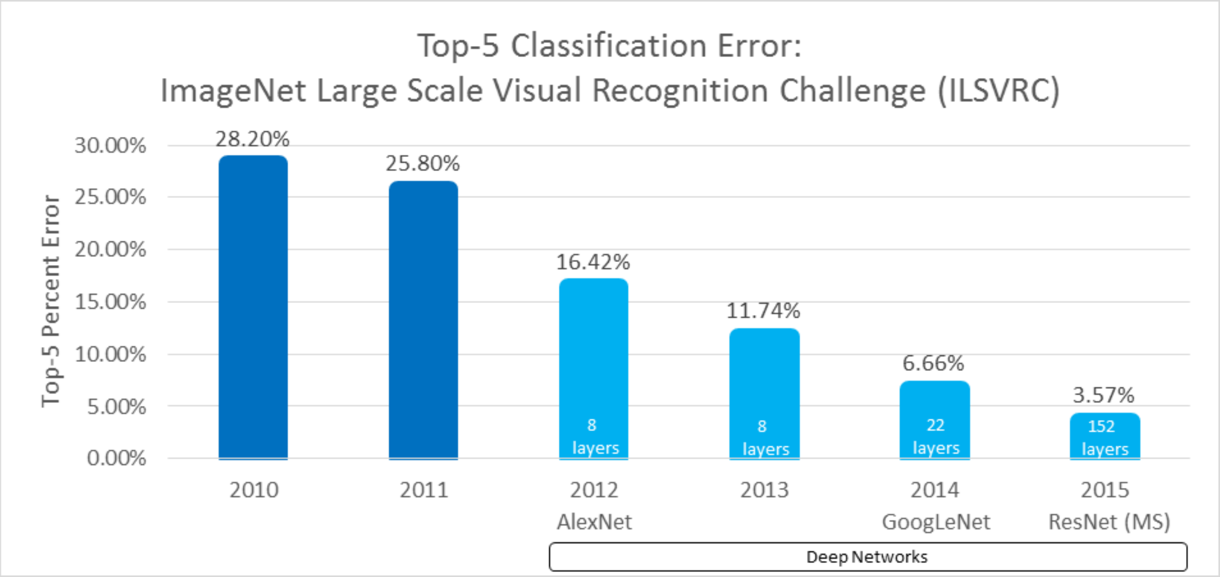
    -- “Person” has “face” has “eye” has “roundness”

    -- “Cat walking right” and “cat facing camera” can be combined into “cat”
  - “Sliding” mean units are re-usable: faster to train, more robust to transformations
  - In practice, we use many other tricks too (beyond composing convolutional layers and ReLUs)
- Still have performance trade-offs in data, memory, time

“Deepness” of learning now matters:
deeper network ⇒ better performance

A foray into history & ImageNet

- Early development in '60s; some interest in '80s
- Mostly scoffed at 'til ~2010… and then it changed
  - Previously only okay performance; neurologists don't like the parallel
  - Recent huge success in image, speech, text recognition
- Newfound success widely attributed to: (a) increased data, (b) increased processing power, (c) training improvements (e.g., ReLU/thresholding at 0)

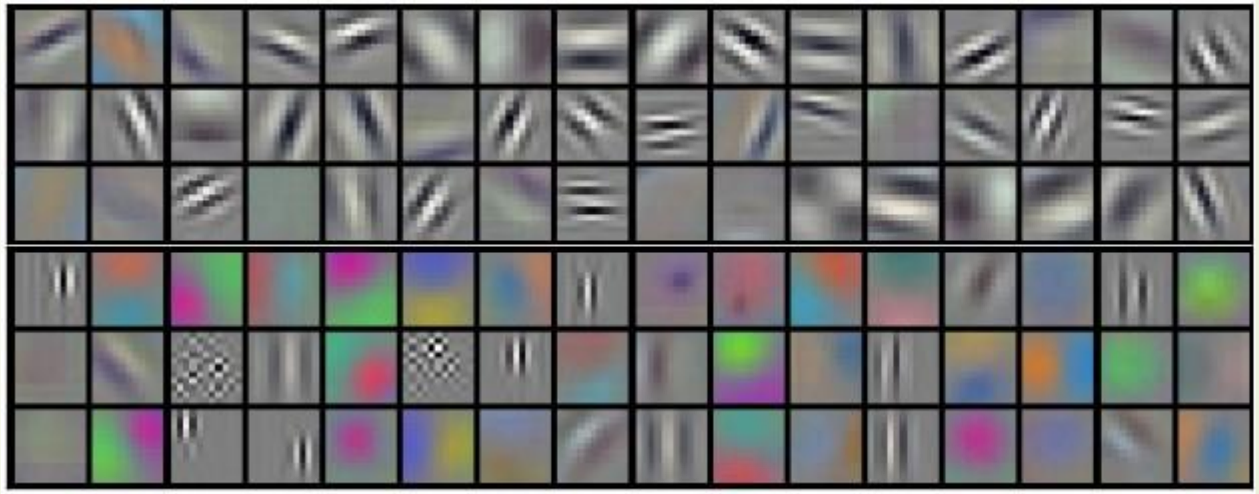


ILSVRC:
1 million images,
1000 categories

Human: 5.1% (Karpathy 2014)

Convolutional neural networks learn useful sub-image features

- Filters behave like learned/derived features (auto-derived visual analogues to “age of first vote”, “TV watched/year”, “family size”, …)



Example first-layer filters learned by Krizhevsky et al. 2012 (11x11x3).

