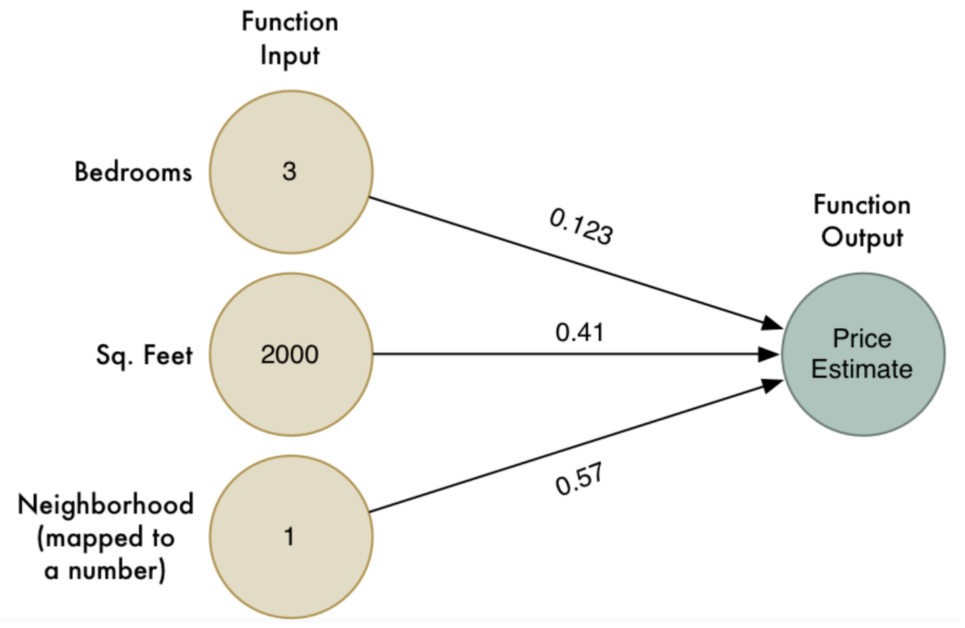
Machine Learning - Part 3

Apr. 15, 2025

Recap question:

A real estate agent computes their price estimate by pretending to be a neuron in a neural network as in:



Which of the following is a valid expression for the price estimate computed by a neural network?

A) $\tanh(0.123 \times 3 + 0.41 \times 2000 + 0.57 \times 1)$

B) $\tanh(3^{0.123} + 0.41 \times 2000 \times 0.57 + 1)$

Recap question:

The answer is A!

For each neuron, we multiply the value coming in from each arrow by its corresponding weight. We add these together and then apply an **activation function** to it (here, tanh). This becomes the **value** of that neuron.

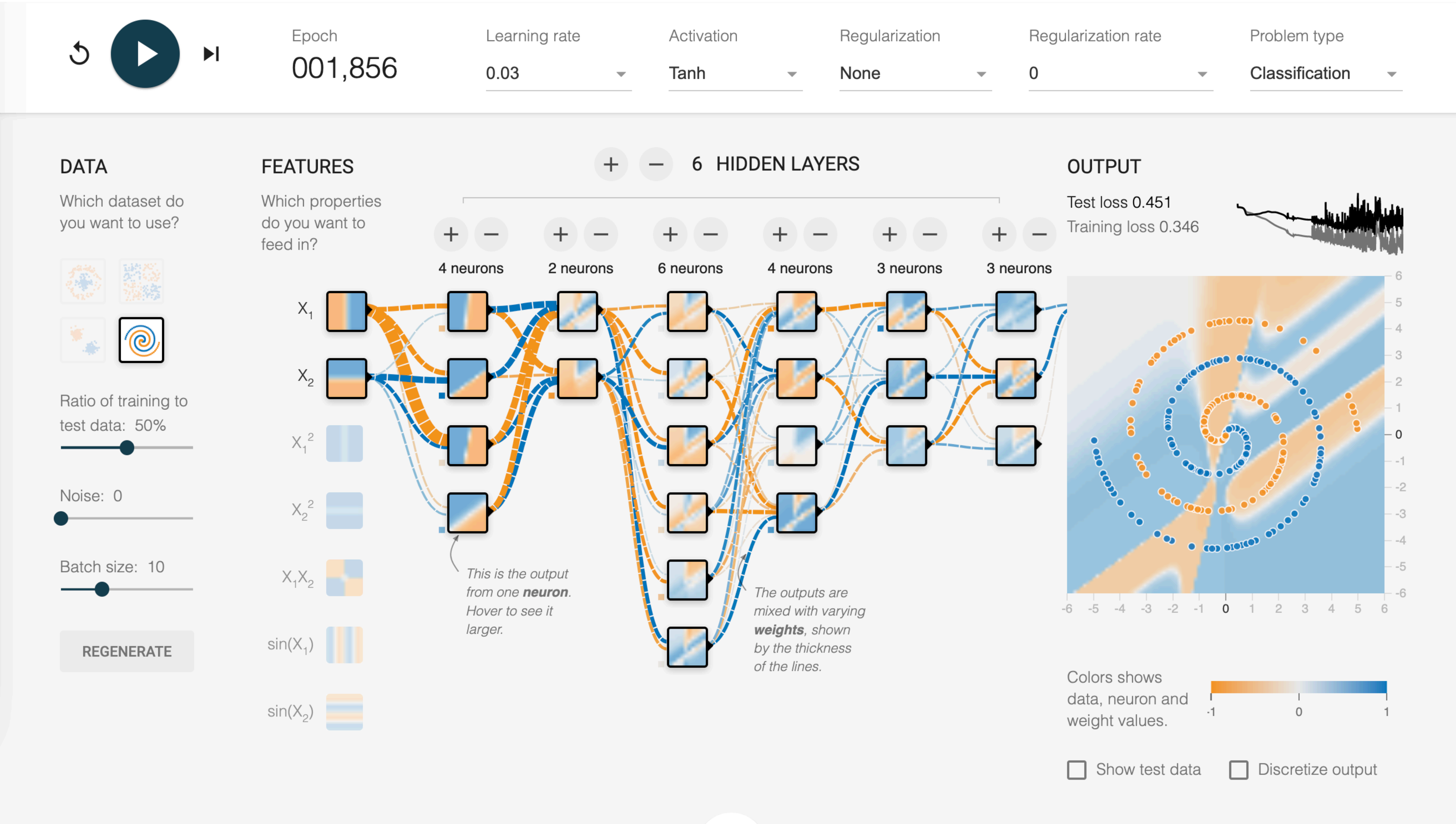
Machine Learning - Part 3

April 15, 2025

By the end of this lecture, you will be able to:

1. Discuss the cost of machine learning
2. Define a convolutional neural network
3. Explain how to use neural networks for facial detection

https://playground.tensorflow.org/



Epoch 001,856

Learning rate 0.03

Activation Tanh

Regularization None

Regularization rate 0

Problem type Classification

DATA

Which dataset do you want to use?

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

REGENERATE

FEATURES

Which properties do you want to feed in?

6 HIDDEN LAYERS

4 neurons, 2 neurons, 6 neurons, 4 neurons, 3 neurons, 3 neurons

*This is the output from one **neuron**. Hover to see it larger.*

*The outputs are mixed with varying **weights**, shown by the thickness of the lines.*

OUTPUT

Test loss 0.451

Training loss 0.346

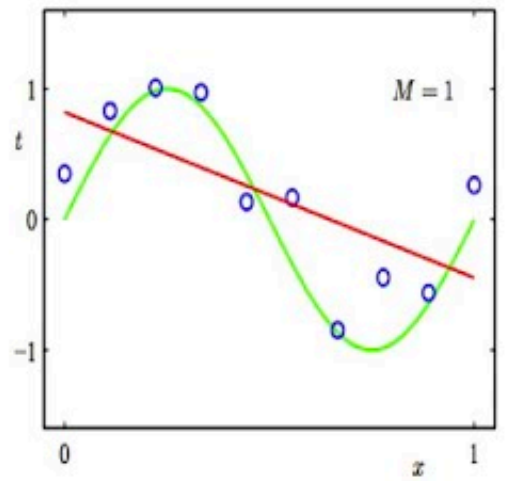
Colors shows data, neuron and weight values.

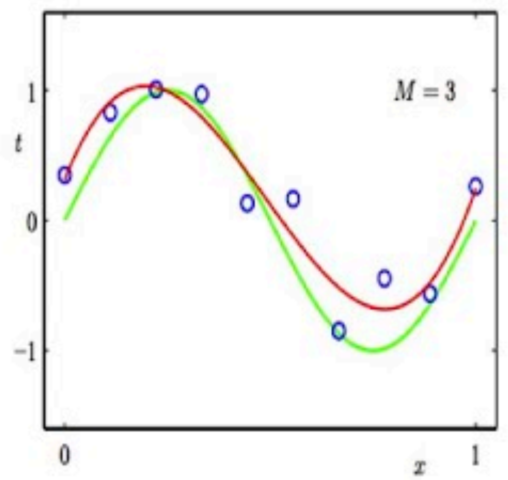
Show test data

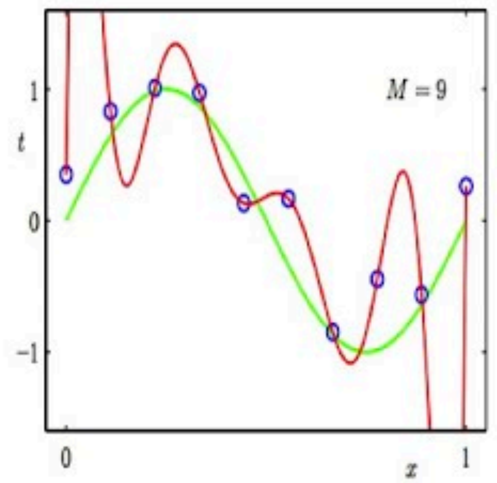
Discretize output

Under- and Over-fitting examples

Regression:



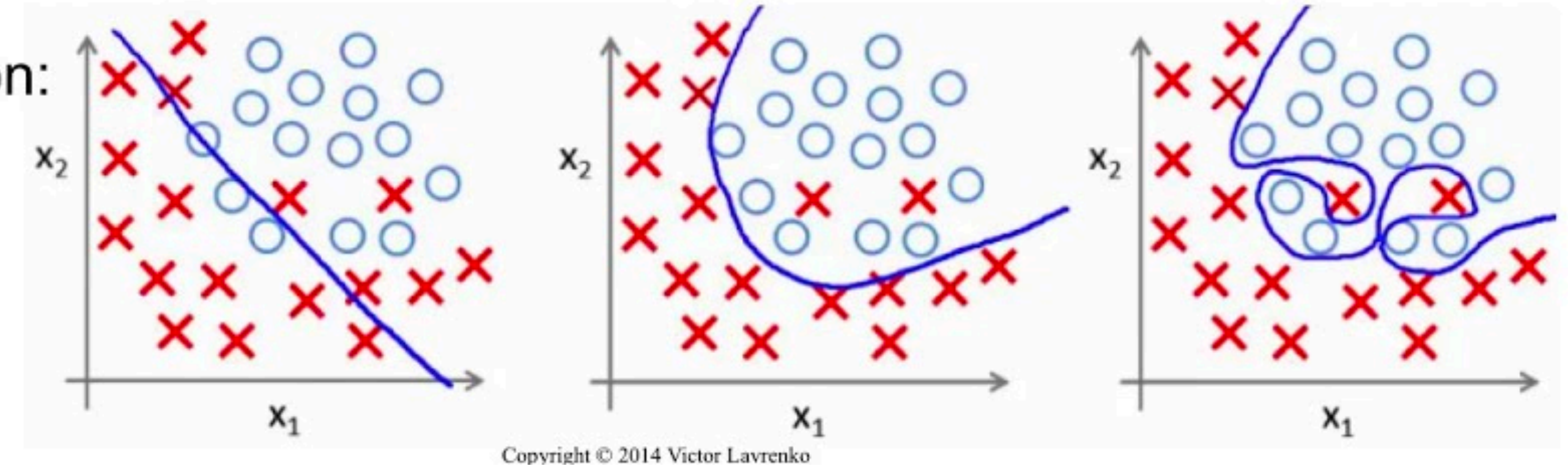


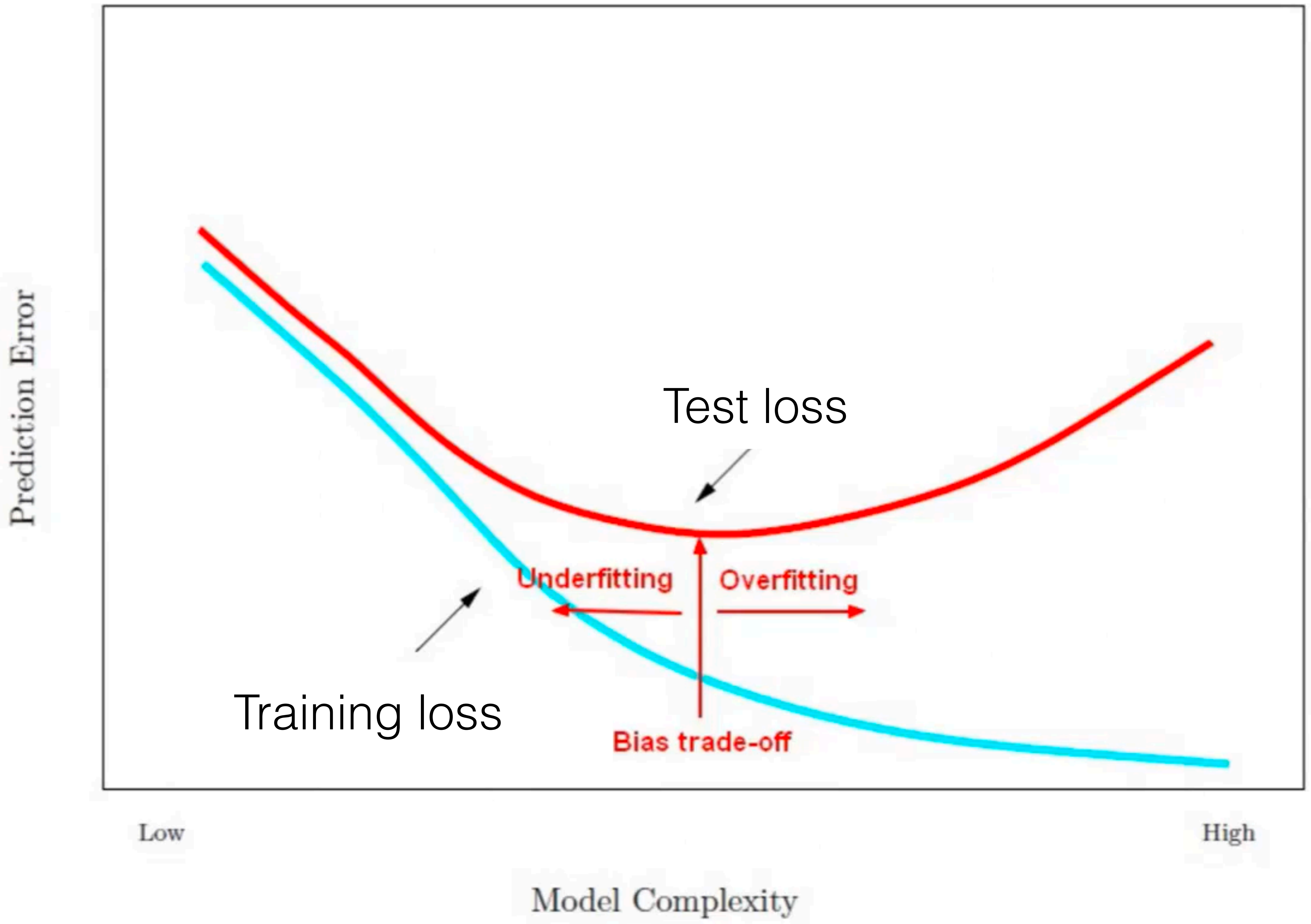


predictor too inflexible: cannot capture pattern

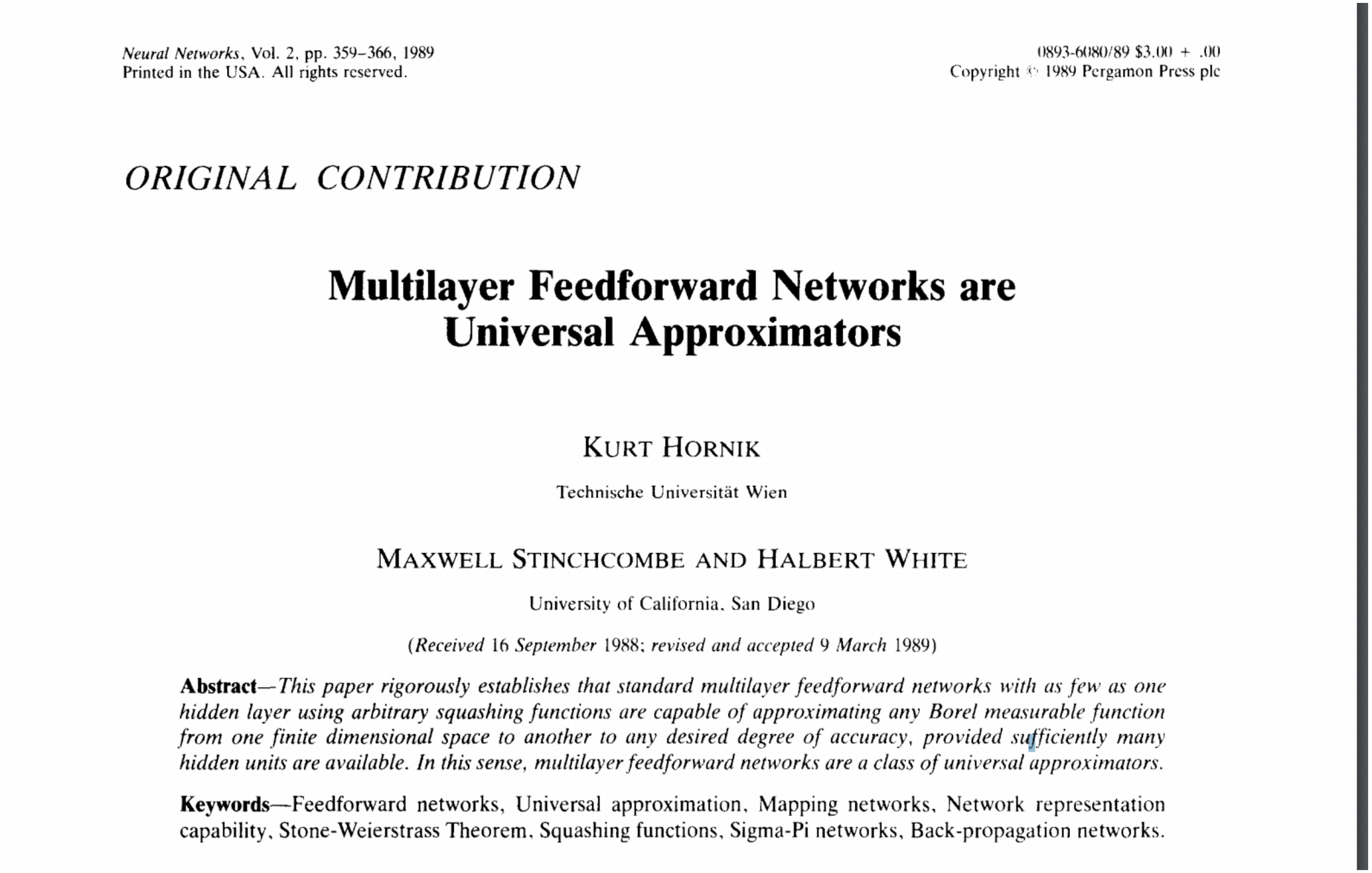
predictor too flexible: fits noise in the data

Classification:




Prediction Error
Test loss
Underfitting
Overfitting
Training loss
Bias trade-off
Low
High
Model Complexity

Deep Neural Networks (DNN)



Neural Networks, Vol. 2, pp. 359–366, 1989
Printed in the USA. All rights reserved.

ORIGINAL CONTRIBUTION

Multilayer Feedforward Networks are Universal Approximators

KURT HORNIK
Technische Universität Wien

MAXWELL STINCHCOMBE AND HALBERT WHITE
University of California, San Diego

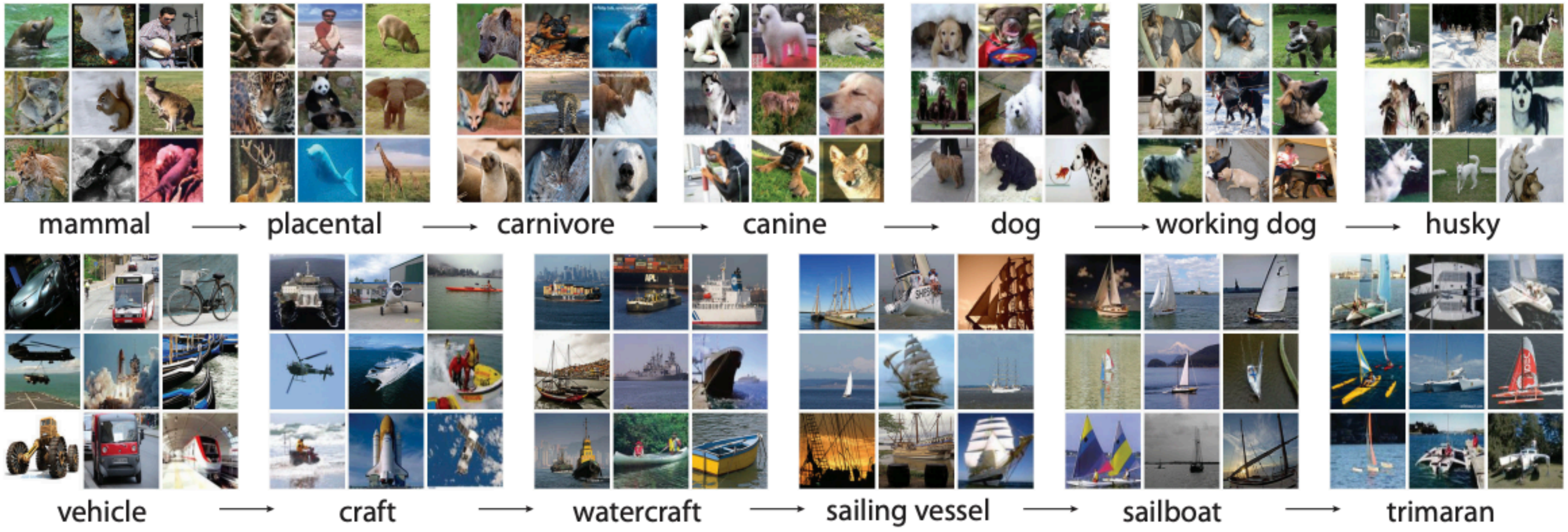
(*Received* 16 *September* 1988; *revised and accepted* 9 *March* 1989)

Abstract—*This paper rigorously establishes that standard multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available. In this sense, multilayer feedforward networks are a class of universal approximators.*

Keywords—Feedforward networks, Universal approximation, Mapping networks, Network representation capability, Stone-Weierstrass Theorem, Squashing functions, Sigma-Pi networks, Back-propagation networks.