Many first-tier computer vision features have this form (“Gabor-like”).

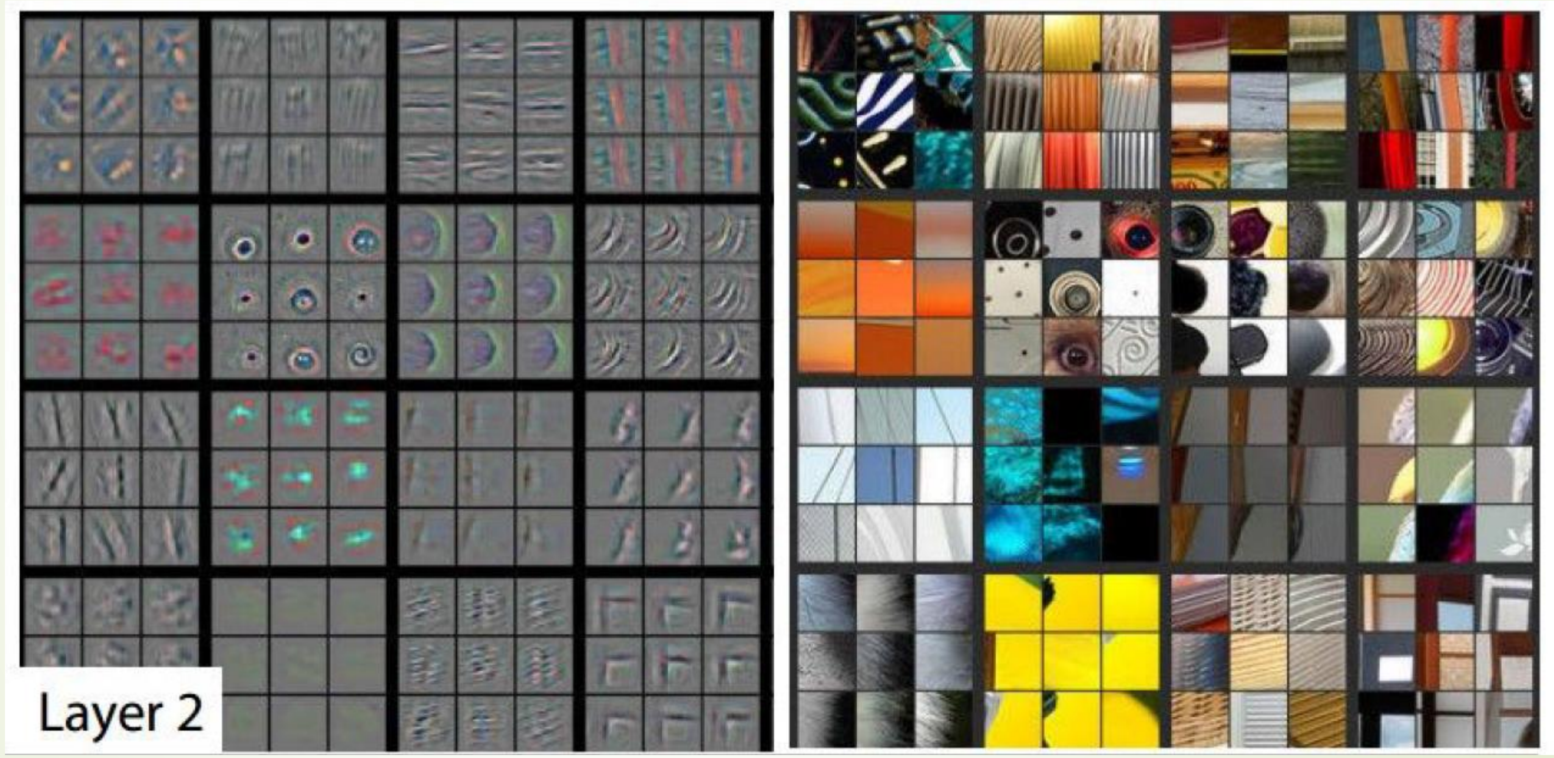
During evaluation, each filter is convolved across the input image to detect features like horizontal edges, color blobs, textures.

Krizhevsky et al. 2012

What are layers detecting?

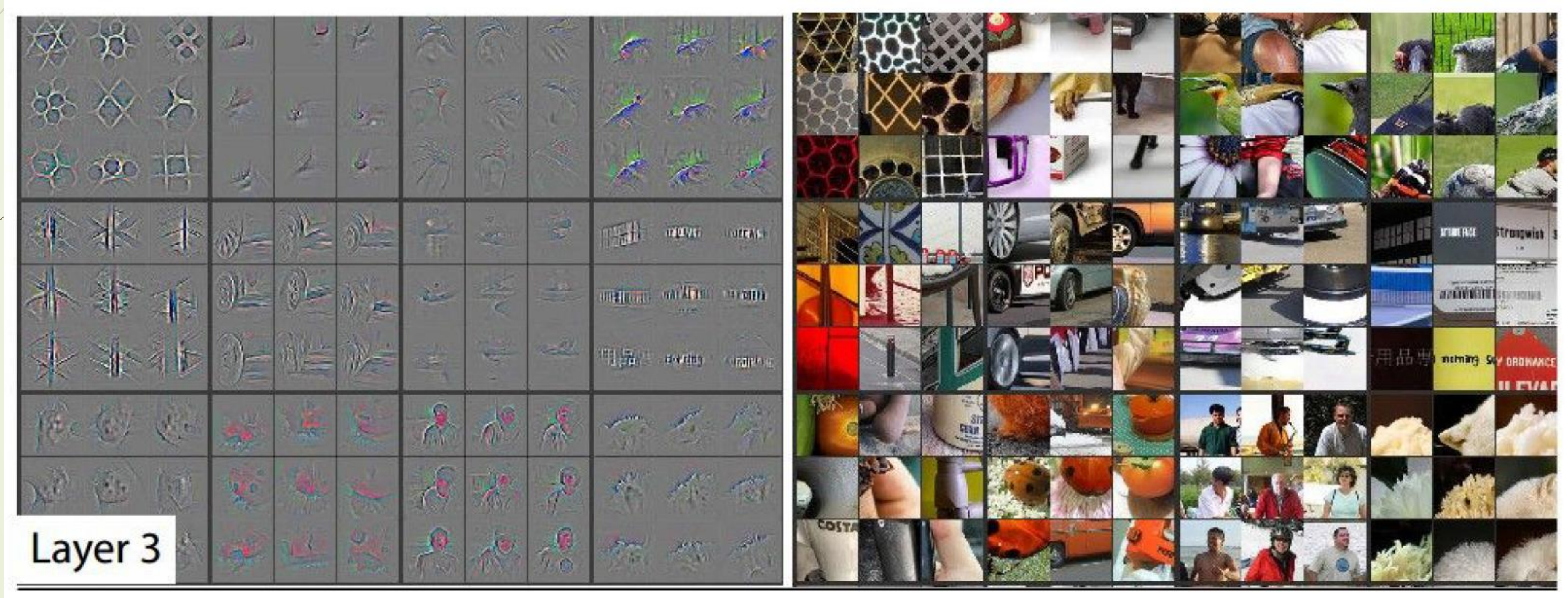
Zeiler and Fergus, 2013

What are layers detecting?