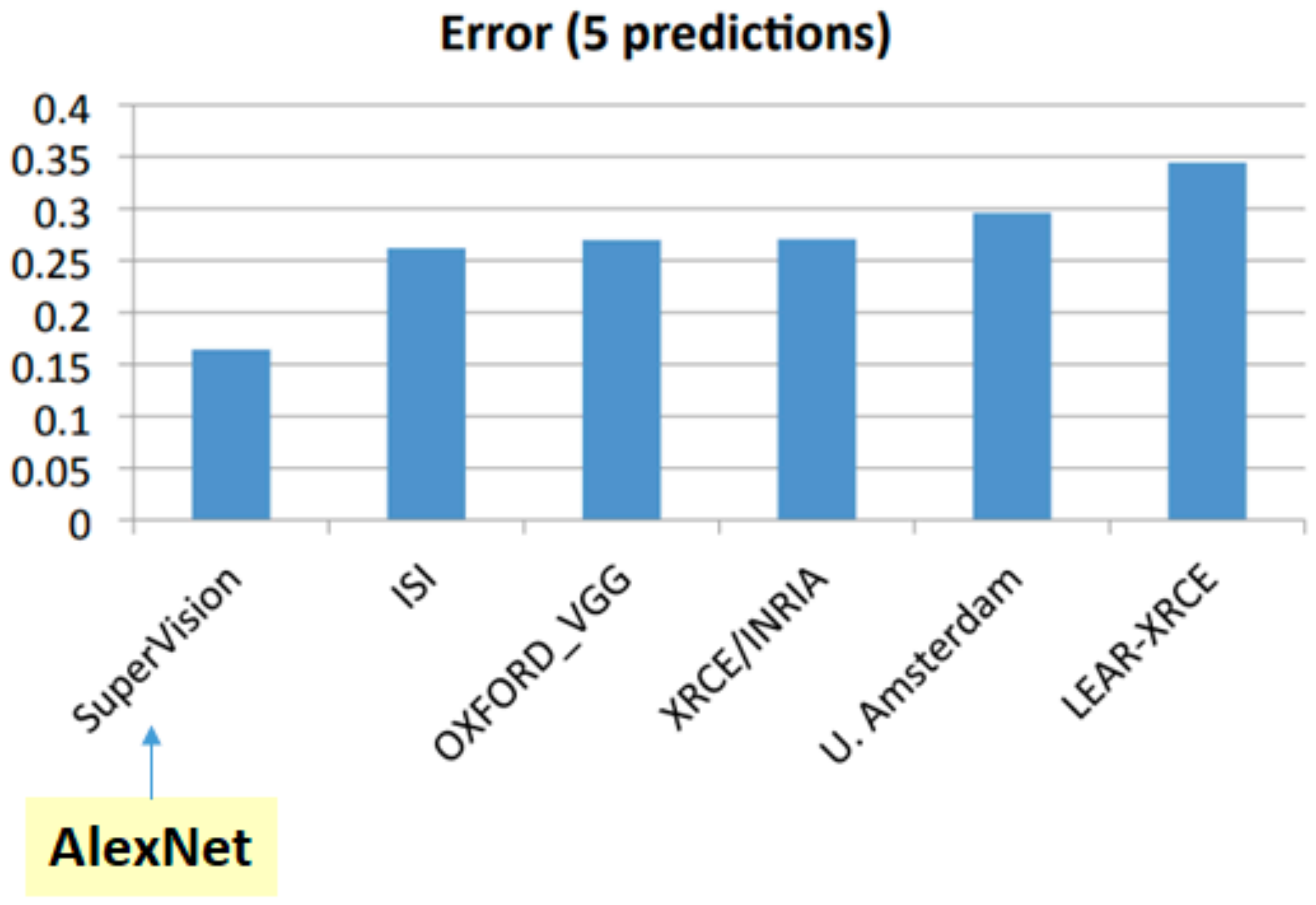
Many researchers did not believe that neural networks are useful, until the ImageNet Challenge in 2012.

More than 14M images divided into 20 000 categories.
Task: perform classification!

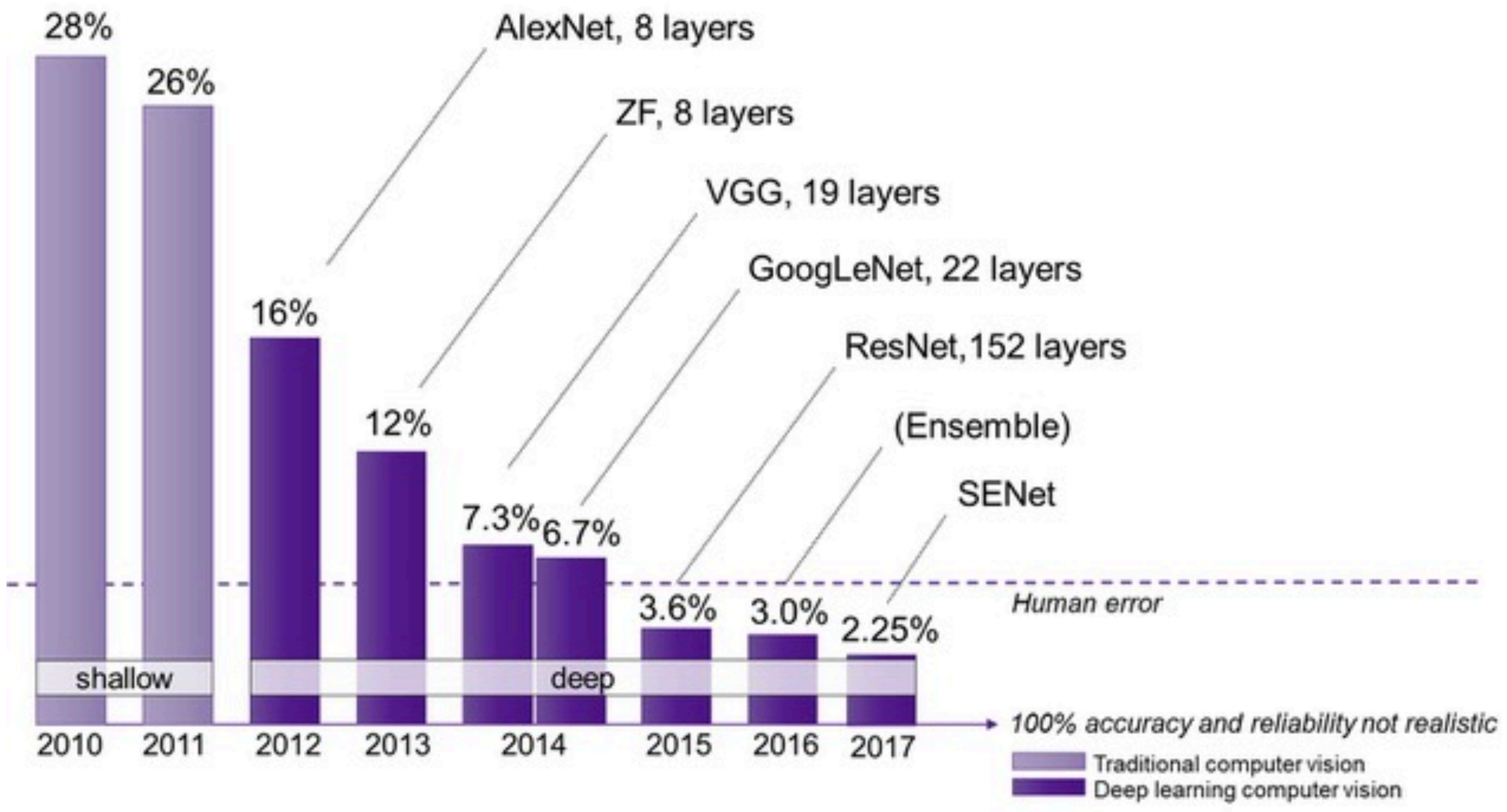


ImageNet Challenge (2012)

Ranking of the best results from each team



ImageNet Challenge

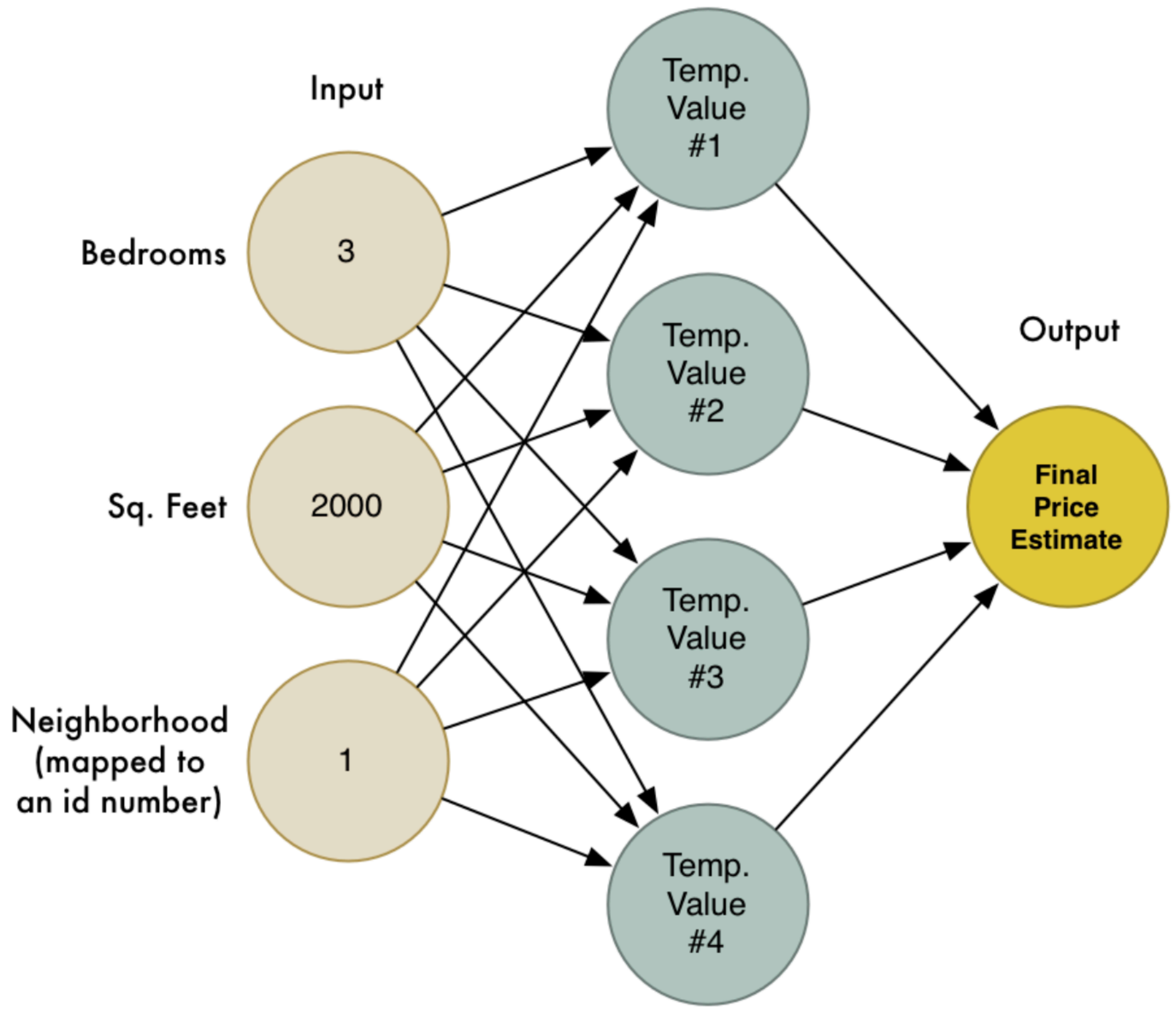


What are the drivers of the improvements?

1. Better hardware
2. Better training algorithms

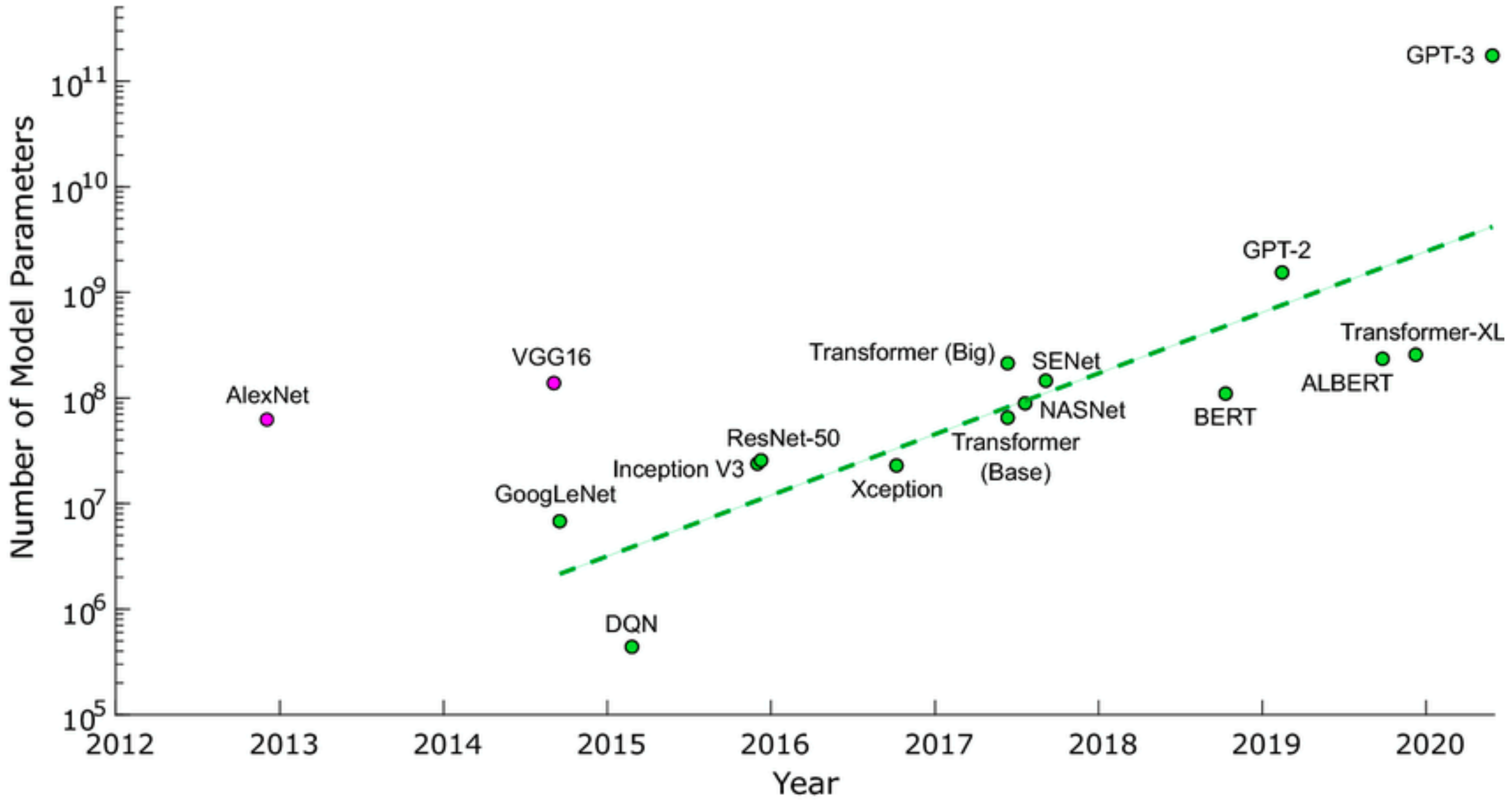
Together, they allow us to train larger and larger networks, using more parameters, which gives better performance.

The number of weights increases

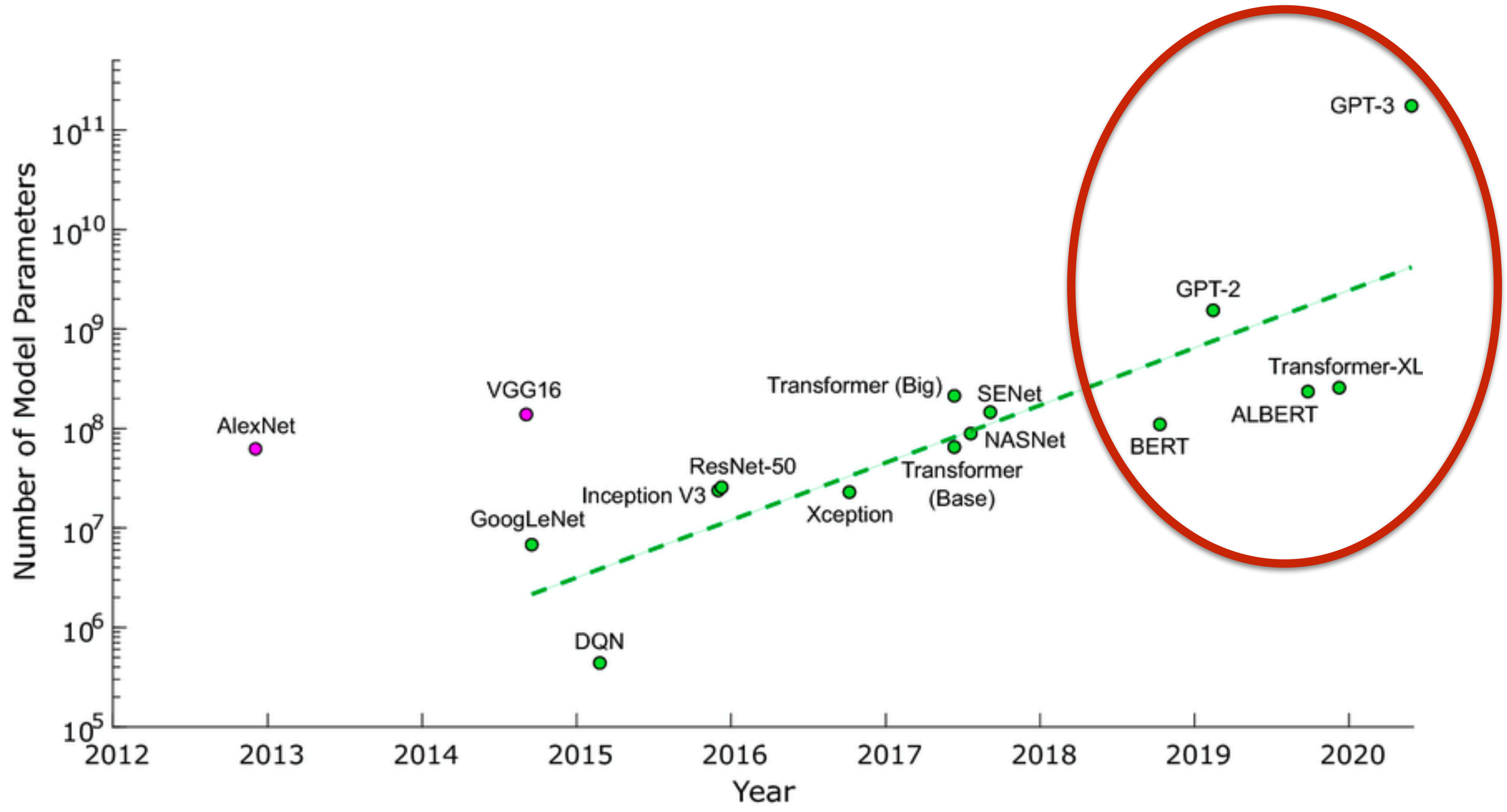


16 weights (since there are 16 arrows)!

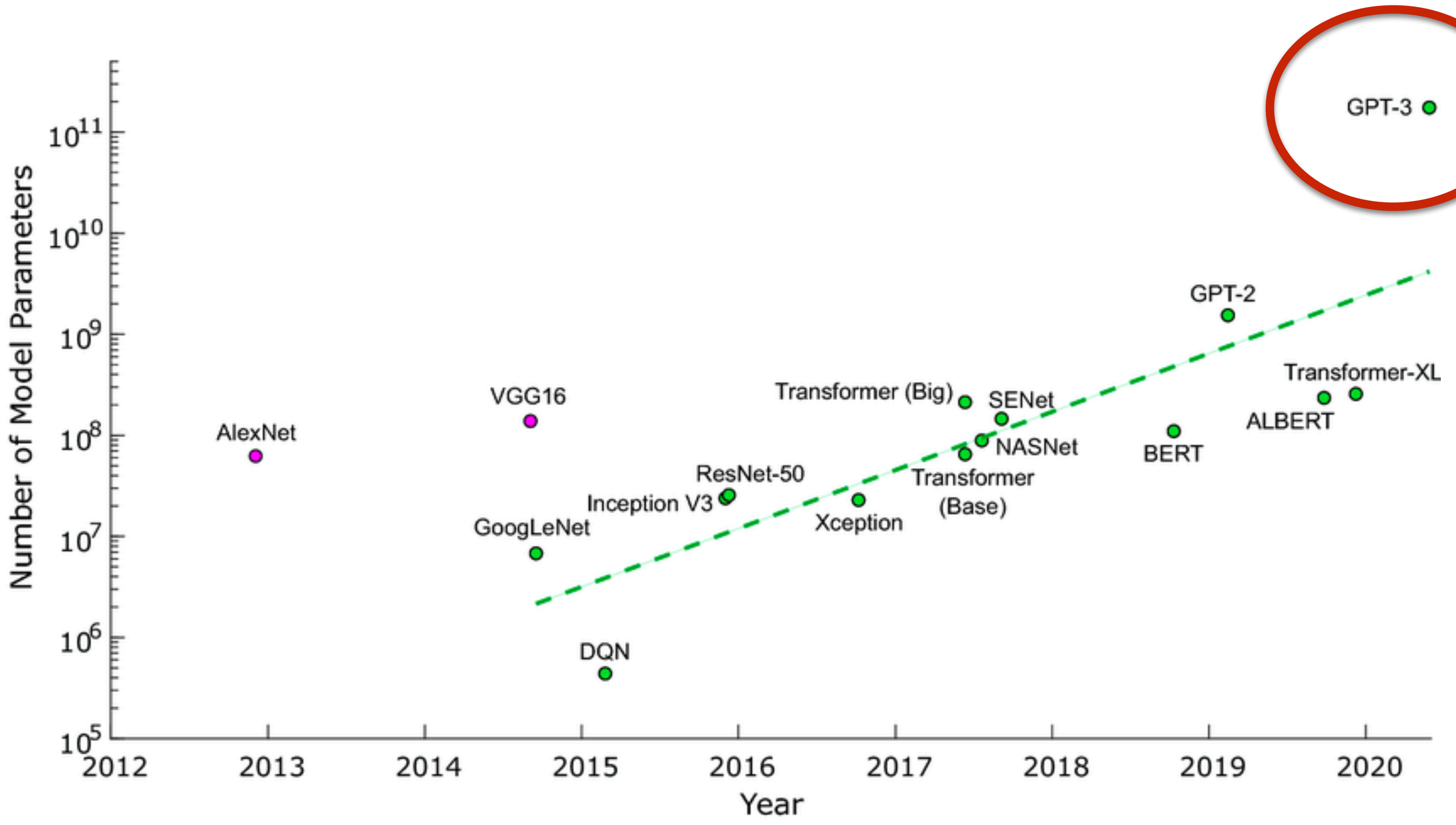
The number of weights increases

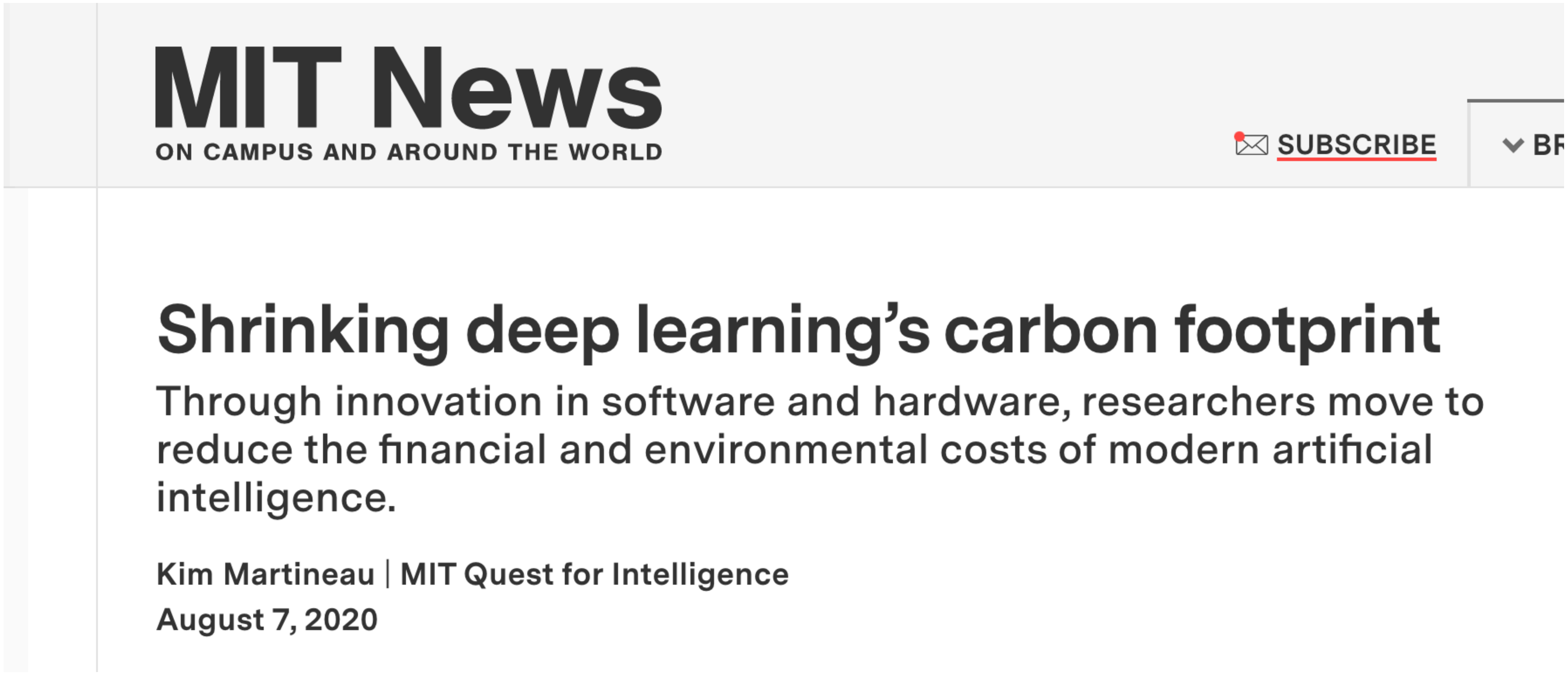


The number of weights increases



The number of weights increases




MIT News

ON CAMPUS AND AROUND THE WORLD

SUBSCRIBE

BR

Shrinking deep learning's carbon footprint

Through innovation in software and hardware, researchers move to reduce the financial and environmental costs of modern artificial intelligence.

Kim Martineau | MIT Quest for Intelligence

August 7, 2020

The training of GPT-3 cost $4.6 million and 355 years in computing time on a single computer. Training smaller models than GPT-3 releases around 626,000 pounds of CO2.

Factors that affect the performance of a Neural Network:

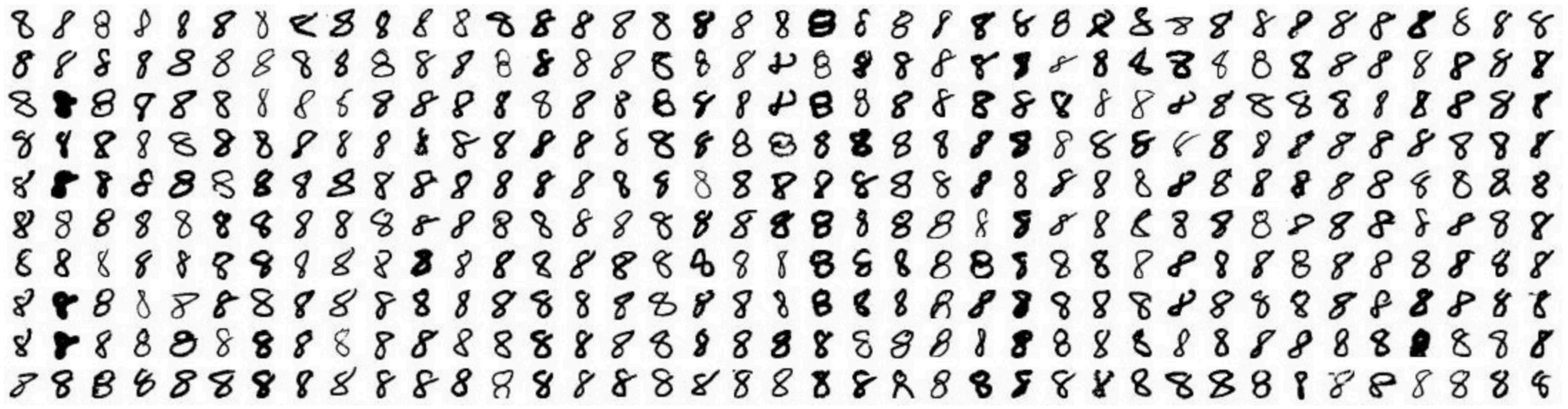
1. Type of data (spiral data set vs the other ones)
2. The architecture of the neural network (number of hidden layers, number of neurons in each hidden layer and the choice of activation function)
3. The method used to train the network from the training data (which we will not discuss)

Remaining topics in the class

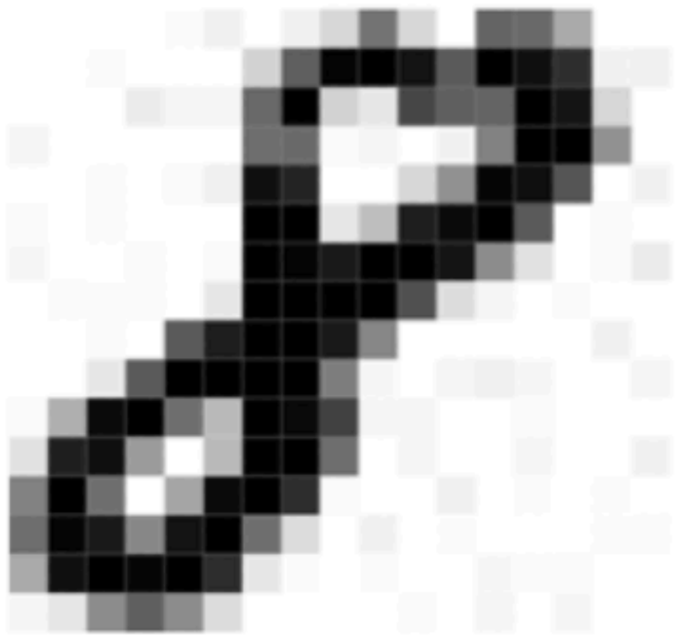
1. Image processing and facial recognition
2. Natural Language Processing (NLP) and the Turing test
3. Generative Adversarial Networks (GAN) and Adversarial Attacks on Neural Networks

Facial recognition and image processing

Image processing and facial recognition



Let’s first see how computers can recognize the hand-written digit “8”.