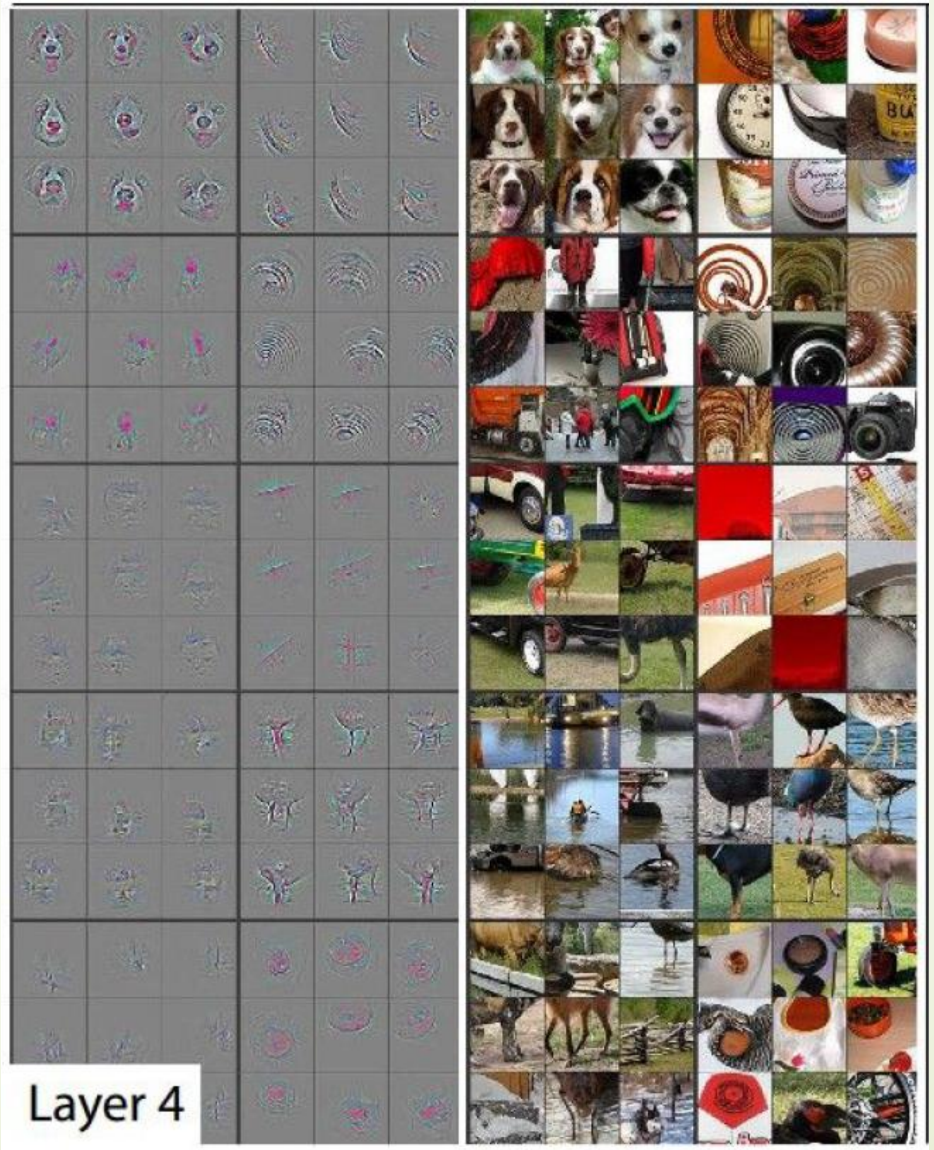
Zeiler and Fergus, 2013

What are layers detecting?



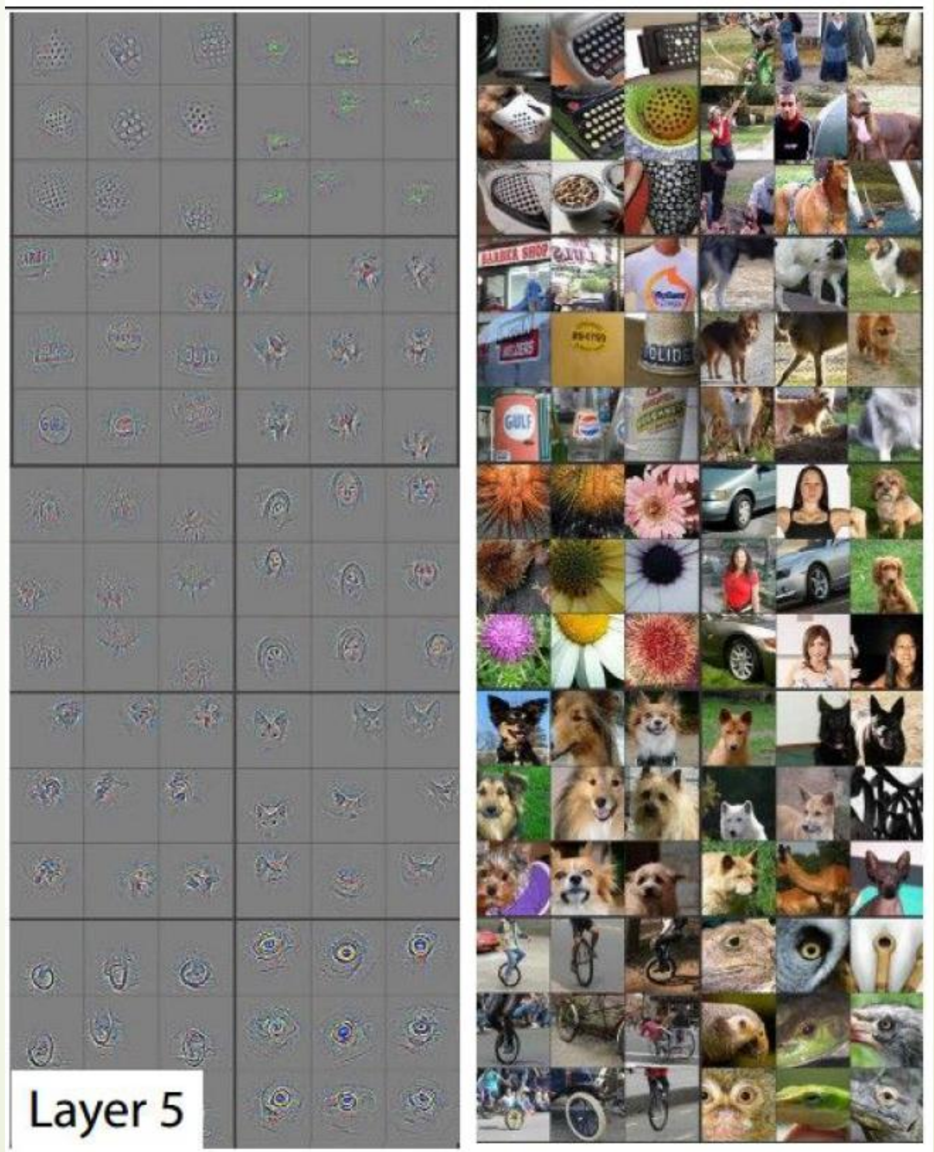
Zeiler and Fergus, 2013

What are layers detecting?