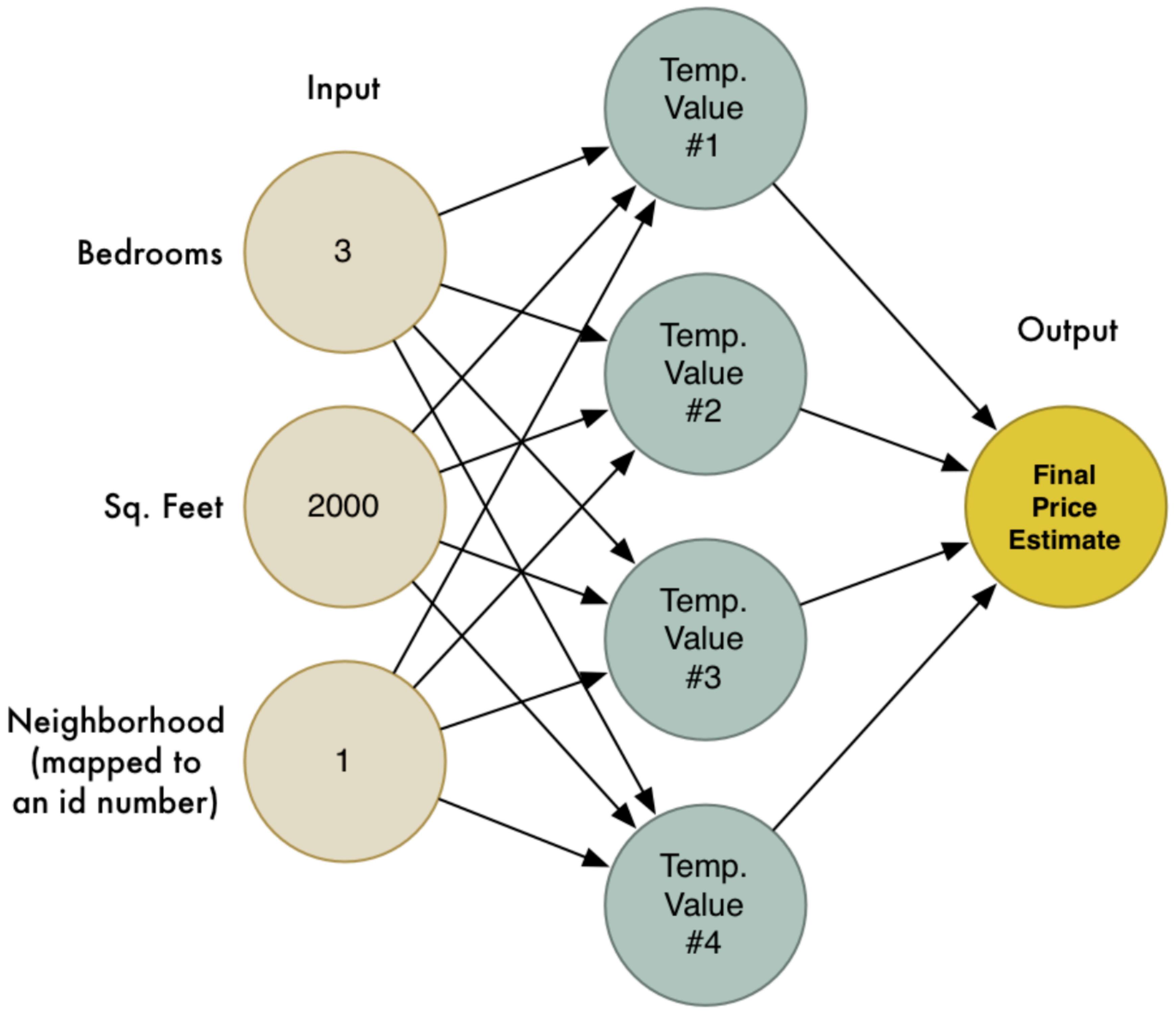


=

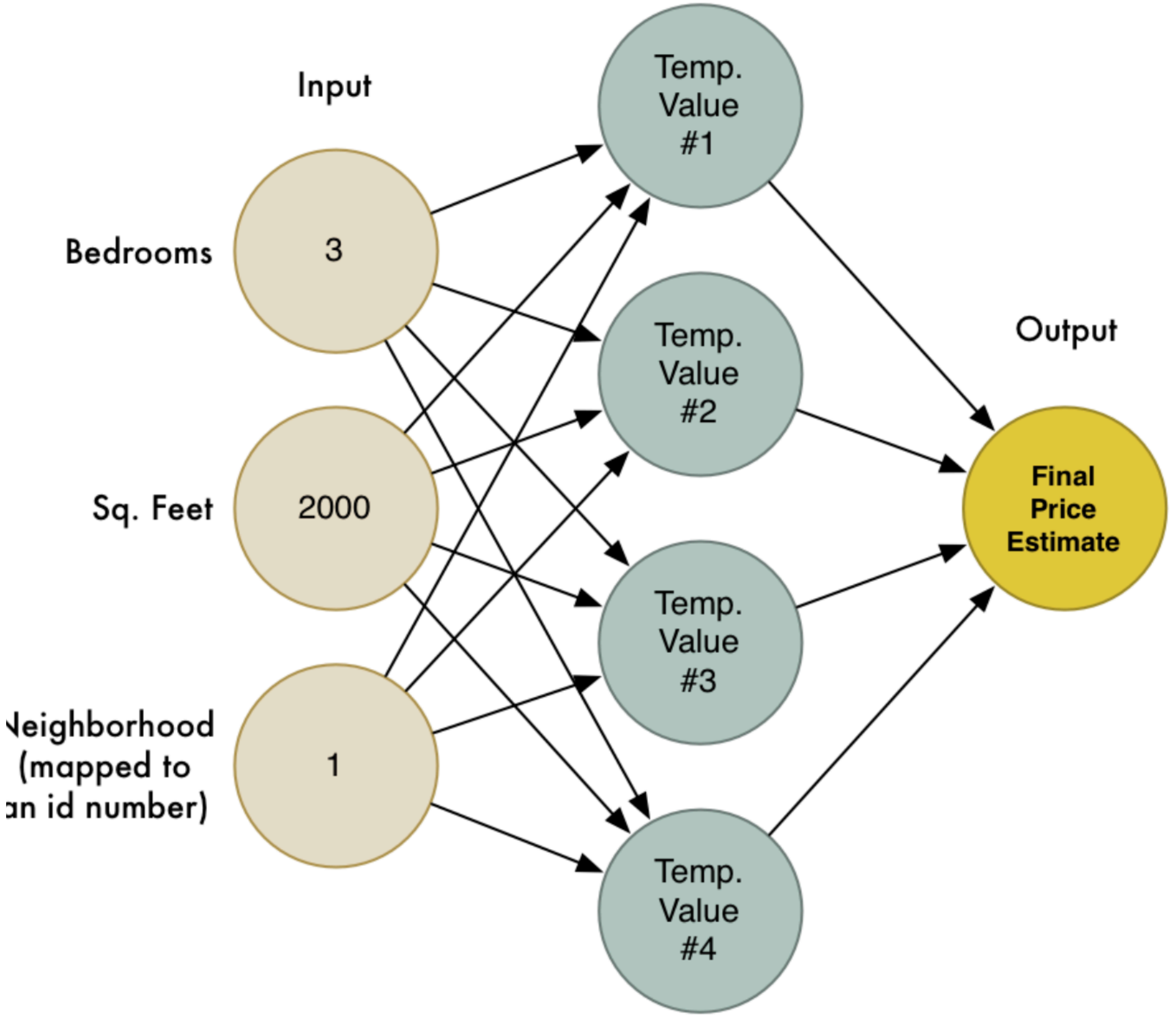
```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 12, 0, 11, 39, 137, 37, 0, 152, 147, 84, 0, 0, 0, 0,
0, 1, 0, 0, 0, 41, 160, 250, 255, 235, 162, 255, 238, 206, 11, 13, 0, 0, 0, 0, 16, 9, 9, 150, 251, 45, 21, 184, 159, 154, 2
55, 233, 40, 0, 0, 10, 0, 0, 0, 0, 0, 145, 146, 3, 10, 0, 11, 124, 253, 255, 107, 0, 0, 0, 0, 3, 0, 4, 15, 236, 216, 0, 0,
38, 109, 247, 240, 169, 0, 11, 0, 1, 0, 2, 0, 0, 0, 253, 253, 23, 62, 224, 241, 255, 164, 0, 5, 0, 0, 6, 0, 0, 4, 0, 3, 252
, 250, 228, 255, 255, 234, 112, 28, 0, 2, 17, 0, 0, 2, 1, 4, 0, 21, 255, 253, 251, 255, 172, 31, 8, 0, 1, 0, 0, 0, 0, 0, 4,
 0, 163, 225, 251, 255, 229, 120, 0, 0, 0, 0, 0, 11, 0, 0, 0, 0, 21, 162, 255, 255, 254, 255, 126, 6, 0, 10, 14, 6, 0, 0, 9
, 0, 3, 79, 242, 255, 141, 66, 255, 245, 189, 7, 8, 0, 0, 5, 0, 0, 0, 0, 26, 221, 237, 98, 0, 67, 251, 255, 144, 0, 8, 0, 0
, 7, 0, 0, 11, 0, 125, 255, 141, 0, 87, 244, 255, 208, 3, 0, 0, 13, 0, 1, 0, 1, 0, 0, 145, 248, 228, 116, 235, 255, 141, 34
, 0, 11, 0, 1, 0, 0, 0, 1, 3, 0, 85, 237, 253, 246, 255, 210, 21, 1, 0, 1, 0, 0, 6, 2, 4, 0, 0, 0, 6, 23, 112, 157, 114, 32
, 0, 0, 0, 0, 2, 0, 8, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

We can treat an 18x18 pixel image as a list of 18x18=324 numbers, containing the greyscale level of each pixel.

Designing a neural network

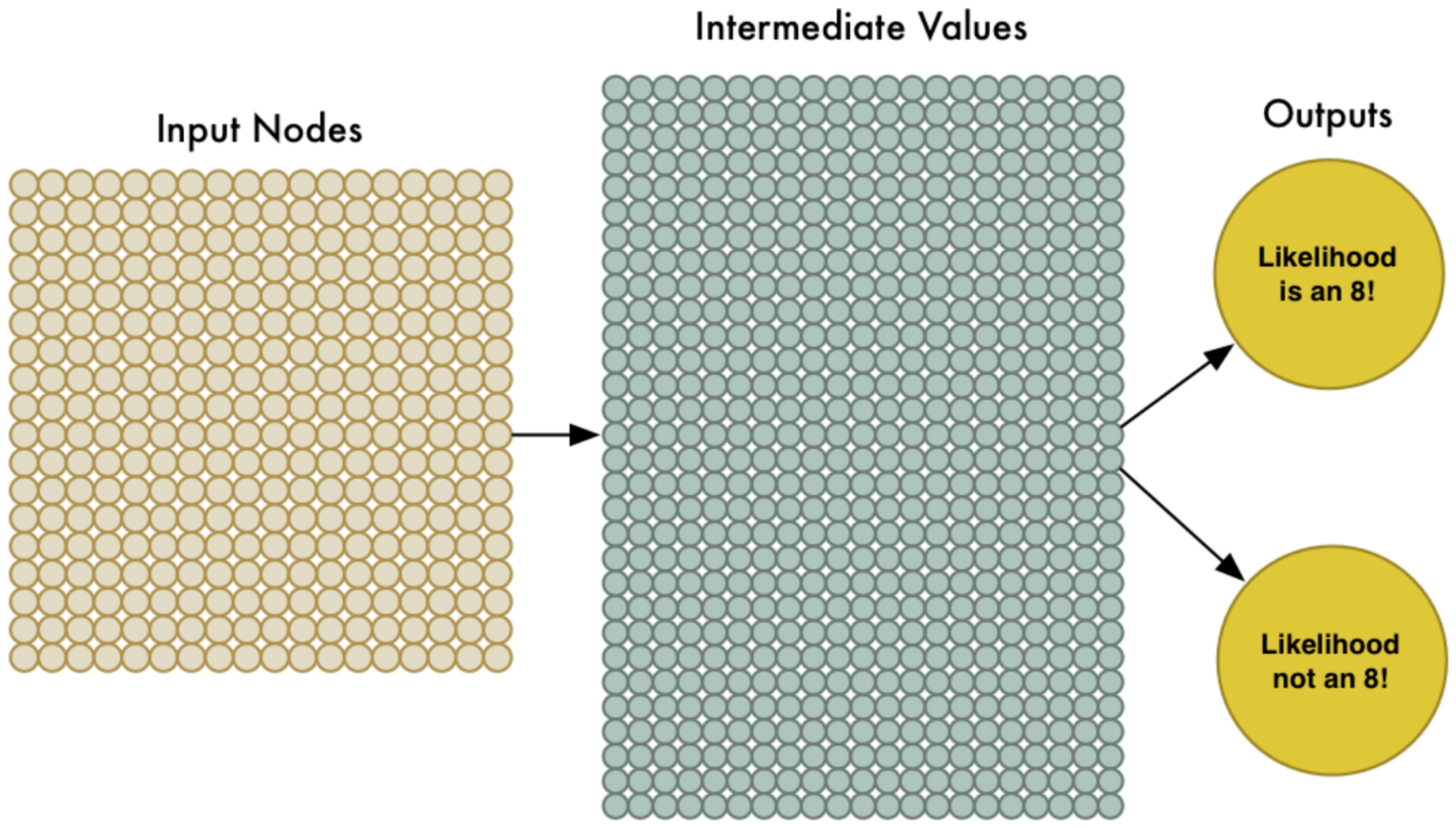


Designing a neural network



Too small!

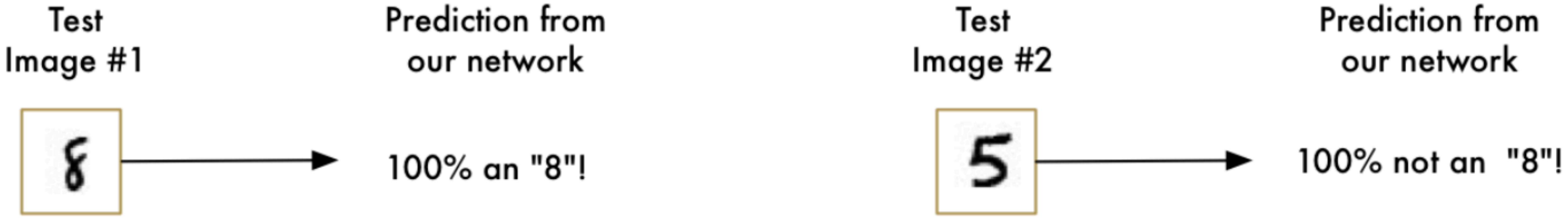
Designing a neural network



Training the neural net

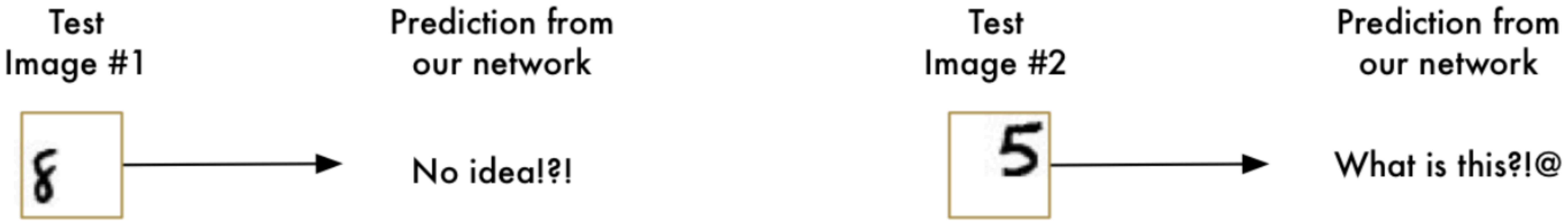
When we feed in an “8”, we’ll tell it the probability the image is an “8” is 100% and the probability it’s not an “8” is 0%. Vice versa for the counter-example images.

Testing the neural net



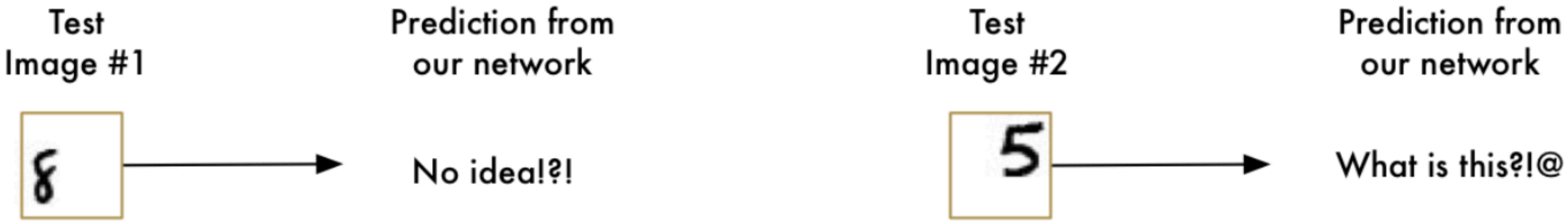
The good news is that our "8" recognizer does work well on simple images where the letter is right in the middle of the image.

Testing the neural net



The bad news is that our “8” recognizer *totally fails* to work when the letter isn’t perfectly centered in the image. Just the slightest position change ruins everything.

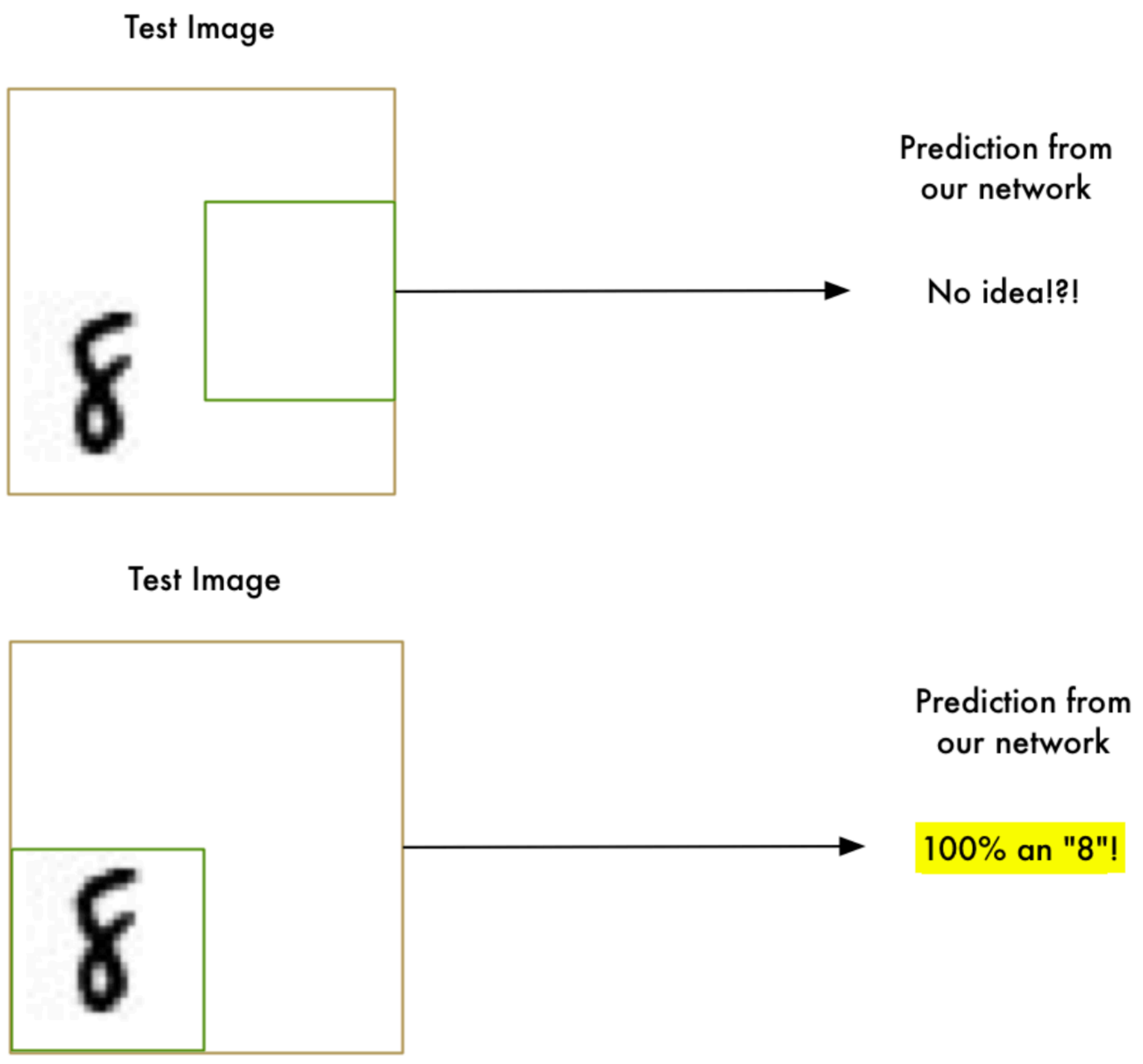
Testing the neural net



The bad news is that our “8” recognizer *totally fails* to work when the letter isn’t perfectly centered in the image. Just the slightest position change ruins everything.

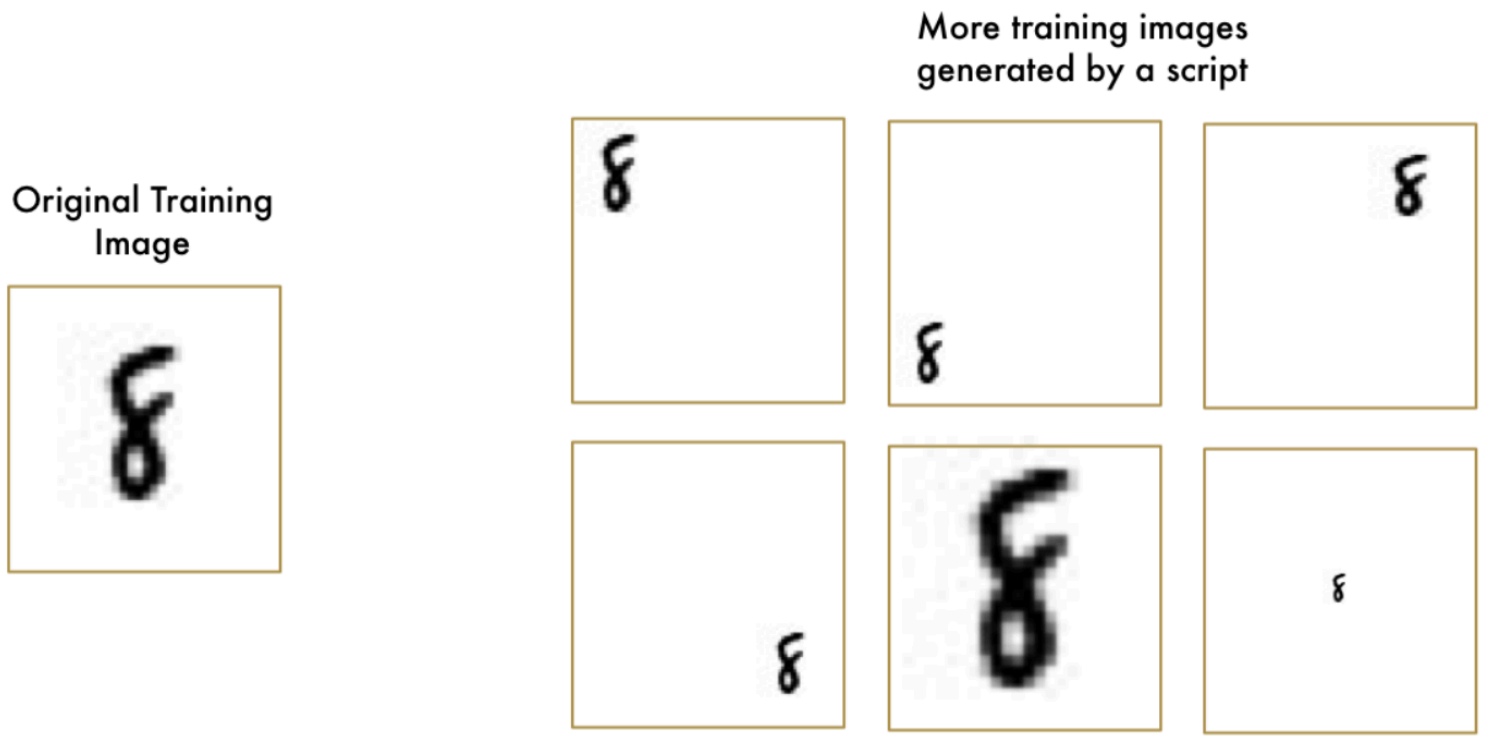
How can we mitigate this?

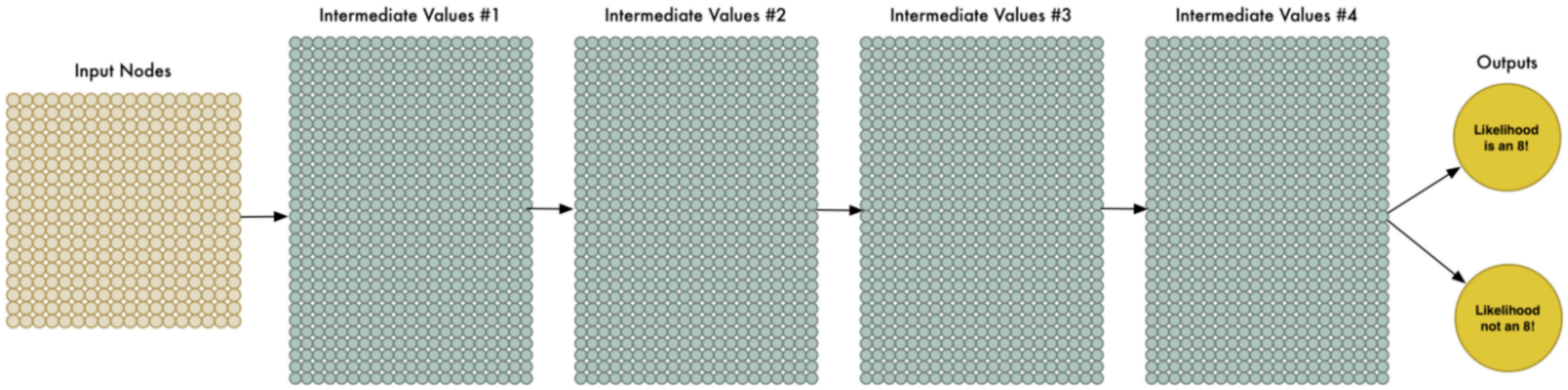
Idea #1: Searching with a sliding window



Very inefficient!

Idea #2: More data and a deep neural net





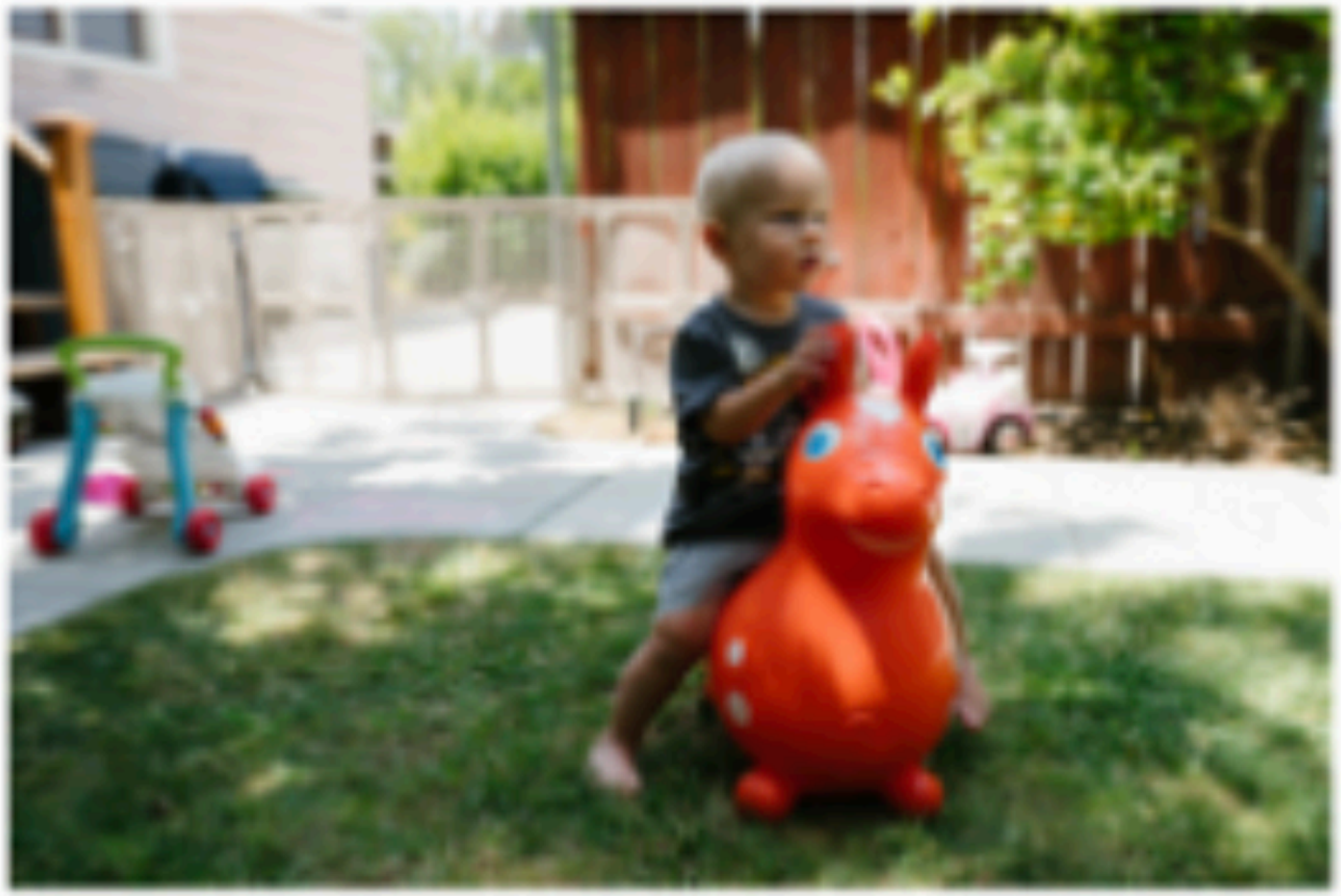
Very inefficient!

The solution: convolution

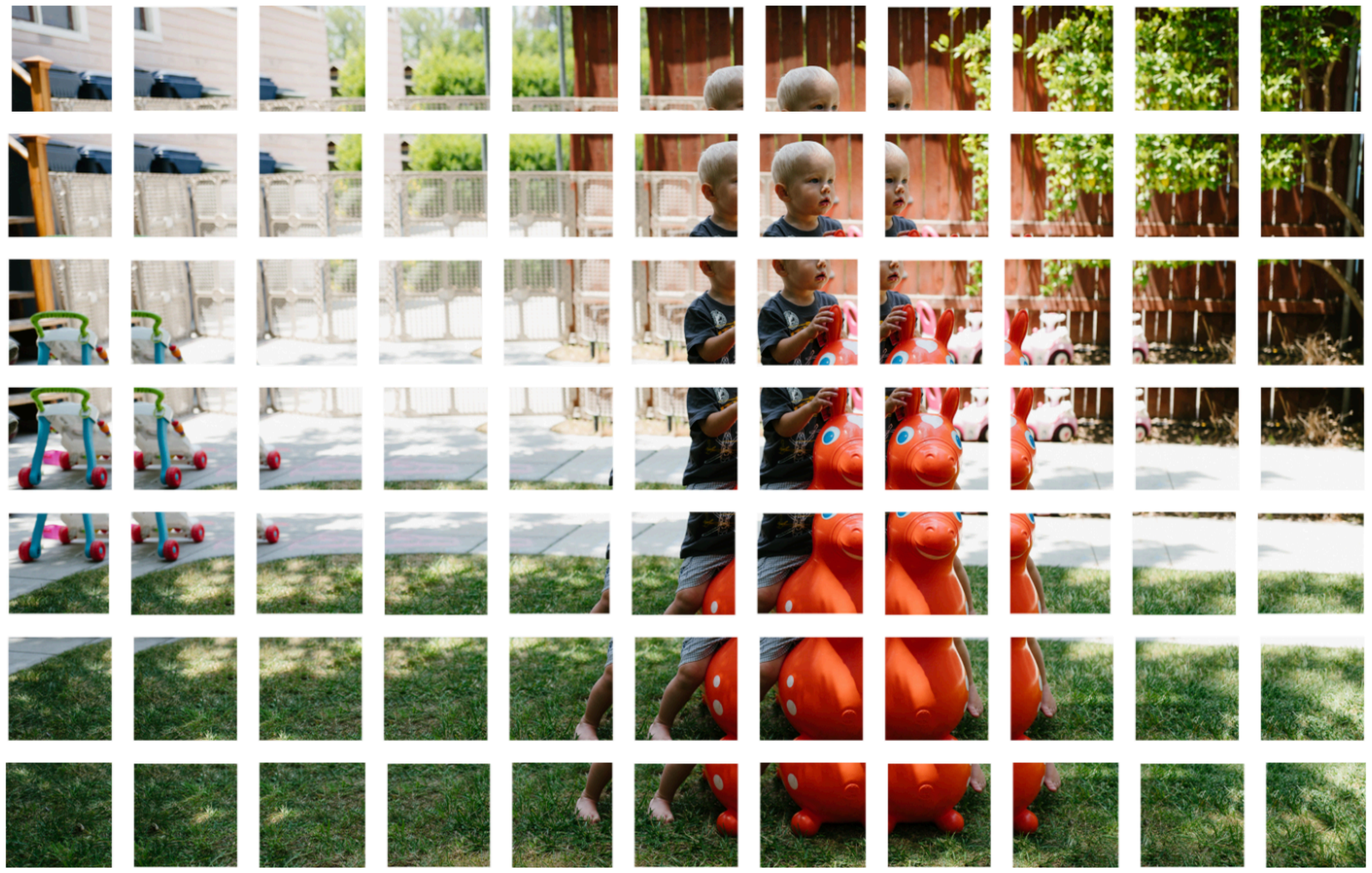
For now, our neural network thinks that an “8” in a different part of the image is an entirely different thing. It doesn’t understand that moving an object around in the picture doesn’t make it something different.

We need to give our neural network understanding of **translation invariance**. This can be done using a process called **convolution**.

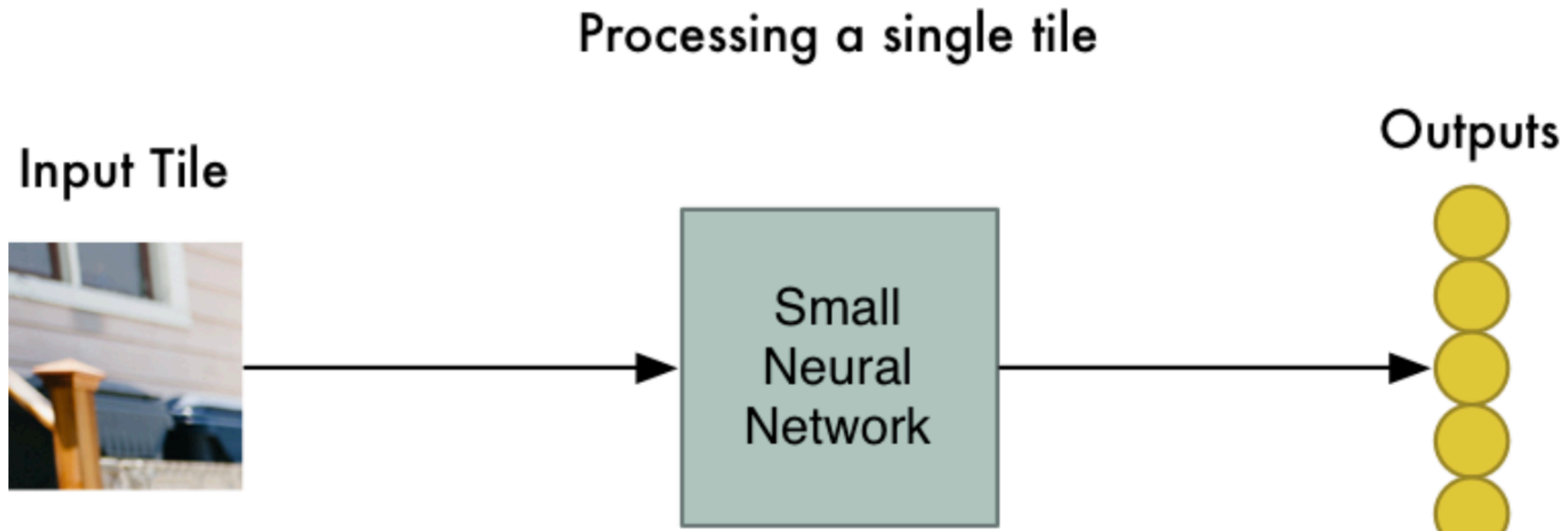
Example: Find a face in the following picture



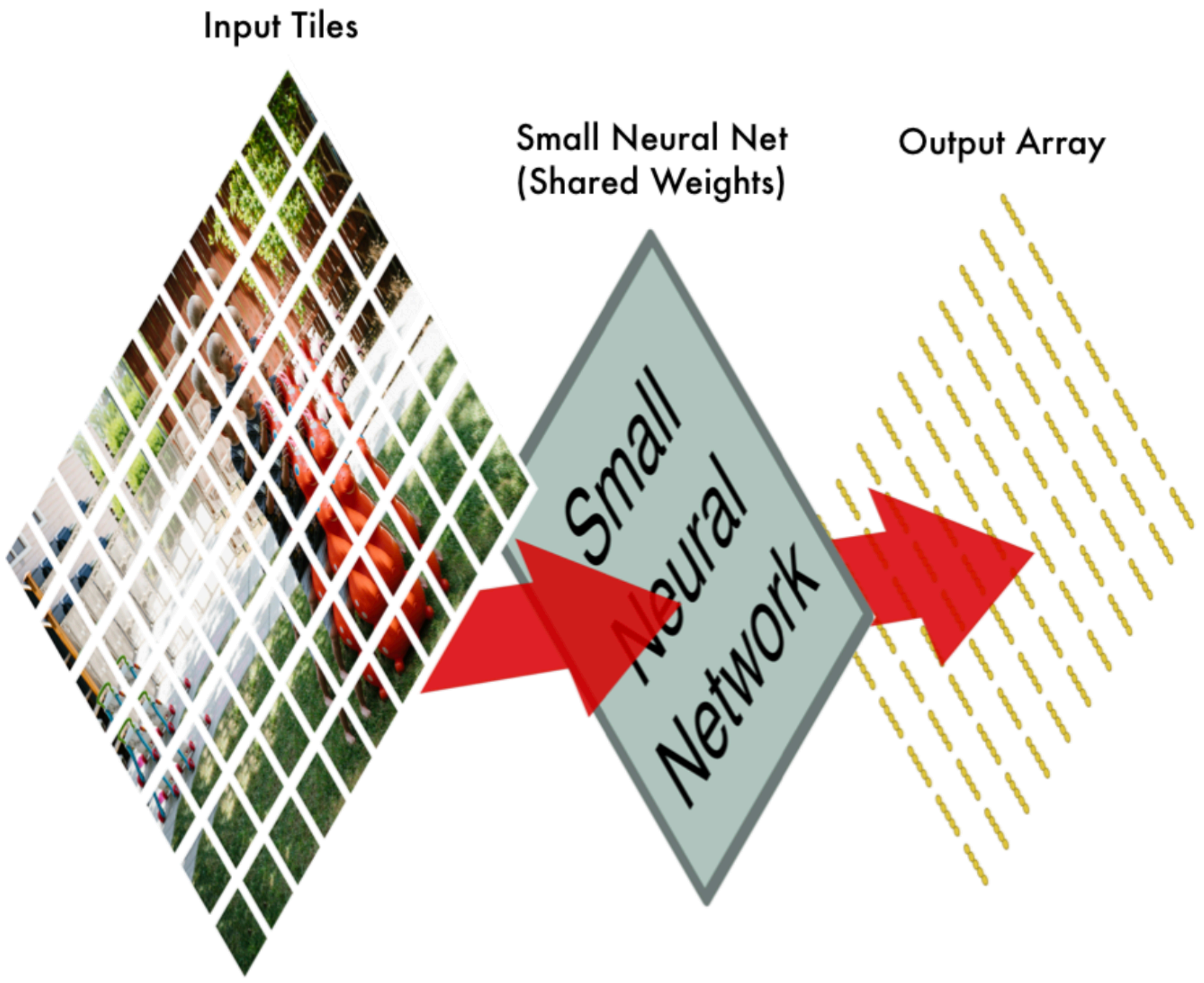
Step 1: Break the image into overlapping image tiles



Step 2: Feed each image tile into a small neural network. We'll keep the same neural network weights for every single tile in the same original image. In other words, we are treating every image tile equally.

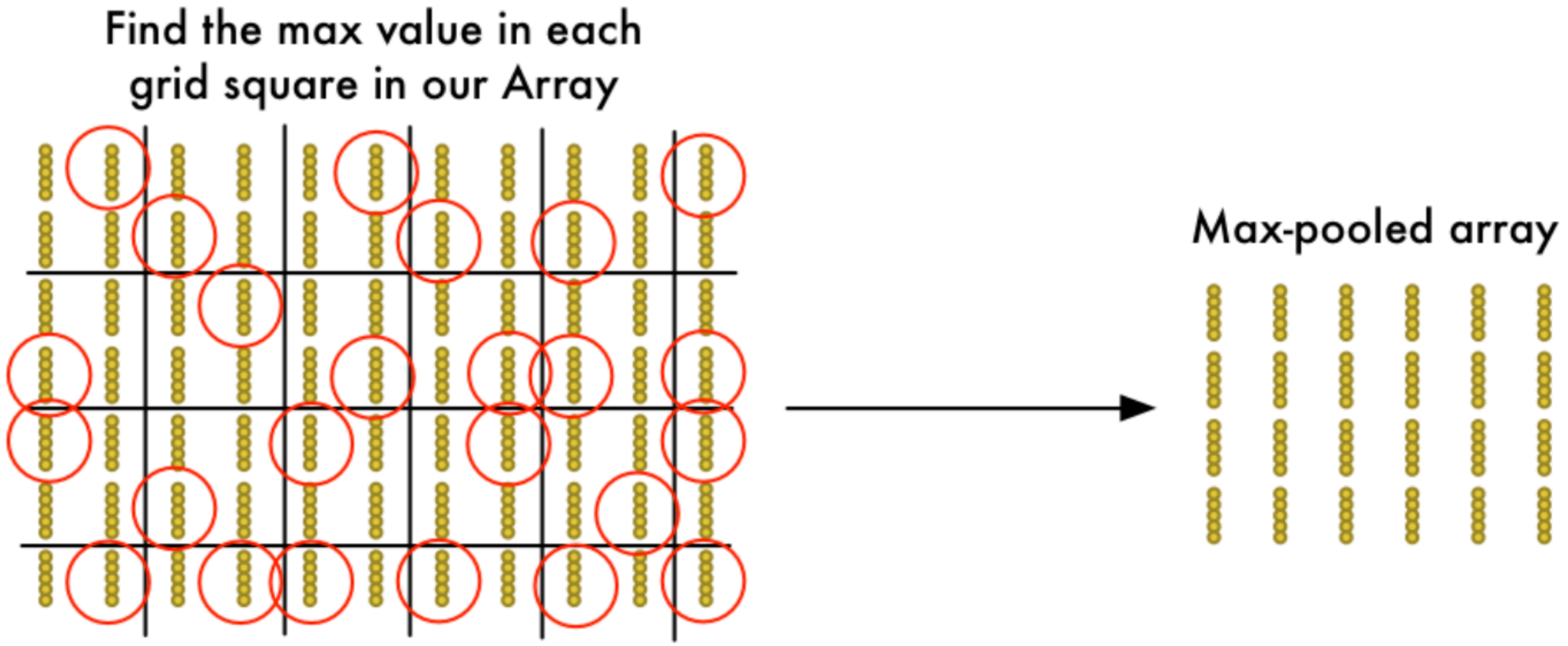


Step 3: Save the results from each tile into a new array in the same arrangement as the original image.

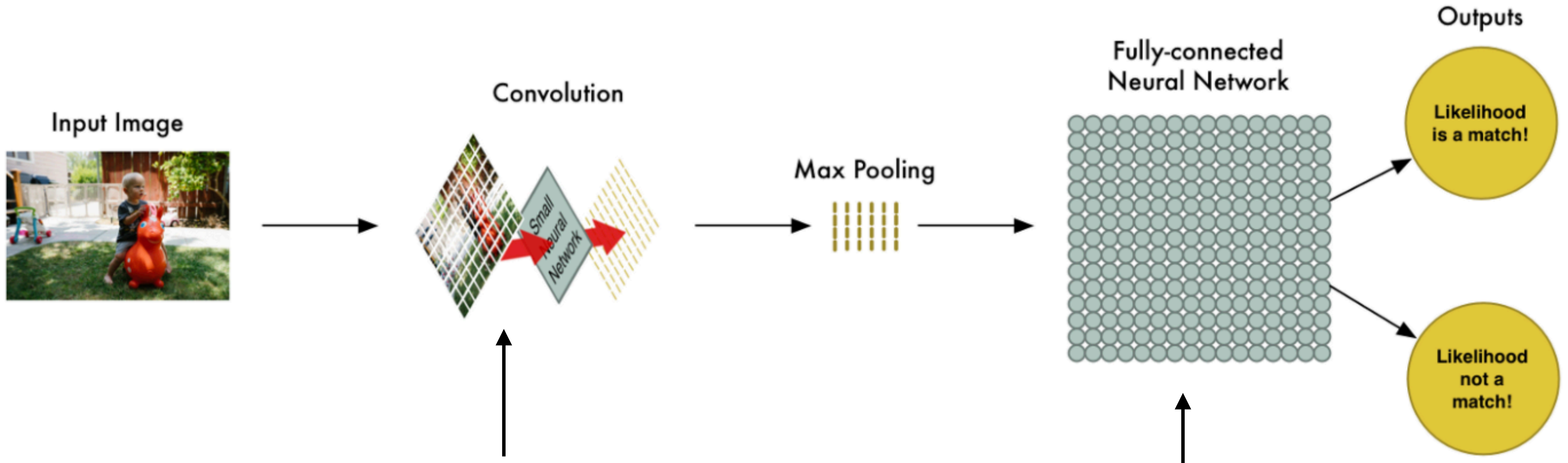


Step 4: Downsampling: The result of Step 3 was an array that maps out which parts of the original image are the most interesting. But that array is still pretty big. To reduce its size, we *downsample* it using an algorithm called *max pooling*.

We look at each 2x2 square of the array and keep the biggest number.

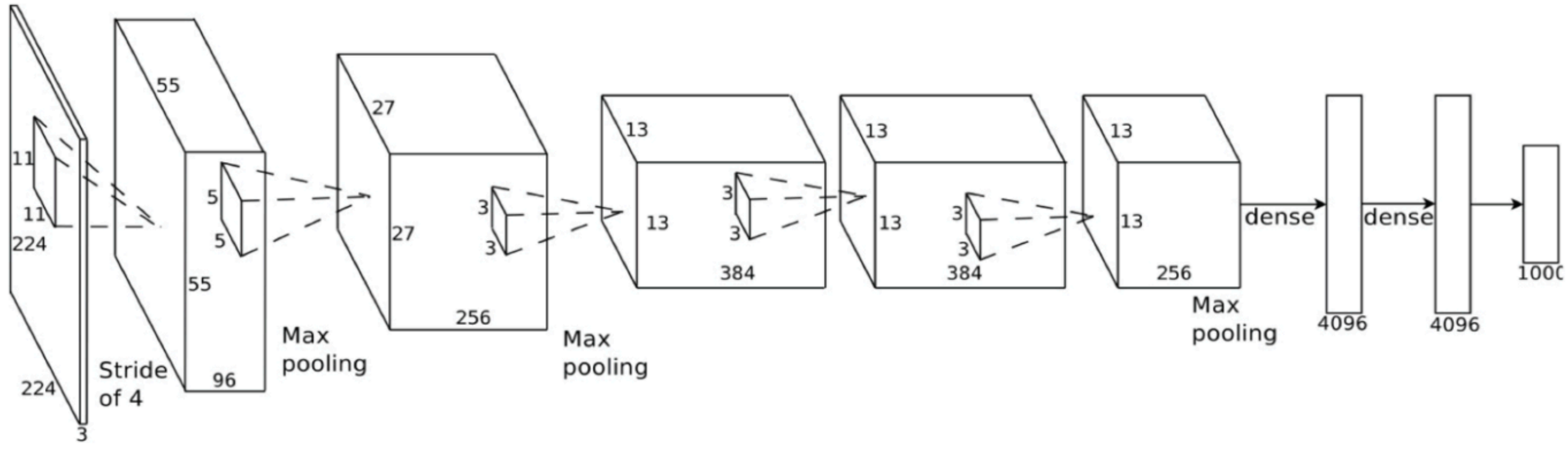


Final step: Make a prediction. We can use that small array as input into *another neural network*. This final neural network will decide if the image is or isn't a match.



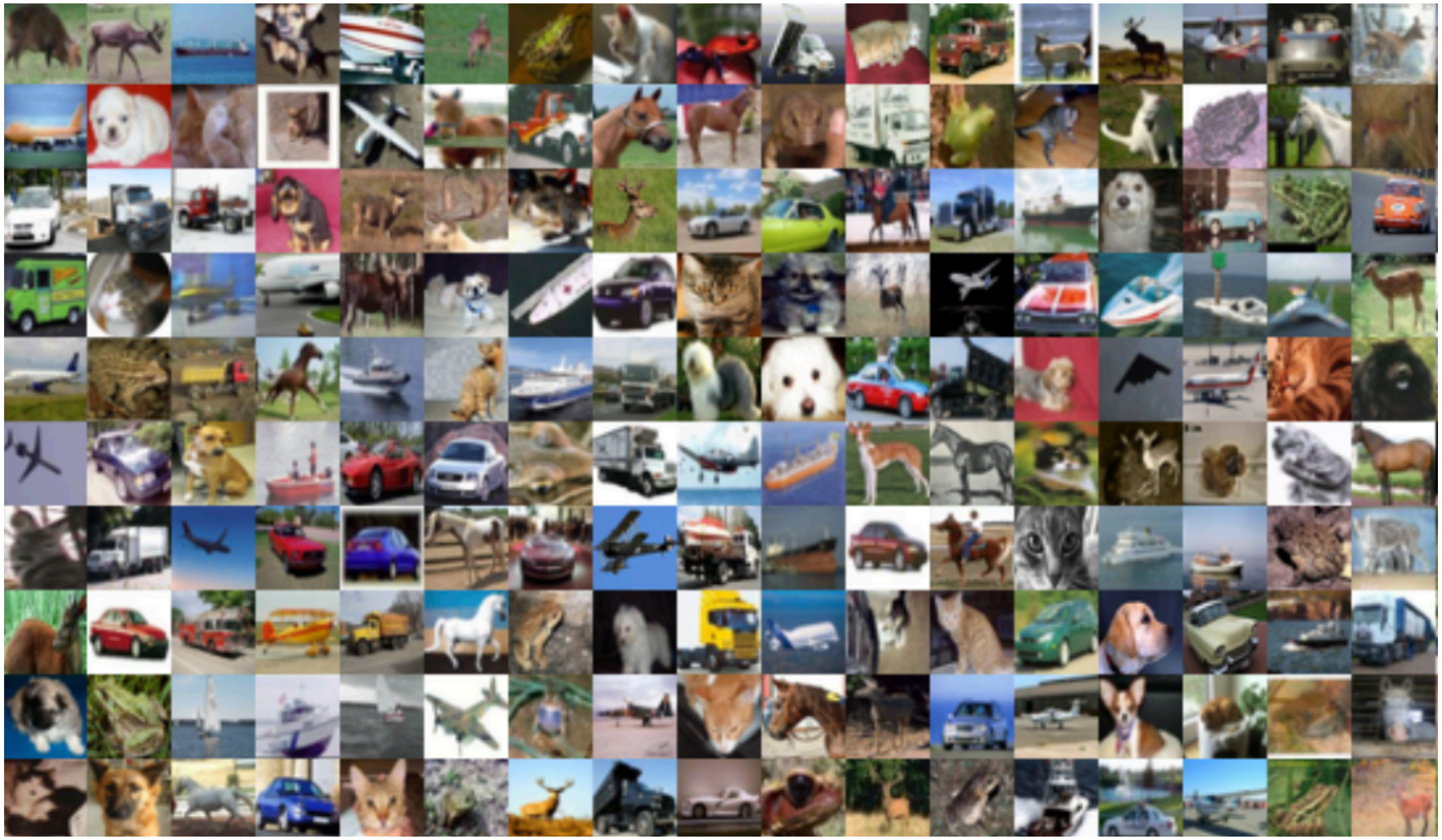
Convolutional layer

Layers from last time



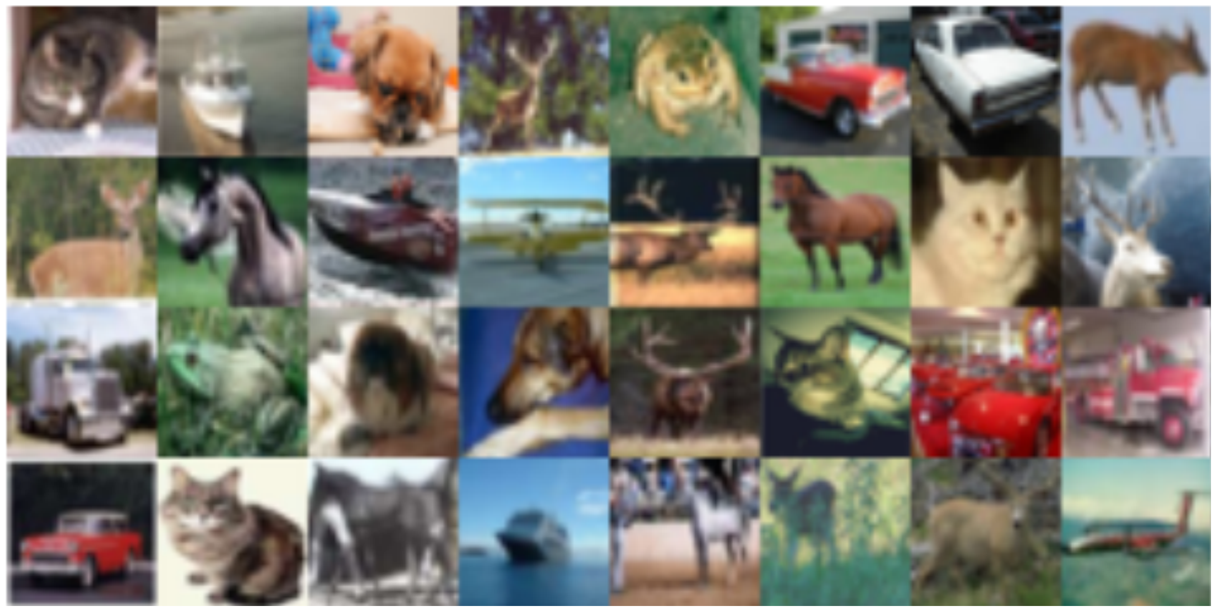
When solving problems in the real world, these steps can be combined and stacked as many times as we want to address tasks of different complexity.

Building a bird classifier



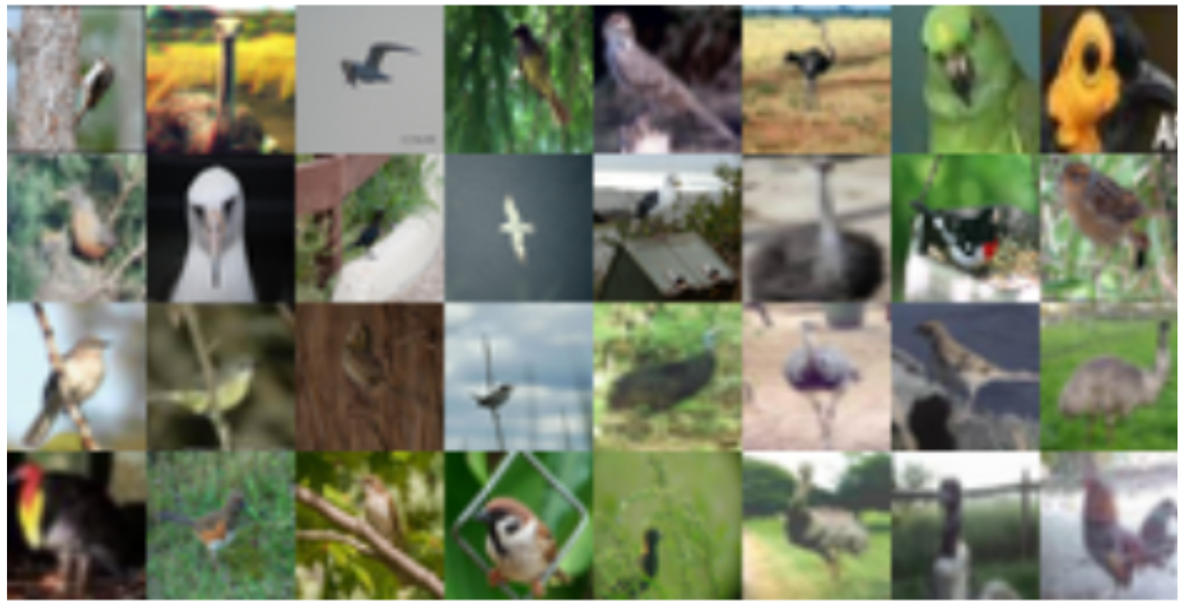
True positives

True negatives



False positives

False negatives



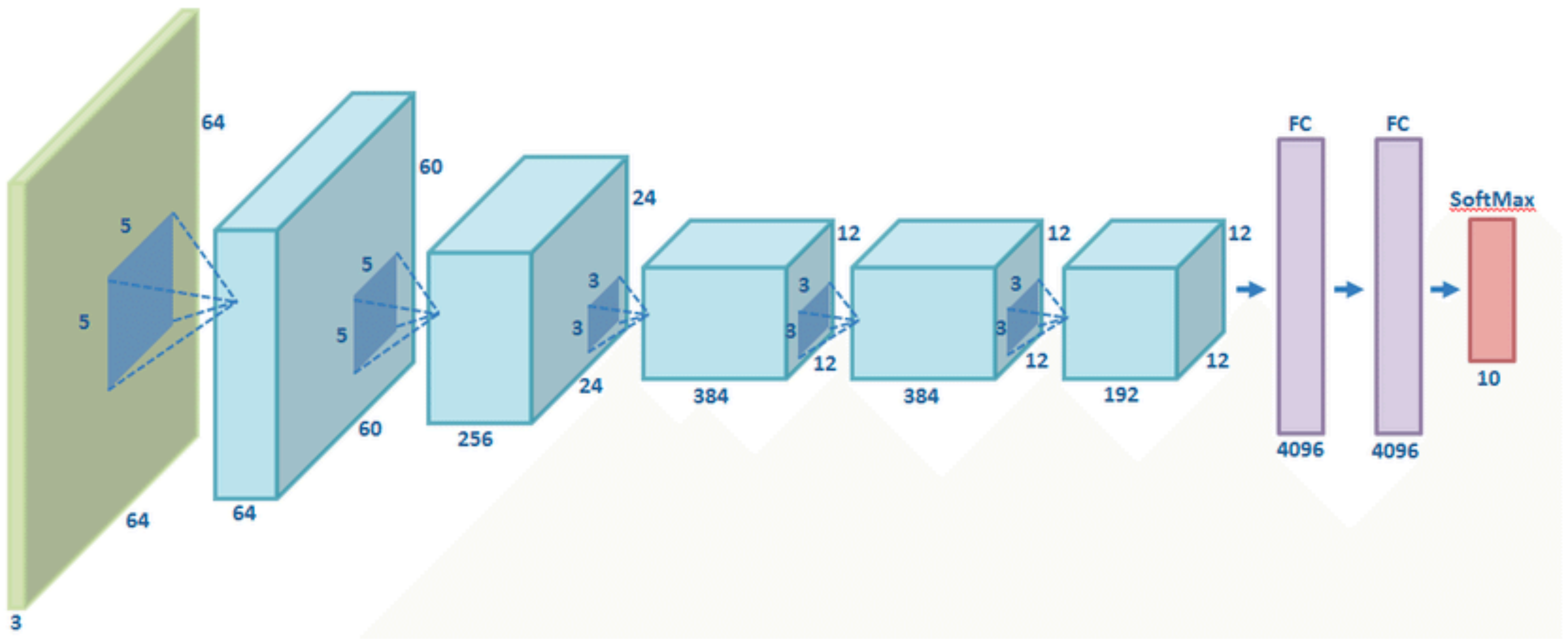
Results for 15,000 Validation Images

(6000 images are birds, 9000 images are not birds)

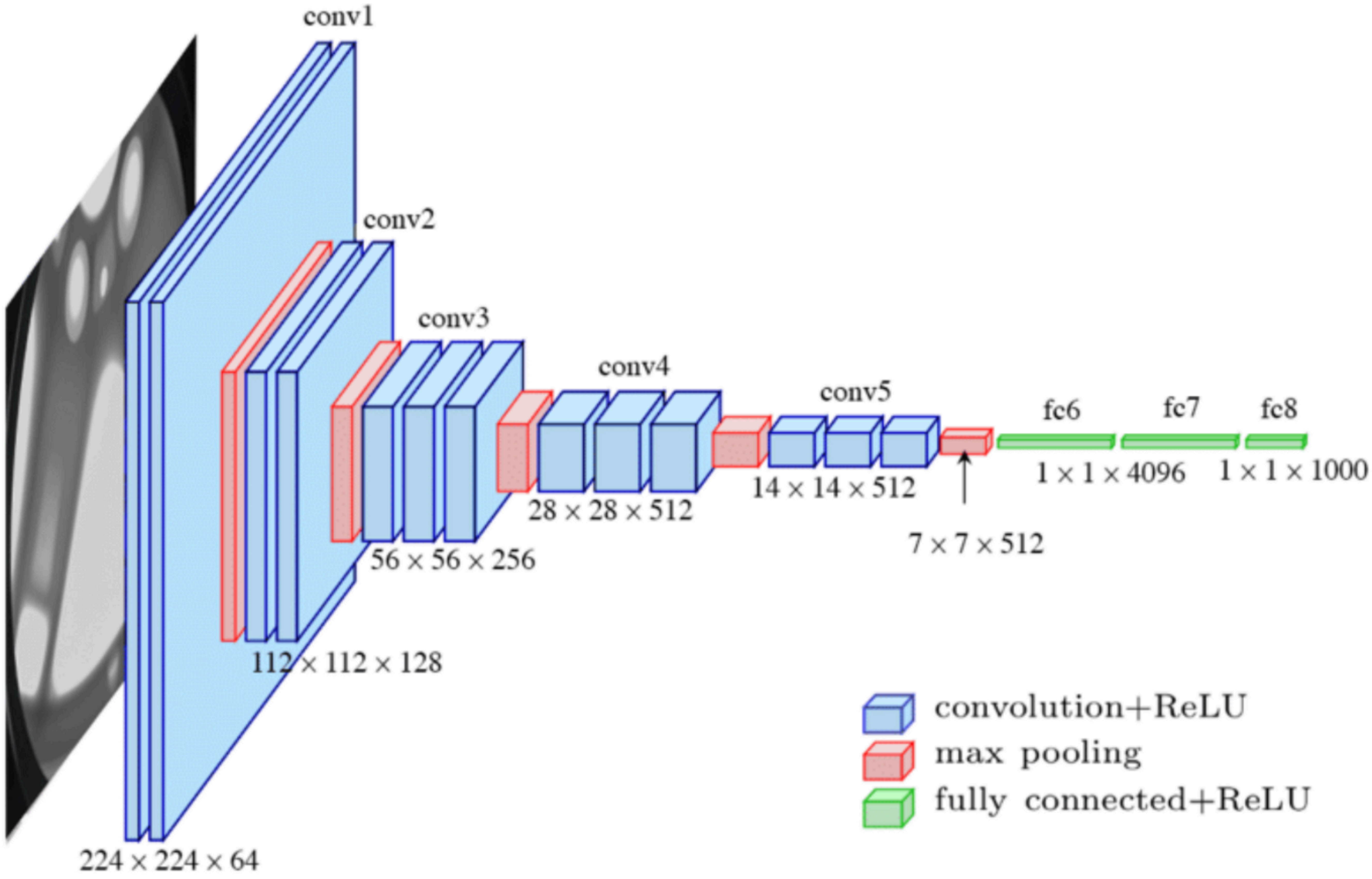
	Predicted 'bird'	Predicted 'not a bird'
Bird	**5,450** *True Positves*	**550** *False Negatives*
Not a Bird	**162** *False Positives*	**8,838** *True Negatives*

Precision *If we predicted 'bird', how often was it really a bird?*	**97.11%** *(True Positives ÷ All Positive Guesses)*
Recall *What percentage of the actual birds did we find?*	**90.83%** *(True Positives ÷ Total Birds in Dataset)*

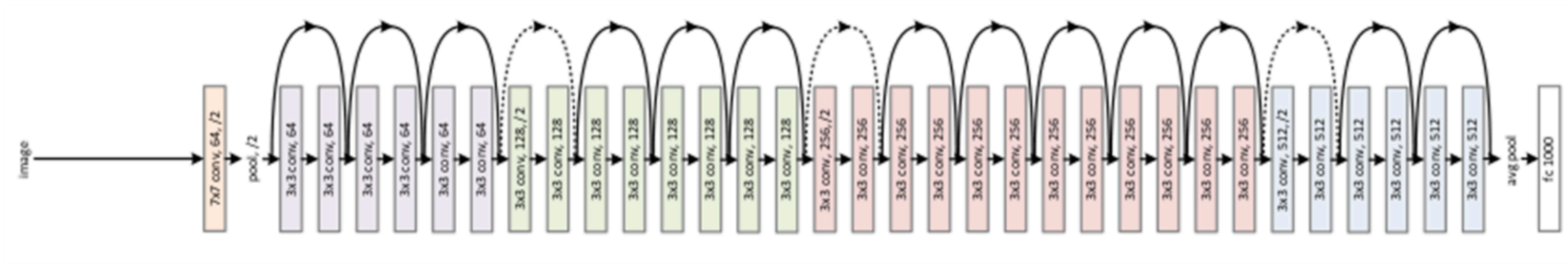
AlexNet (16% error on ImageNet)



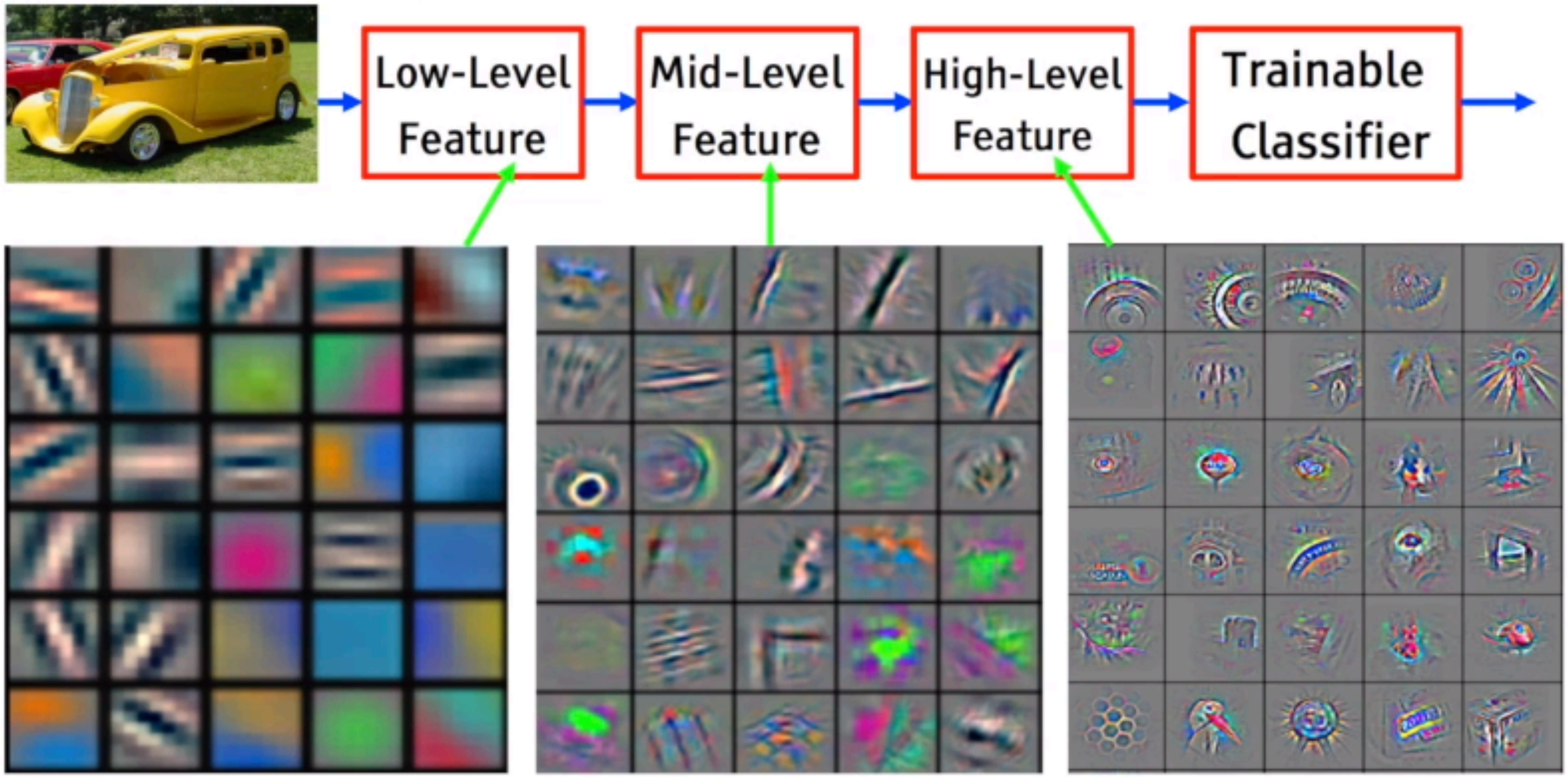
VGG16 (7.3% error on ImageNet)



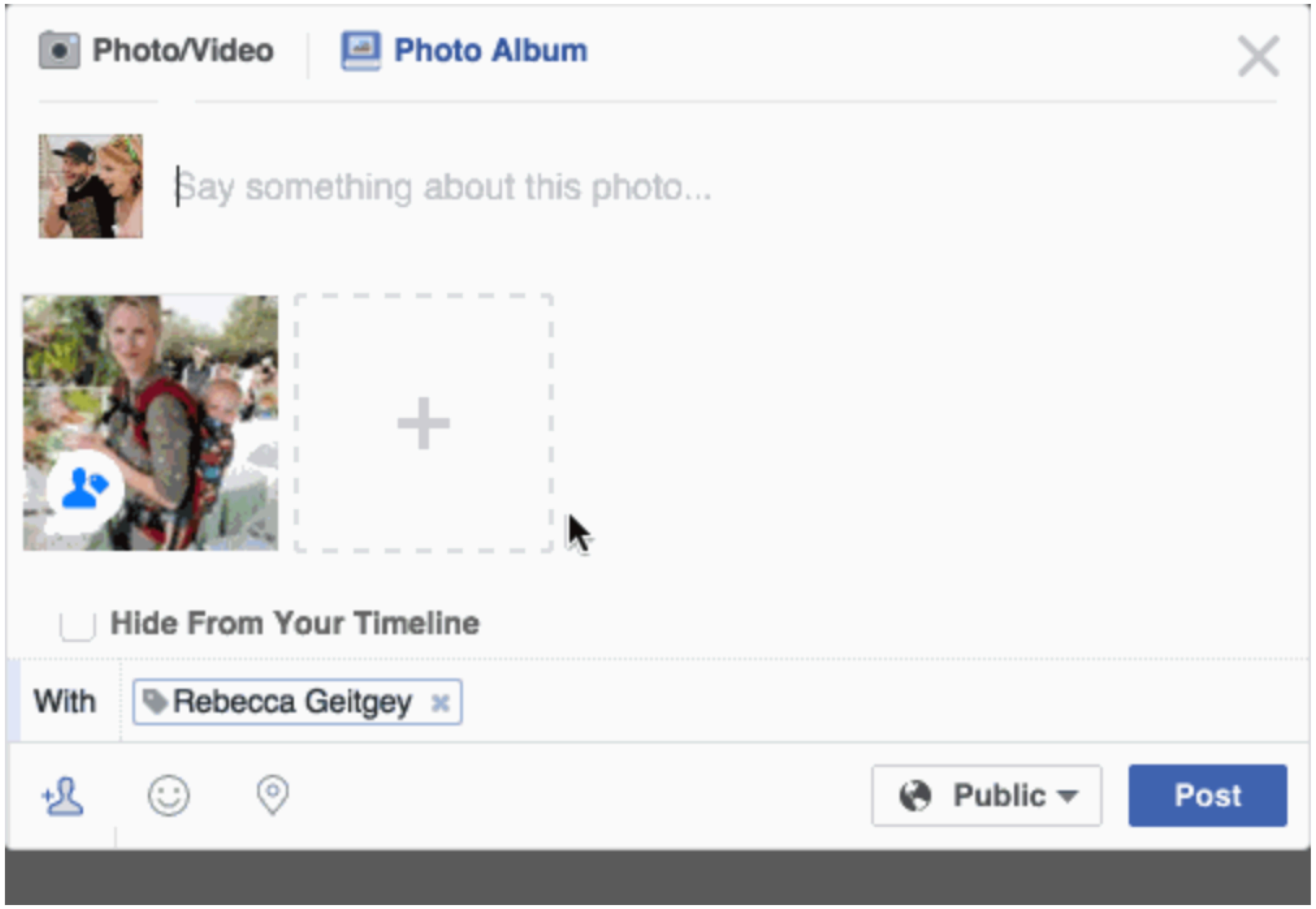
ResNet (3.6% error on ImageNet)



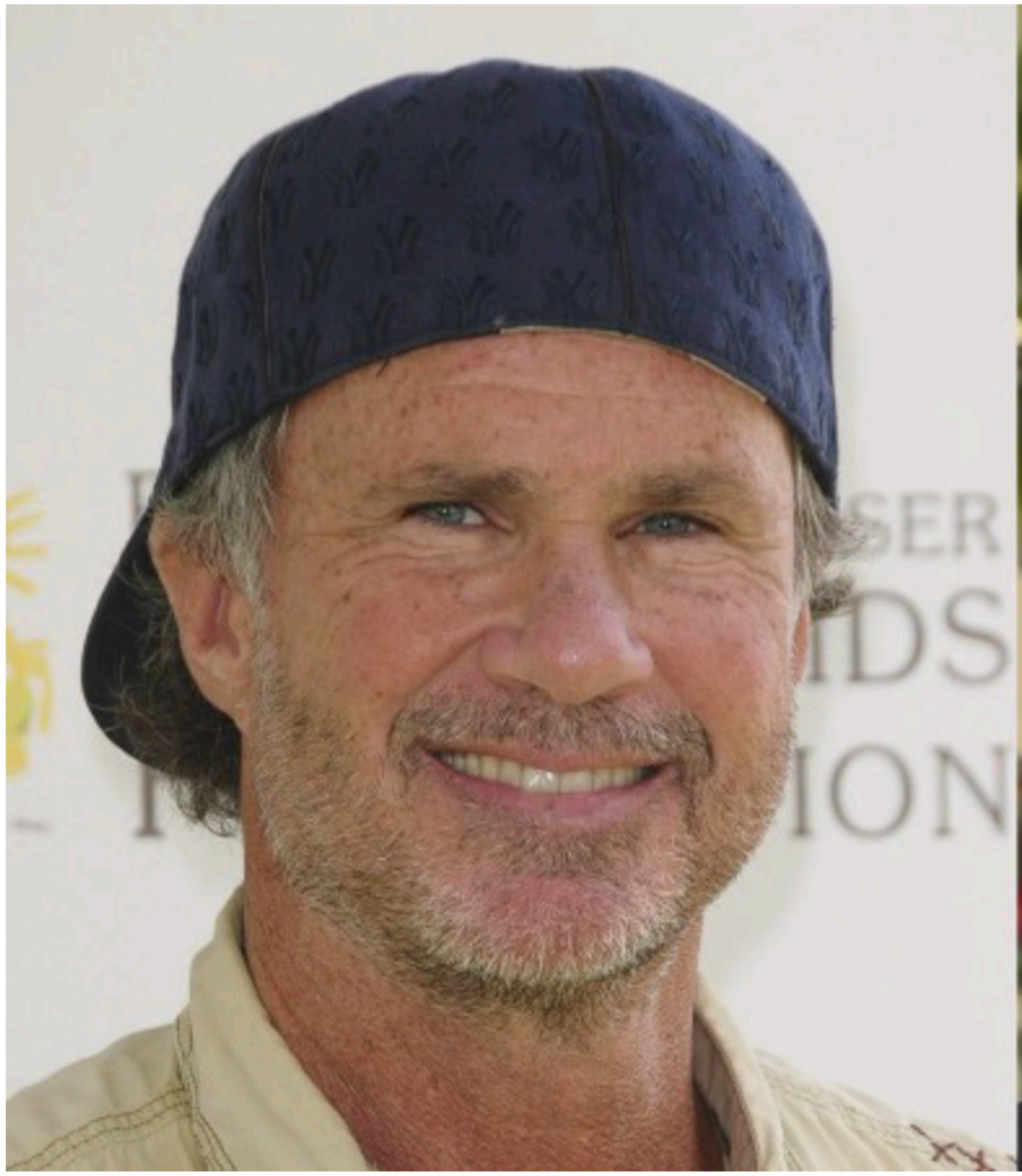
Convolutional layers extract features from an image

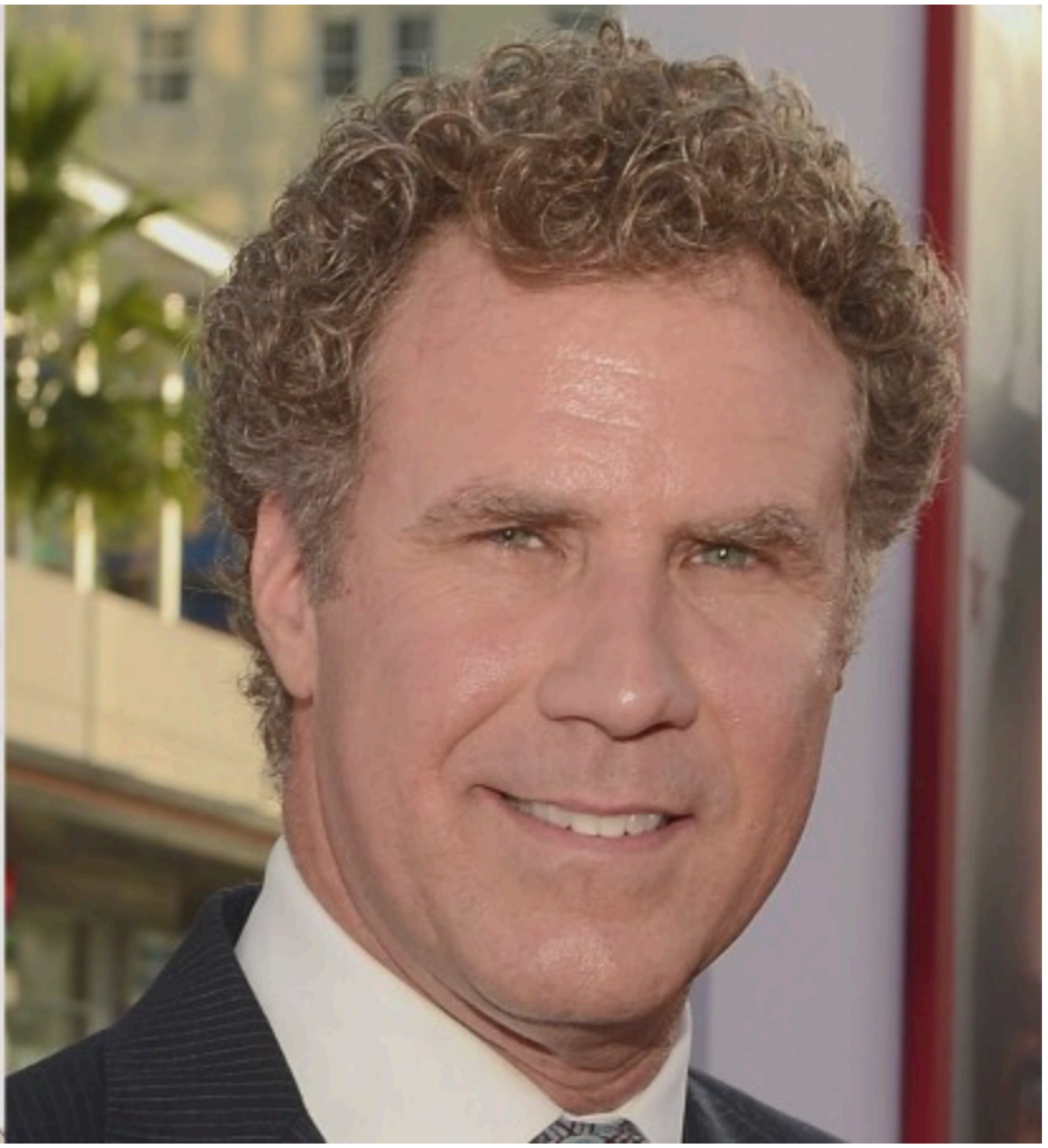


Example: Facebook automatically tags the people in your photos that you only tagged a couple of times before.