Zeiler and Fergus, 2013

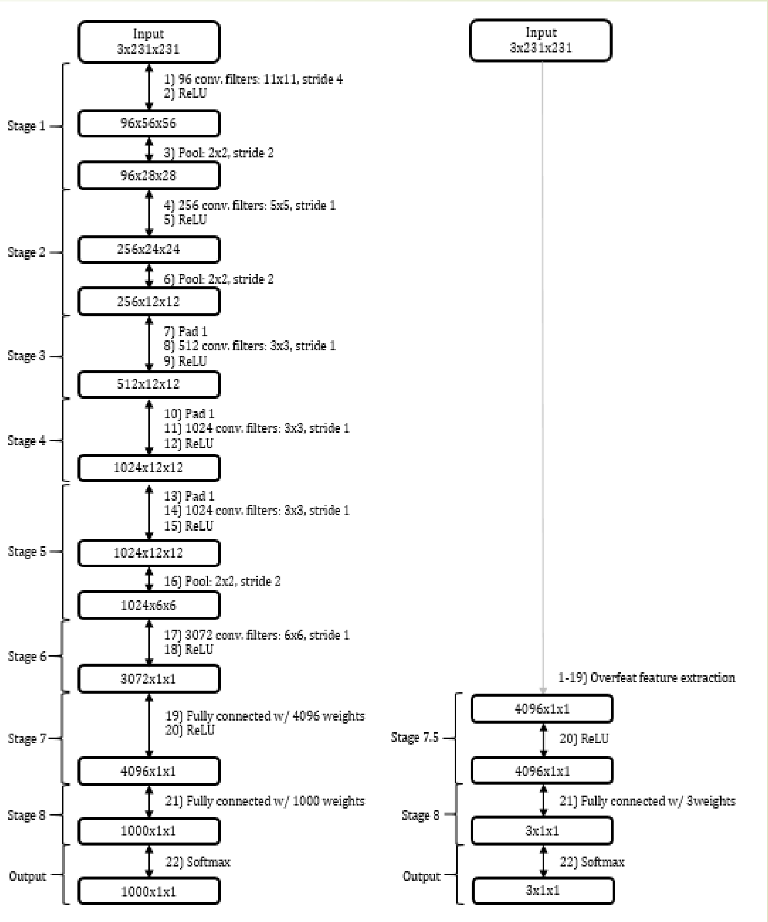
What are layers detecting?



Zeiler and Fergus, 2013

To use networks yourself…

- Neural networks have had most success when the data is:
  - Labeled
  - Exhaustive and low-level (e.g. images, audio)
  - Substantial in quantity
  - From a realm with unclear or underperformant theory-driven features
- More computing power (on GPUs) helps
- Limited data?
  No problem: use transfer learning



Left: Original network
Right: Retraining the classifier head

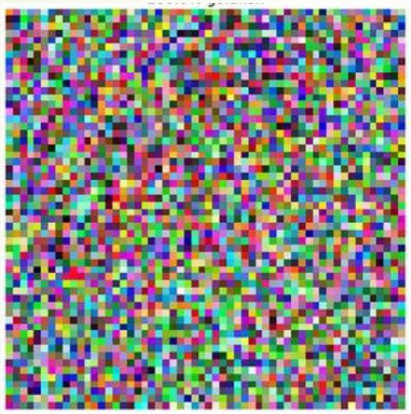
Software

- **Caffe** (UC Berkeley):
  the original; C++ with Python & MATLAB bindings; underdocumented; being revised
- **Torch** (NYU & IDIAP; Facebook, Google DeepMind):
  Lua; easy to convert to GPU; active development
- **Theano** (Montreal):
  Python; symbolic computation; two high level wrappers (Keras, Lasagne)
- **TensorFlow** (Google):
  Python; symbolic computation; multiple high level wrapper (Keras and others); helpful dashboards; extra parallelism