Facial recognition

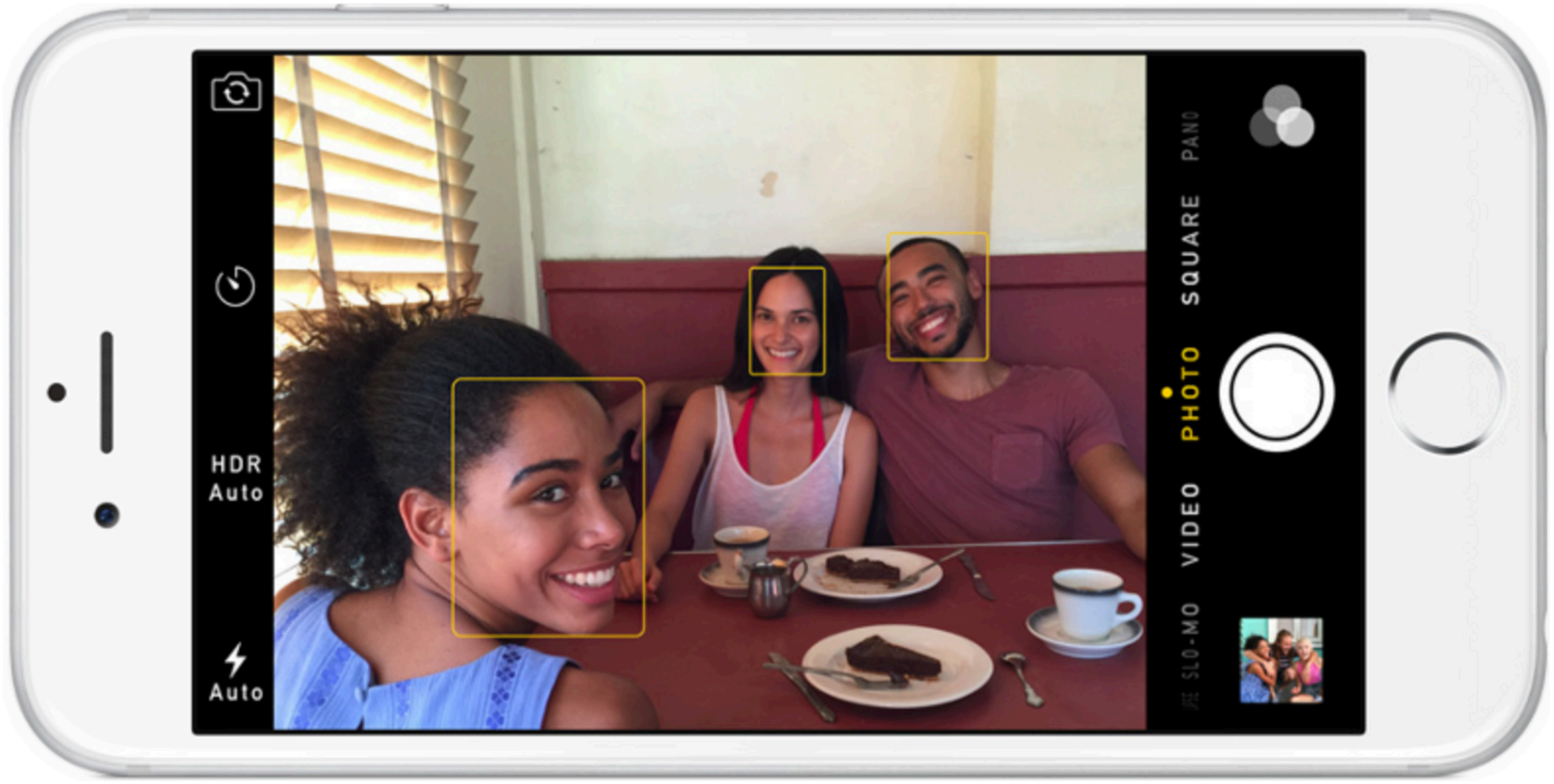

Chad Smith (a famous rock musician)


Will Ferrell (a famous actor)

Steps for face recognition

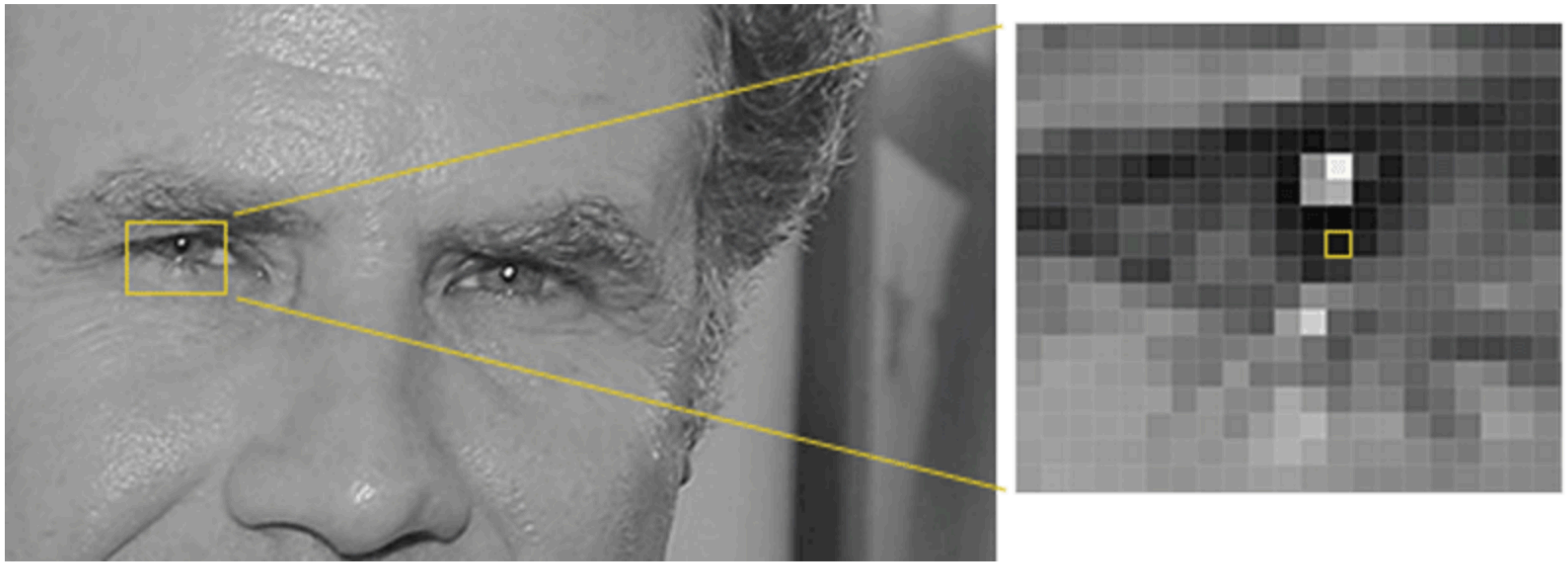
1. Look at the image and find all the faces in it.
2. Focus on each face and be able to understand that even if a face is turned in a weird direction or in bad lighting, it is still the same person.
3. Highlight the unique features of the face that can be used to distinguish it from other people - for example, how big the eyes are or how long the face is, etc.
4. Compare these unique features of a person with the features of other people you know to determine the name of the person.

Step 1: Finding all faces

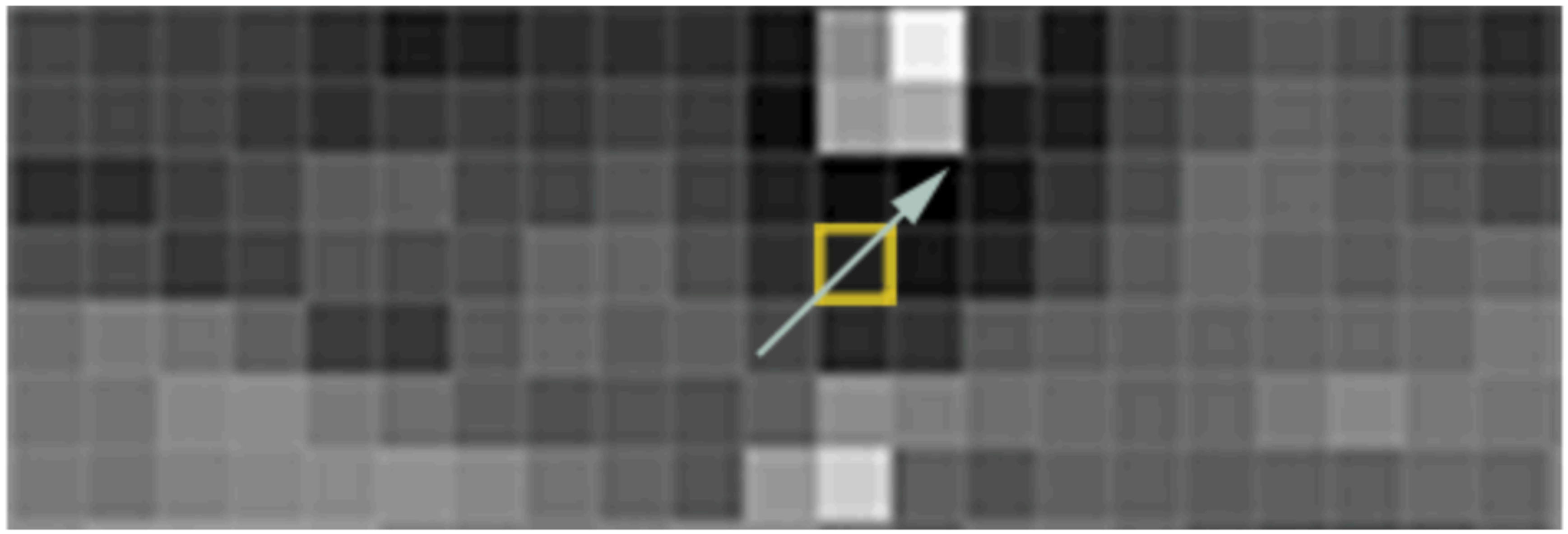


Face detection became the mainstream in the early 2000s when Paul Viola and Michael Jones invented a way to detect faces that was fast enough to work on cheap cameras.

However, there are now many more reliable solutions. We are going to use a method discovered in 2005 called histogram of oriented gradients (HOG).

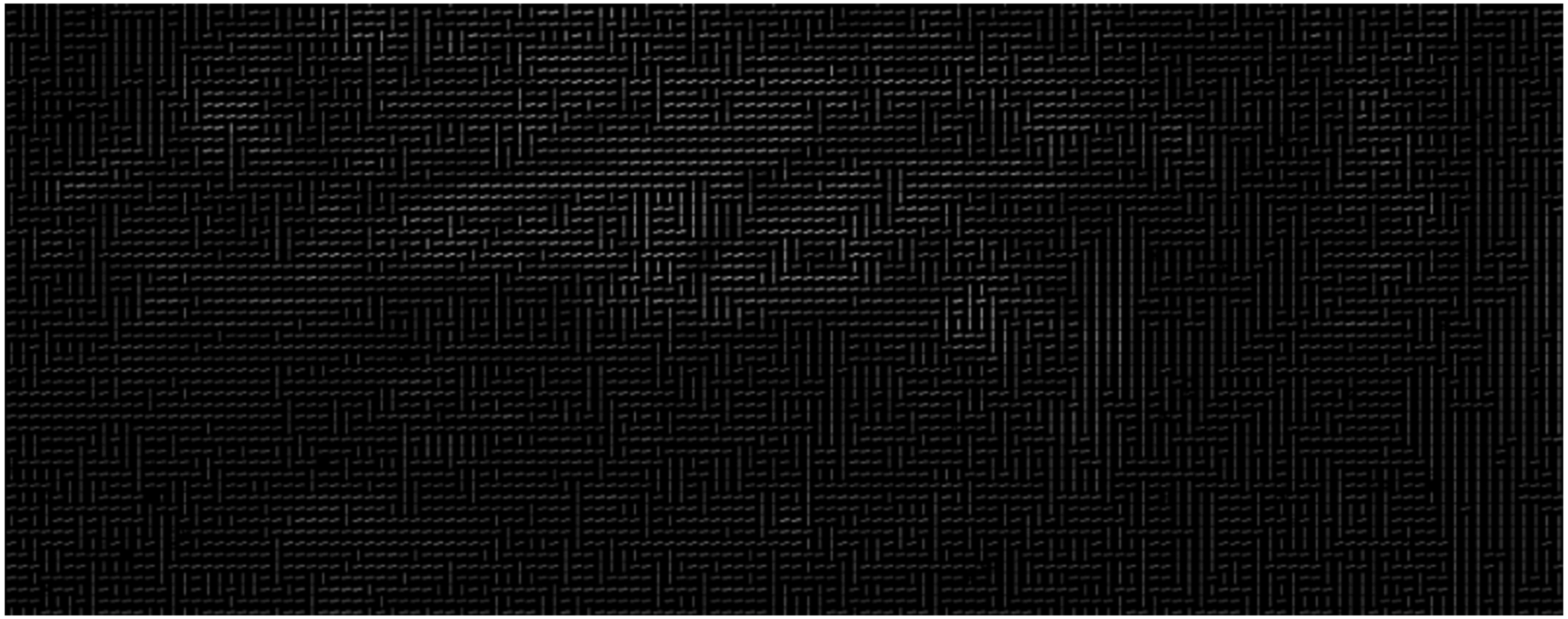


We look at each individual pixel in our image sequentially. For each individual pixel, its immediate environment should be considered.



Our goal is to highlight how dark the current pixel is in comparison with the pixels directly adjacent to it. Then draw an arrow showing the direction in which the image becomes darker.

If we repeat this process for each individual pixel in the image, then, in the end, each pixel will be replaced by an arrow. These arrows are called *gradients*, and they show the flow from light to dark throughout the image.

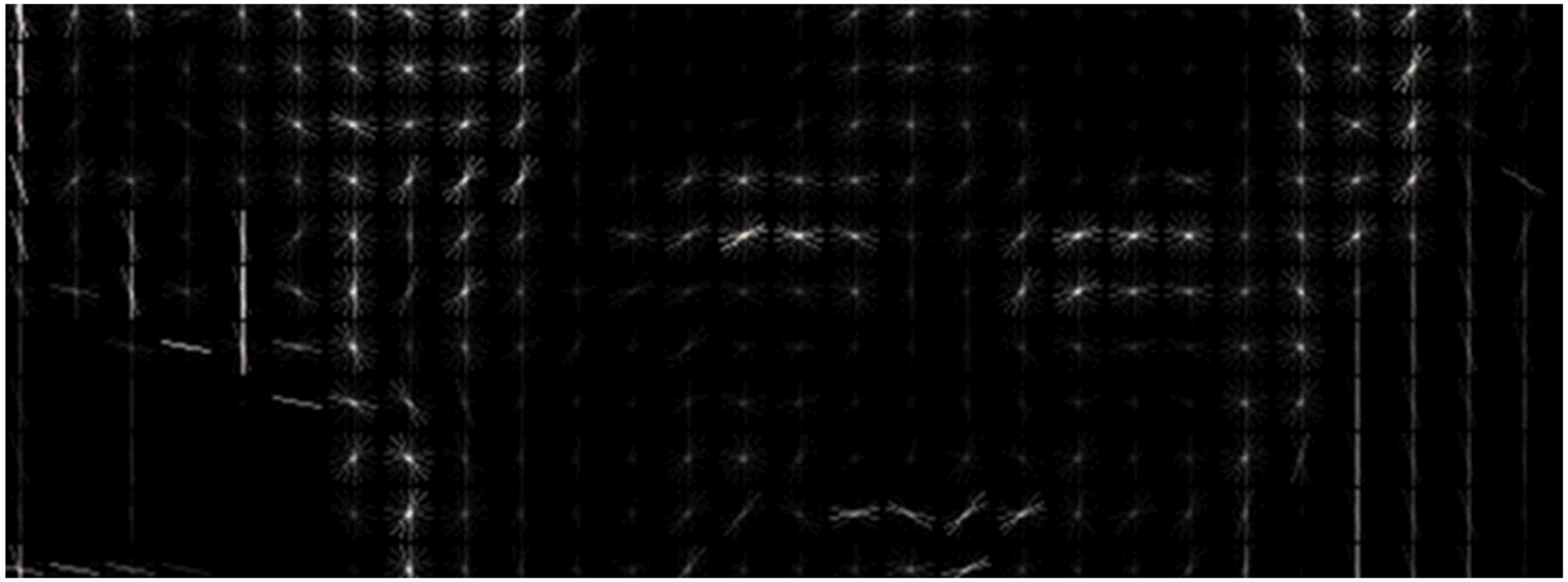


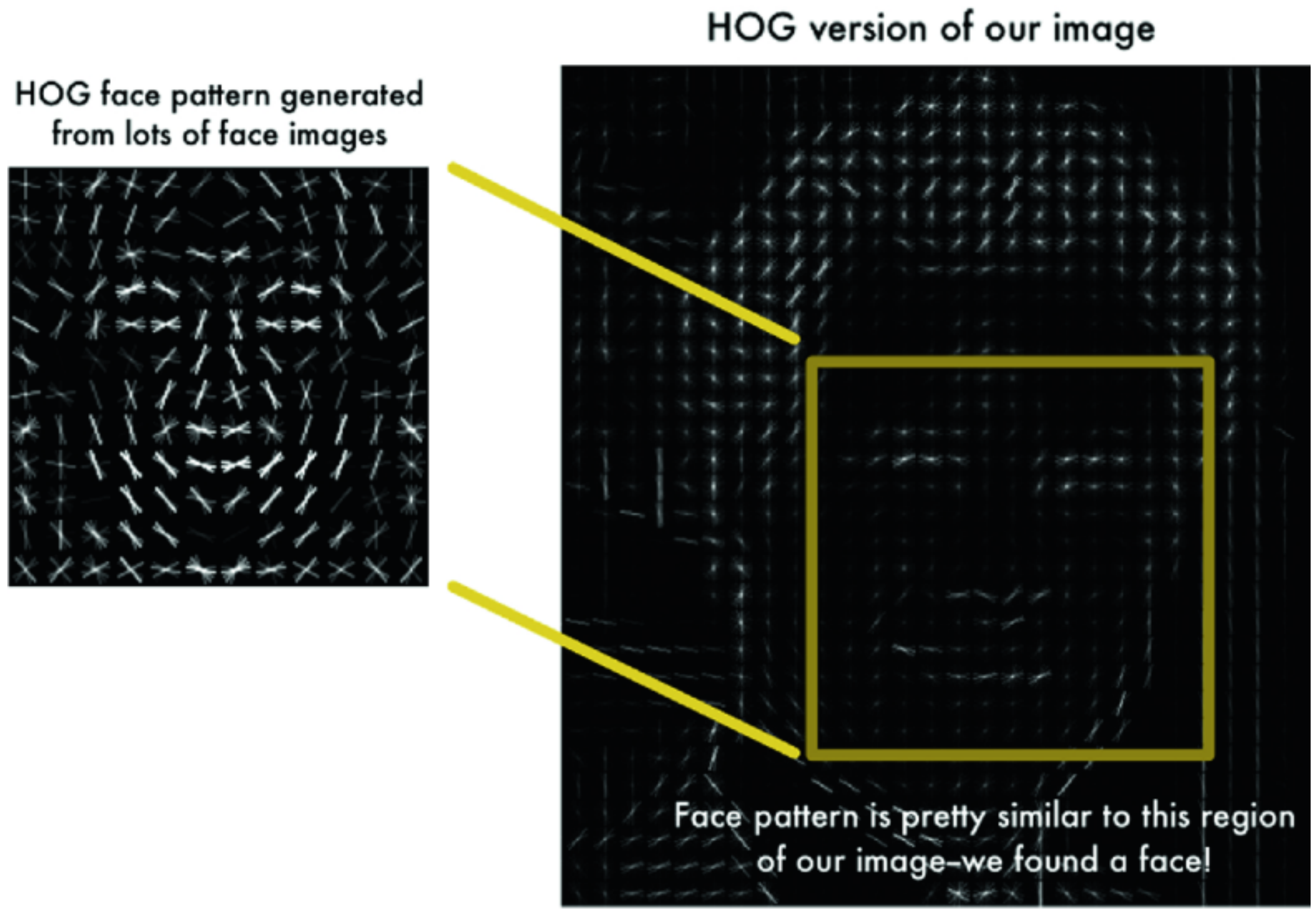
When we analyze pixels directly, dark and light images of the same person will have very different pixel intensities. But if we consider only the directional changes in brightness, then both dark and light images will have exactly the same representation.

However, preserving the gradient for each individual pixel gives us way too many details, so we need a way to keep only the basic structure of the image.

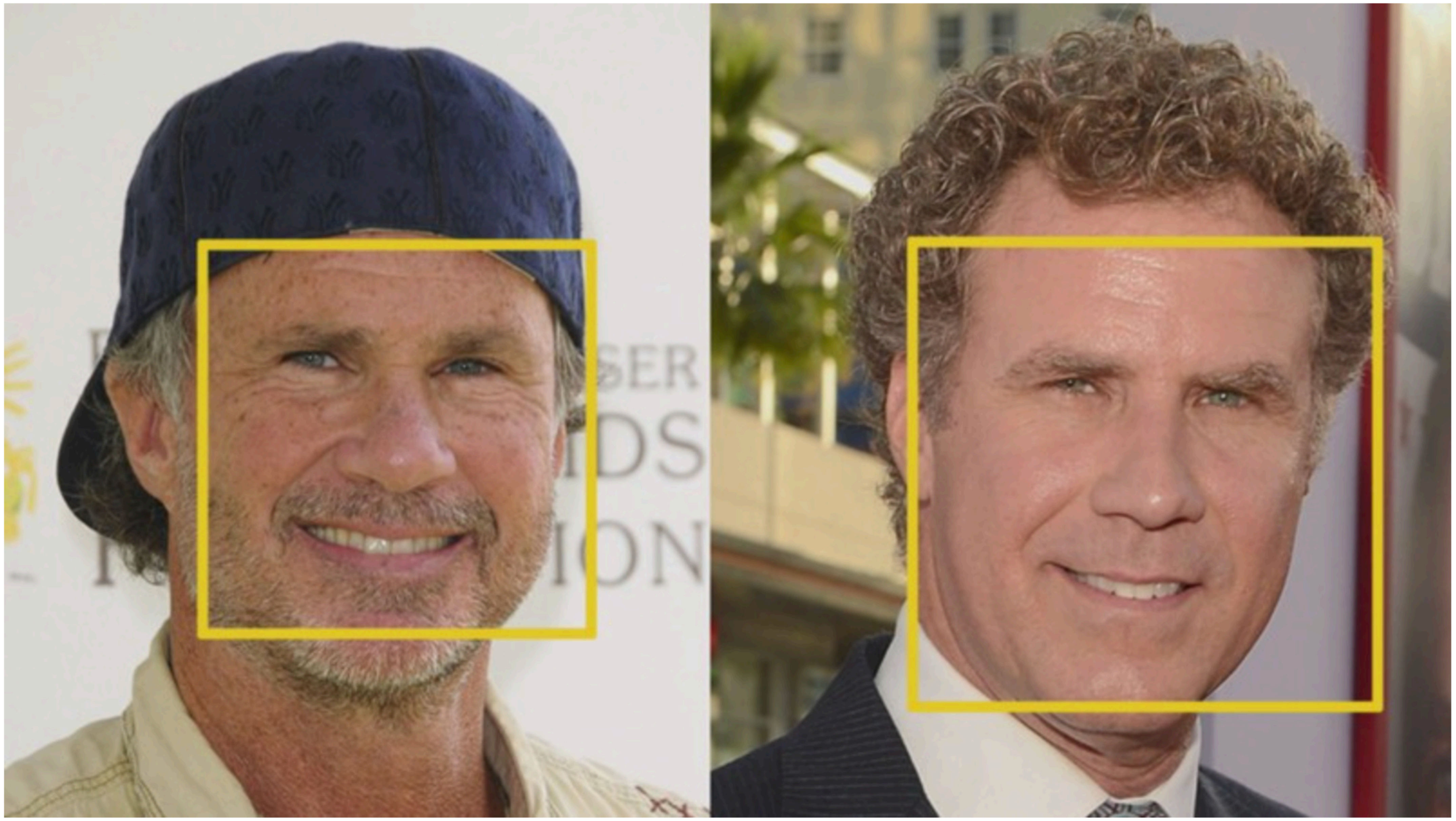
Solution

1. Break the image into small squares of 16x16 pixels in each.
2. In each square, you should calculate how many gradient arrows are shown in each main direction (i.e. how many arrows are directed up, up right, right, etc.).
3. Replace the square under consideration by an arrow with the direction prevailing in this square.

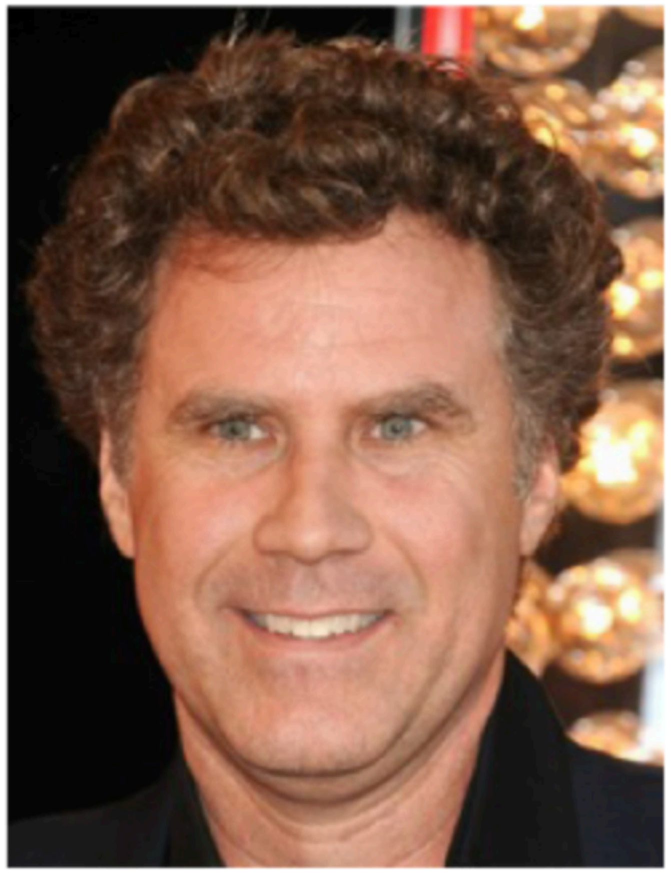


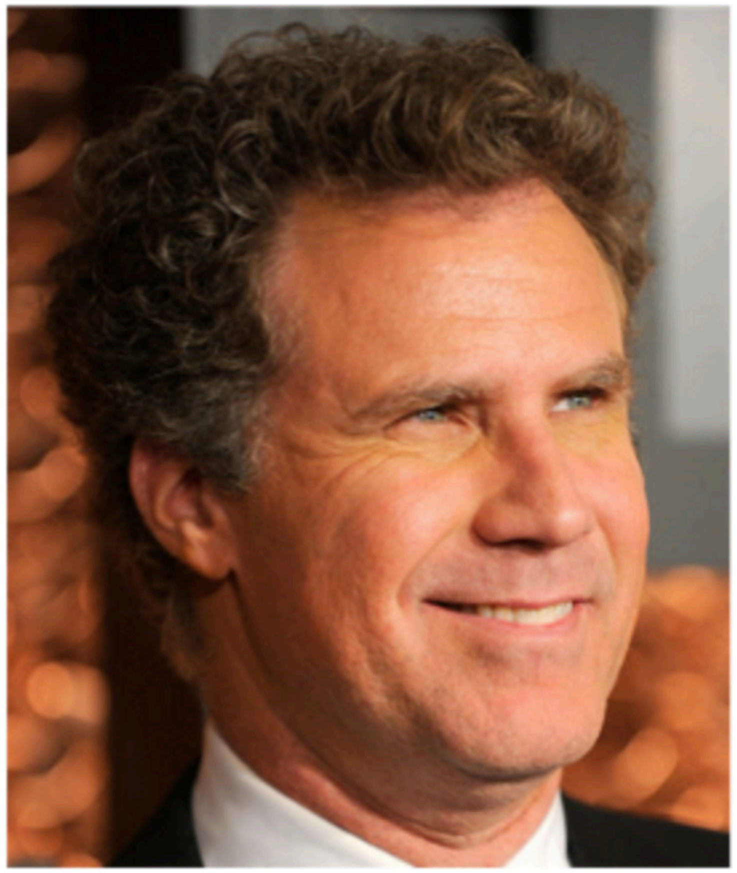

HOG version of our image
HOG face pattern generated from lots of face images
Face pattern is pretty similar to this region of our image–we found a face!

To detect faces in this HOG image, all that is required is to find a part of the image that is most similar to the known HOG structure obtained from the group of faces used for training.



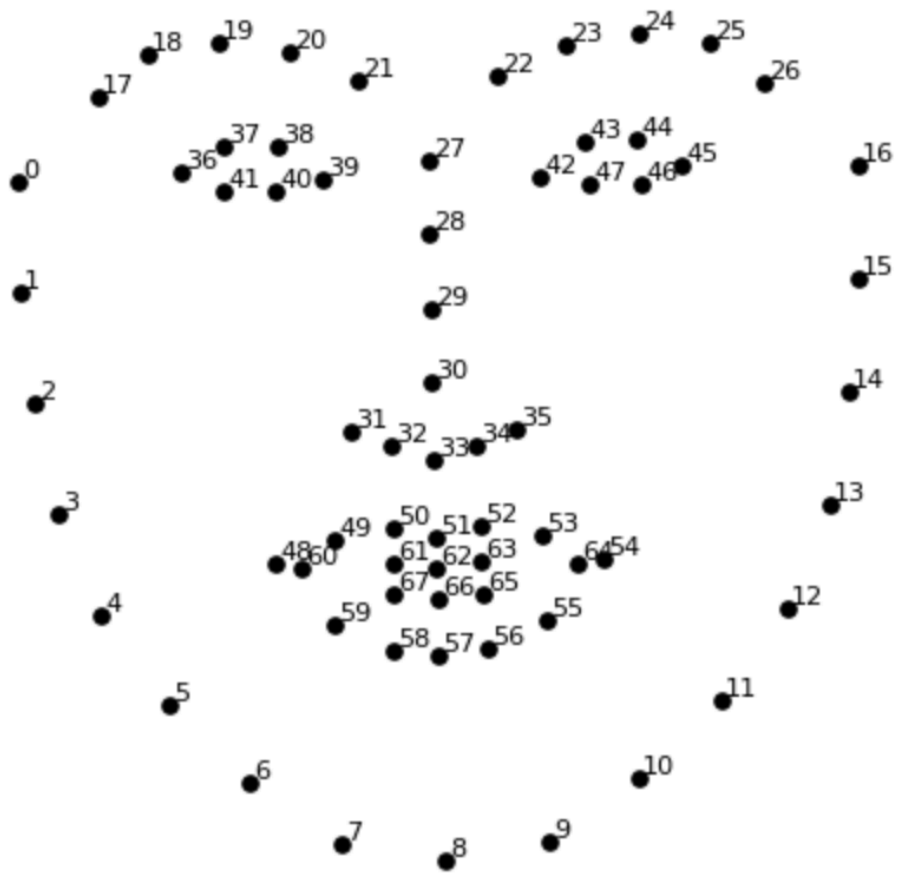
Step 2: Location and display of faces





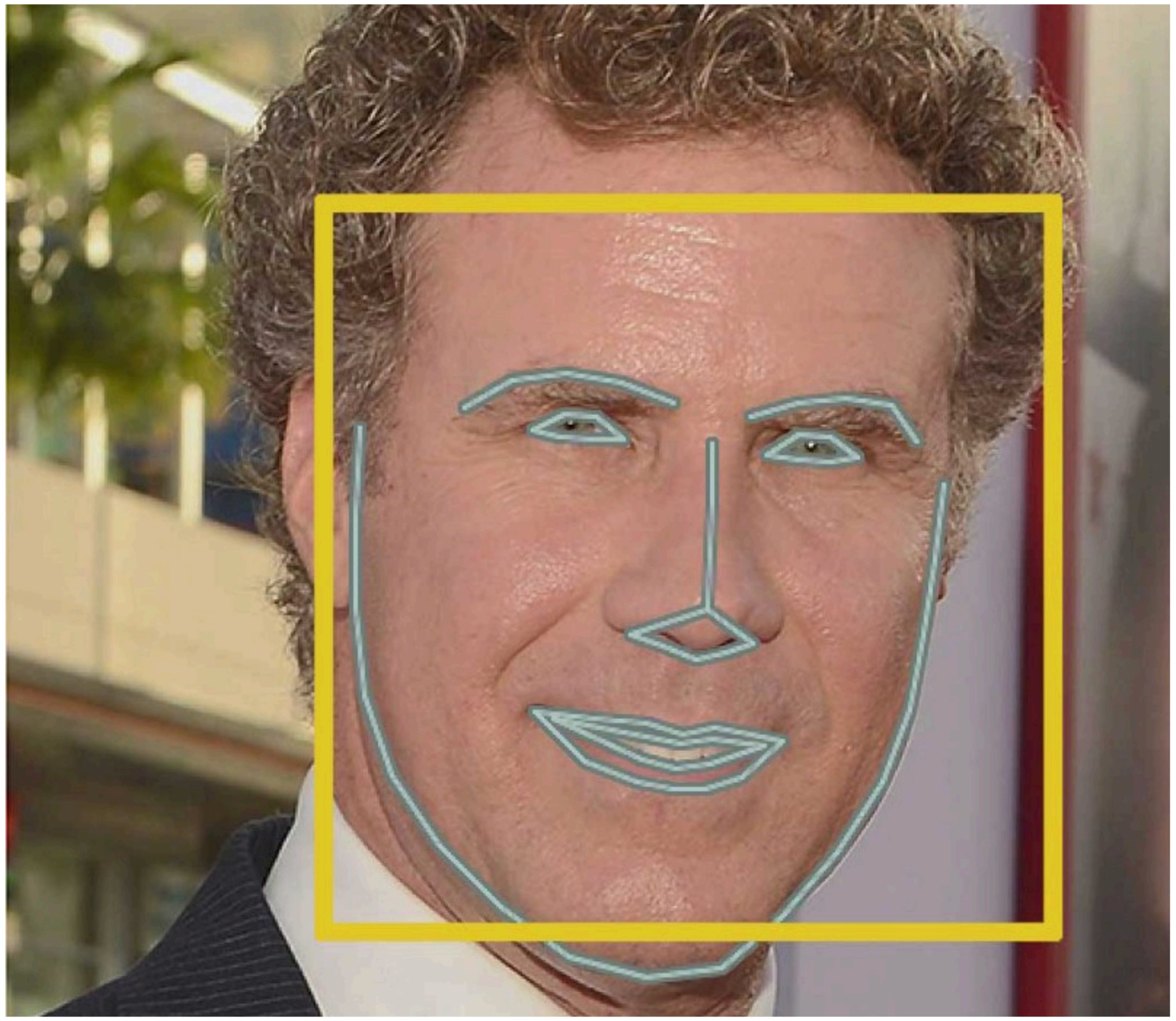
People can easily see that both images refer to the same person, but computers will consider them as faces of two different people.

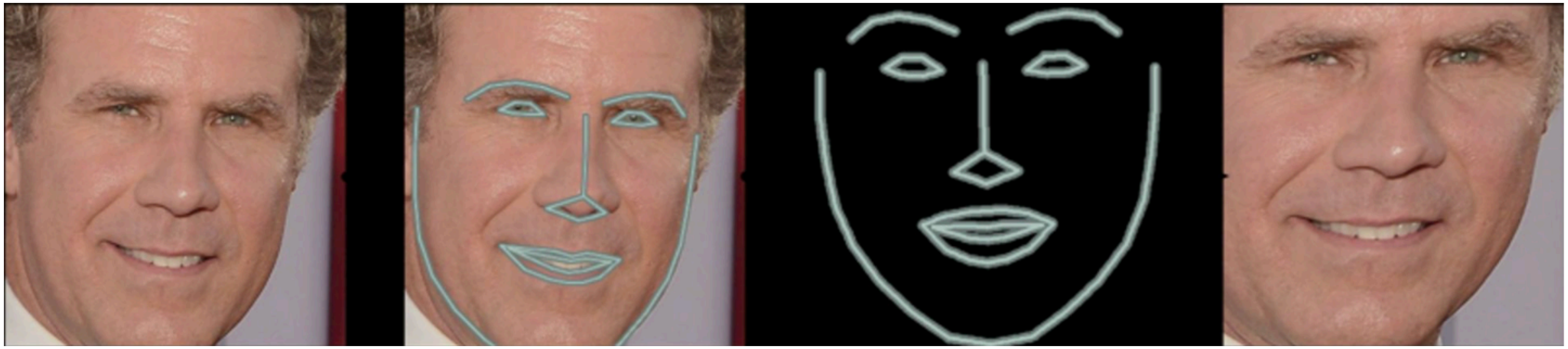
To address this issue, we will transform each image so that the eyes and lips are always in the same place on the image.



We use a face landmark estimation algorithm called “assessment of anthropometric points.”

The main idea is that 68 specific points (marks) are found on each face - the protruding part of the chin, the outer edge of each eye, the inner edge of each eyebrow, etc.





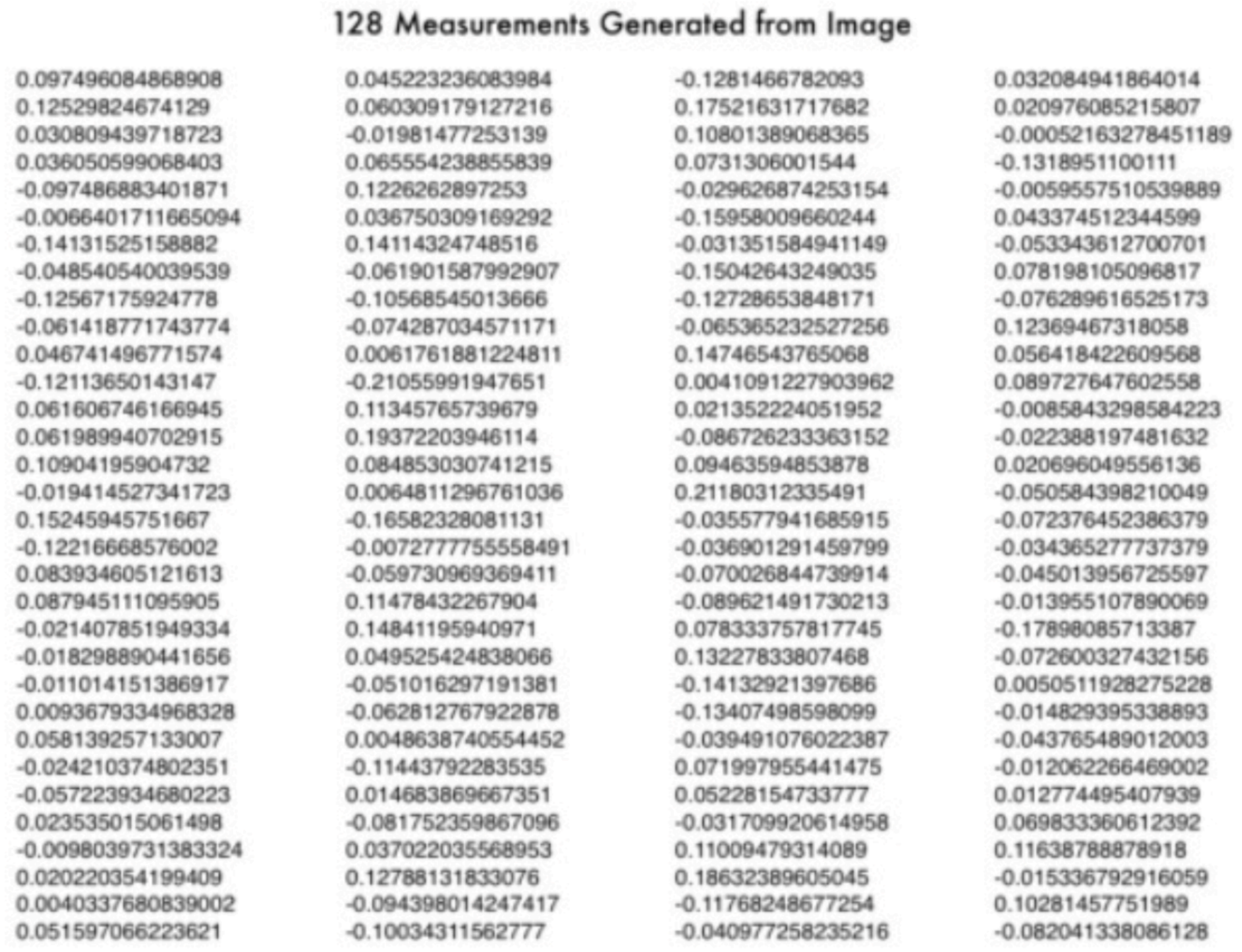
Now that we know where the eyes and mouth are, we'll just rotate, scale and move the image so that the eyes and mouth are centered as best as possible.

Steps for face recognition

1. Look at the image and find all the faces in it.
2. Focus on each face and be able to understand that even if a face is turned in a weird direction or in bad lighting, it is still the same person.
3. Highlight the unique features of the face that can be used to distinguish it from other people - for example, how big the eyes are or how long the face is, etc.
4. Compare these unique features of a person with the features of other people you know to determine the name of the person.

Step 3: Encoding Faces

Input Image

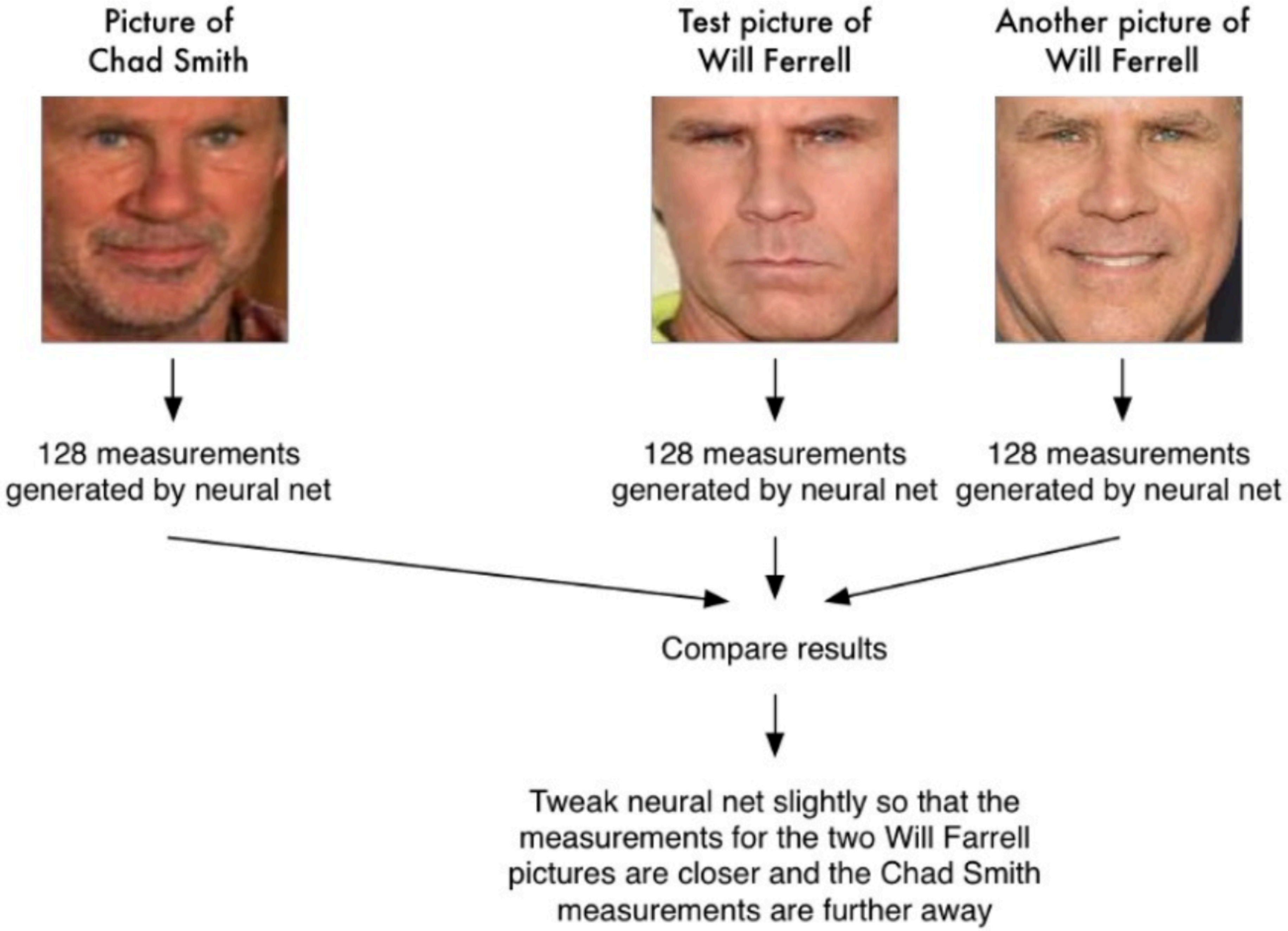


128 Measurements Generated from Image

0.097496084868908	0.045223236083984	-0.1281466782093	0.032084941864014
0.12529824674129	0.060309179127216	0.17521631717682	0.020976085215807
0.030809439718723	-0.01981477253139	0.10801389068365	-0.00052163278451189
0.036050599068403	0.065554238855839	0.0731306001544	-0.1318951100111
-0.097486883401871	0.1226262897253	-0.029626874253154	-0.0059557510539889
-0.0066401711665094	0.036750309169292	-0.15958009660244	0.043374512344599
-0.14131525158882	0.14114324748516	-0.031351584941149	-0.053343612700701
-0.048540540039539	-0.061901587992907	-0.15042643249035	0.078198105096817
-0.12567175924778	-0.10568545013666	-0.12728653848171	-0.076289616525173
-0.061418771743774	-0.074287034571171	-0.065365232527256	0.12369467318058
0.046741496771574	0.0061761881224811	0.14746543765068	0.056418422609568
-0.12113650143147	-0.21055991947651	0.0041091227903962	0.089727647602558
0.061606746166945	0.11345765739679	0.021352224051952	-0.0085843298584223
0.061989940702915	0.19372203946114	-0.086726233363152	-0.022388197481632
0.10904195904732	0.084853030741215	0.09463594853878	0.020696049556136
-0.019414527341723	0.0064811296761036	0.21180312335491	-0.050584398210049