Fooling images

Stanford CS231N; Nguyen, Yosinski and Clune 2014; Szegedy et al. 2013

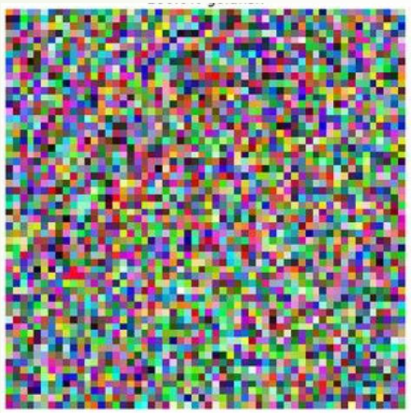
Fooling images



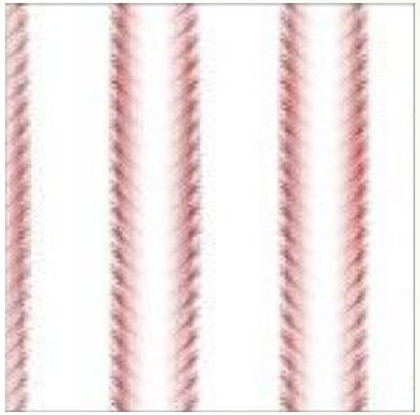
goldfish

Stanford CS231N; Nguyen, Yosinski and Clune 2014; Szegedy et al. 2013

Fooling images



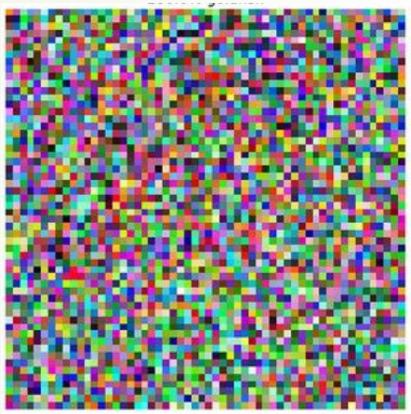
goldfish



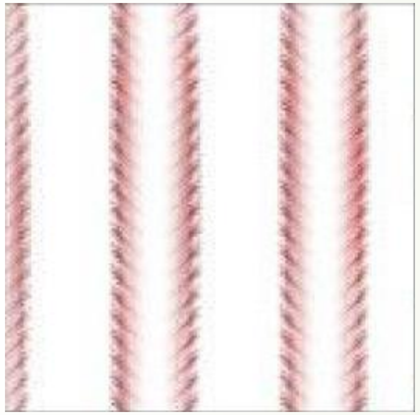
baseball

Stanford CS231N; Nguyen, Yosinski and Clune 2014; Szegedy et al. 2013

Fooling images



goldfish



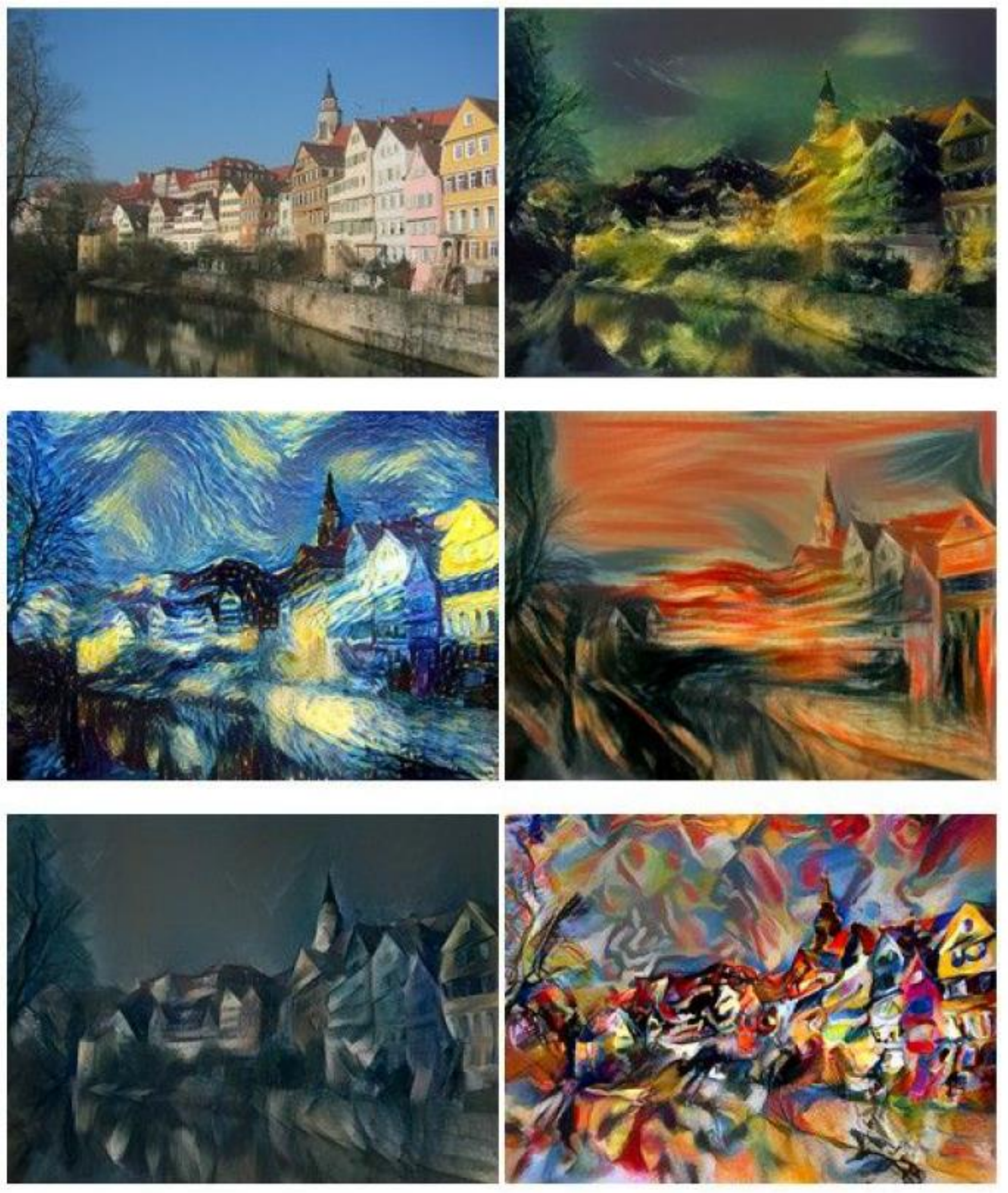
baseball

ostrich

Stanford CS231N; Nguyen, Yosinski and Clune 2014; Szegedy et al. 2013

Style transfer

- Content and style of art can be separated
- It is possible to keep underlying content structure and also approximate style tendencies



Gatys, Ecker, Bethge 2015; Justin Johnson

Image captioning

"little girl is eating piece of cake."

Image captioning

"little girl is eating piece of cake."

"baseball player is throwing ball in game."

Image captioning

"little girl is eating piece of cake."

"baseball player is throwing ball in game."

"woman is holding bunch of bananas."

Image captioning

"little girl is eating piece of cake."

"baseball player is throwing ball in game."

"woman is holding bunch of bananas."

"a cat is sitting on a couch with a remote control."

Image captioning

"little girl is eating piece of cake."

"baseball player is throwing ball in game."

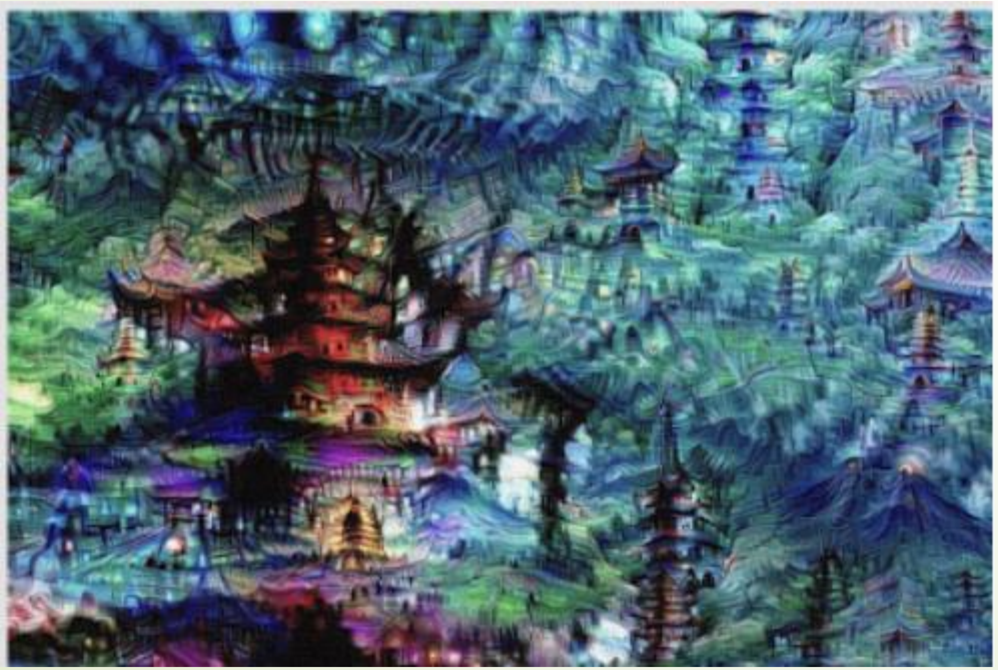
"woman is holding bunch of bananas."

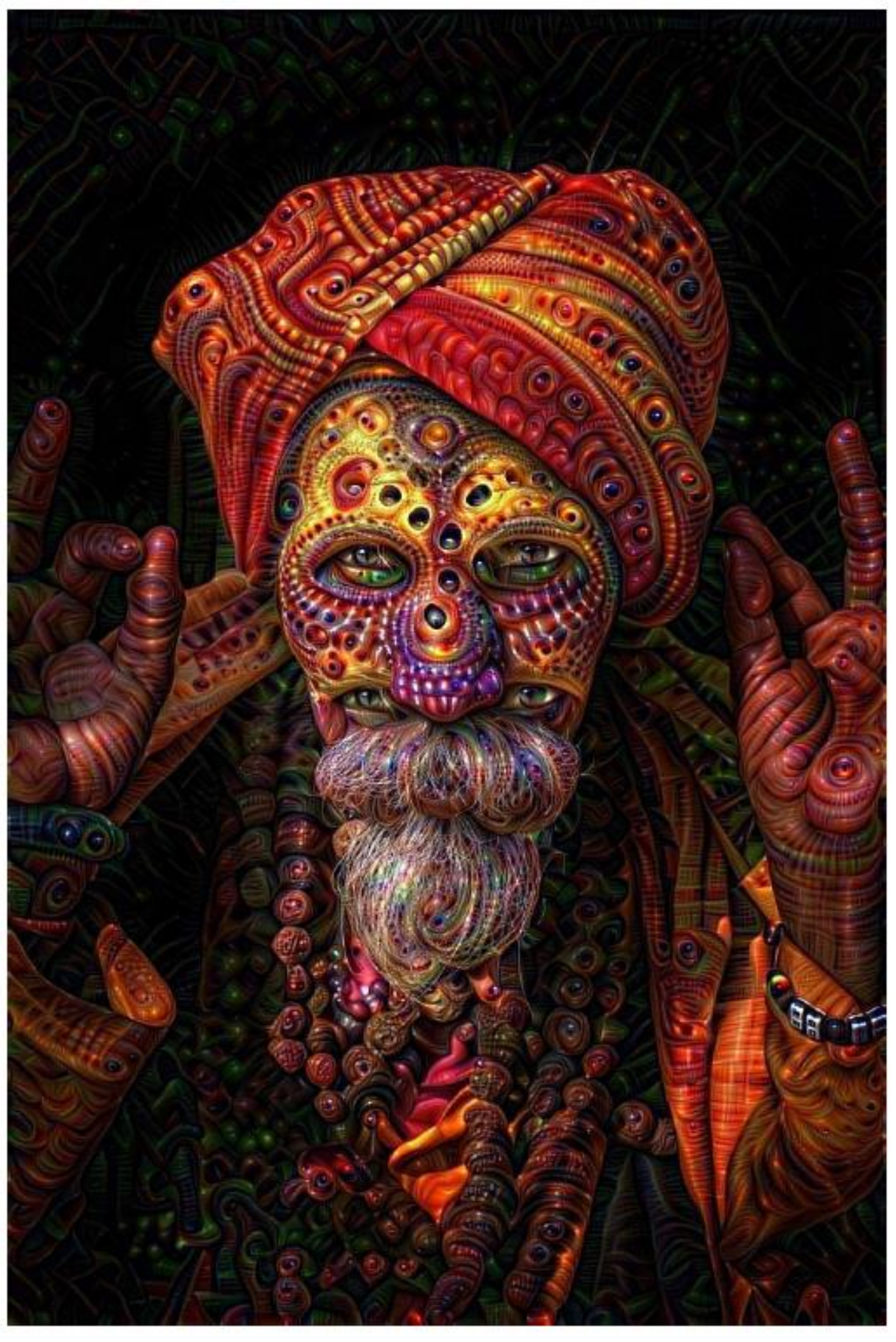
"a cat is sitting on a couch with a remote control."

"a young boy is holding a baseball bat."

Google DeepDream

- Whatever the image looks like in a region, make it look more like that
- DeepDream goes to the grocery store: https://www.youtube.com/watch?v=DgPaCWJL7XI





Google Research; Stanford CS 231N

Further resources

- To learn more:
  - http://cs231n.stanford.edu/ [texts, slides, YouTube videos, homework]
  - https://www.coursera.org/course/neuralnets [more math, less vision]
- Software:
  - https://github.com/torch
  - https://www.tensorflow.org/
  - http://caffe.berkeleyvision.org/
  - http://deeplearning.net/software/theano/
- ImageNet
  - http://www.image-net.org/
- Caffe model zoo:
  - https://github.com/BVLC/caffe/wiki/Model-Zoo

At 1:00, you are now able to…

- **Competence** (able to perform on your own, with varying levels of perfection):
  - Understand how neural networks extend linear classification, and have some intuition for how and why they are more powerful
  - Know when a neural network variant might be appropriate for your problem and why
  - Know how to get more help
- **Exposure** (aware):
  - Articulate some of the challenges in computer vision
  - Articulate the broad strokes of gradient descent
  - Recognize the phrase "backpropagation" (it is how we train networks)
  - Recognize the phrase "convolutional neural network" (it's state-of-the-art vision)
  - Express the history of neural networks and some reasons deep learning has been causing so much excitement in recent years
  - Be familiar with tools that make networks easier to use: transfer learning + software
  - Recall images from a handful of cool recent papers