0.15245945751667	-0.16582328081131	-0.035577941685915	-0.072376452386379
-0.12216668576002	-0.0072777755558491	-0.036901291459799	-0.034365277737379
0.083934605121613	-0.059730969369411	-0.070026844739914	-0.045013956725597
0.087945111095905	0.11478432267904	-0.089621491730213	-0.013955107890069
-0.021407851949334	0.14841195940971	0.078333757817745	-0.17898085713387
-0.018298890441656	0.049525424838066	0.13227833807468	-0.072600327432156
-0.011014151386917	-0.051016297191381	-0.14132921397686	0.0050511928275228
0.0093679334968328	-0.062812767922878	-0.13407498598099	-0.014829395338893
0.058139257133007	0.0048638740554452	-0.039491076022387	-0.043765489012003
-0.024210374802351	-0.11443792283535	0.071997955441475	-0.012062266469002
-0.057223934680223	0.014683869667351	0.05228154733777	0.012774495407939
0.023535015061498	-0.081752359867096	-0.031709920614958	0.069833360612392
-0.0098039731383324	0.037022035568953	0.11009479314089	0.11638788878918
0.020220354199409	0.12788131833076	0.18632389605045	-0.015336792916059
0.0040337680839002	-0.094398014247417	-0.11768248677254	0.10281457751989
0.051597066223621	-0.10034311562777	-0.040977258235216	-0.082041338086128

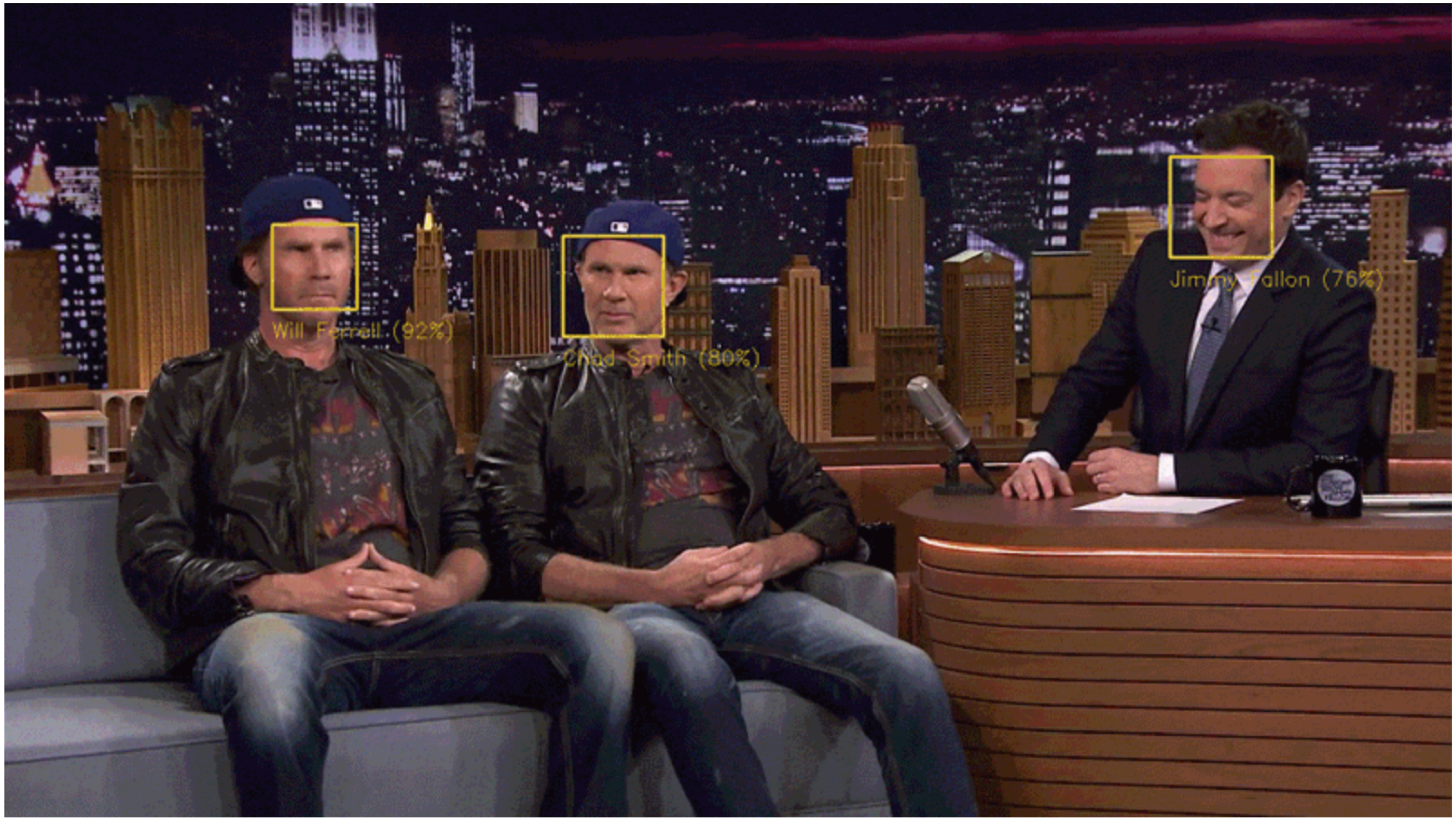
Train a neural network to generate 128 characteristics for each face.

A single "triplet" training step


Picture of Chad Smith
Test picture of Will Ferrell
Another picture of Will Ferrell
128 measurements generated by neural net
128 measurements generated by neural net
128 measurements generated by neural net
Compare results
Tweak neural net slightly so that the measurements for the two Will Farrell pictures are closer and the Chad Smith measurements are further away

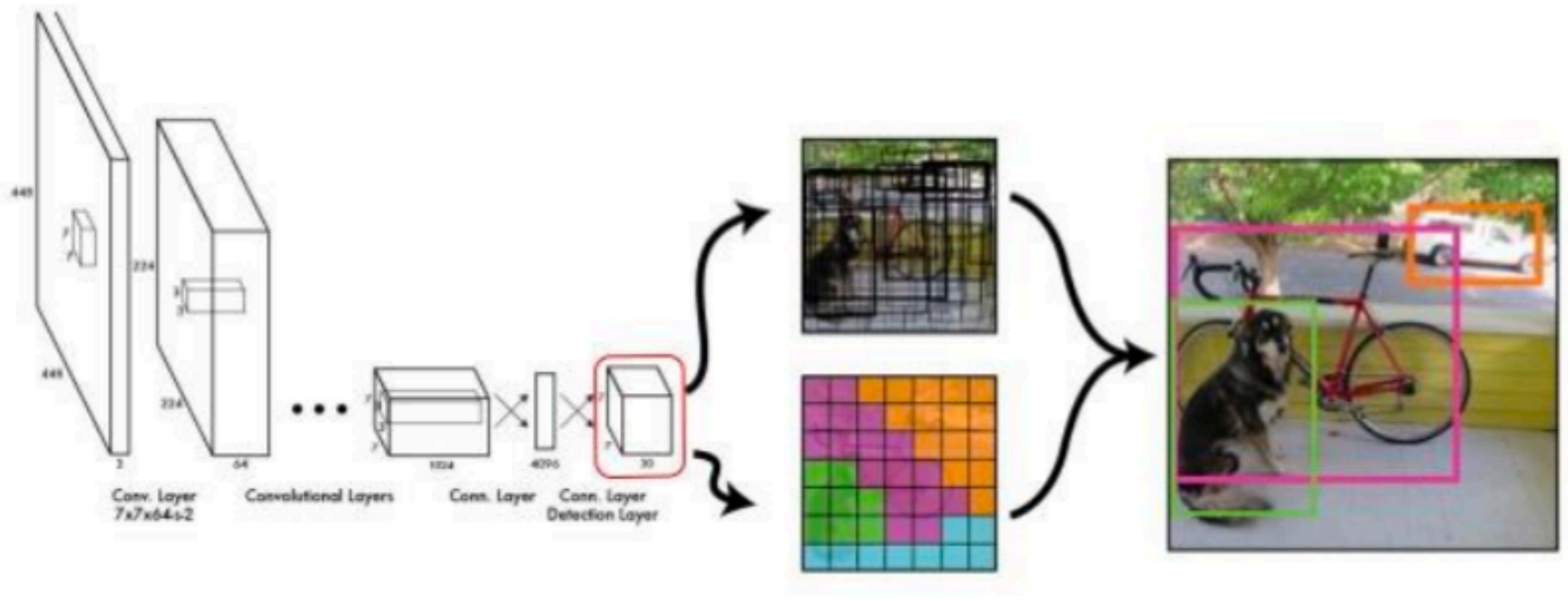
Step 4: Finding a person's name after encoding a face

This step is the easiest one in the pipeline. We compare a given face encoding with a list of known face encodings and find the best match. The result gives the name of the person.


Will Ferrell (92%)
Chad Smith (80%)
Jimmy Fallon (76%)

All steps can be combined into a single Neural Network!

YOLO: You Only Look Once



You only look once: Unified, real-time object detection

J Redmon, S Divvala, R Girshick… - Proceedings of the IEEE …, 2016 - cv-foundation.org

… Using our system, **you only look once** (YOLO) at an image to … of-the-art detectors, we **look** at a detailed breakdown of results … [19] For each category at test time we **look** at the top N predic…

☆ Save 99 Cite Cited by 23515 Related articles All 44 versions »

You Only Look Once: Unified, Real-Time Object Detection

Joseph Redmon*, Santosh Divvala*†, Ross Girshick¶, Ali Farhadi*†
University of Washington*, Allen Institute for AI†, Facebook AI Research¶
http://pjreddie.com/yolo/

Abstract

We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

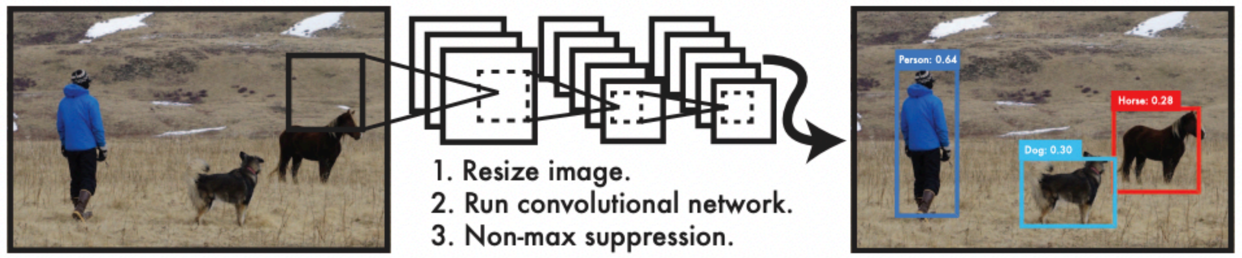
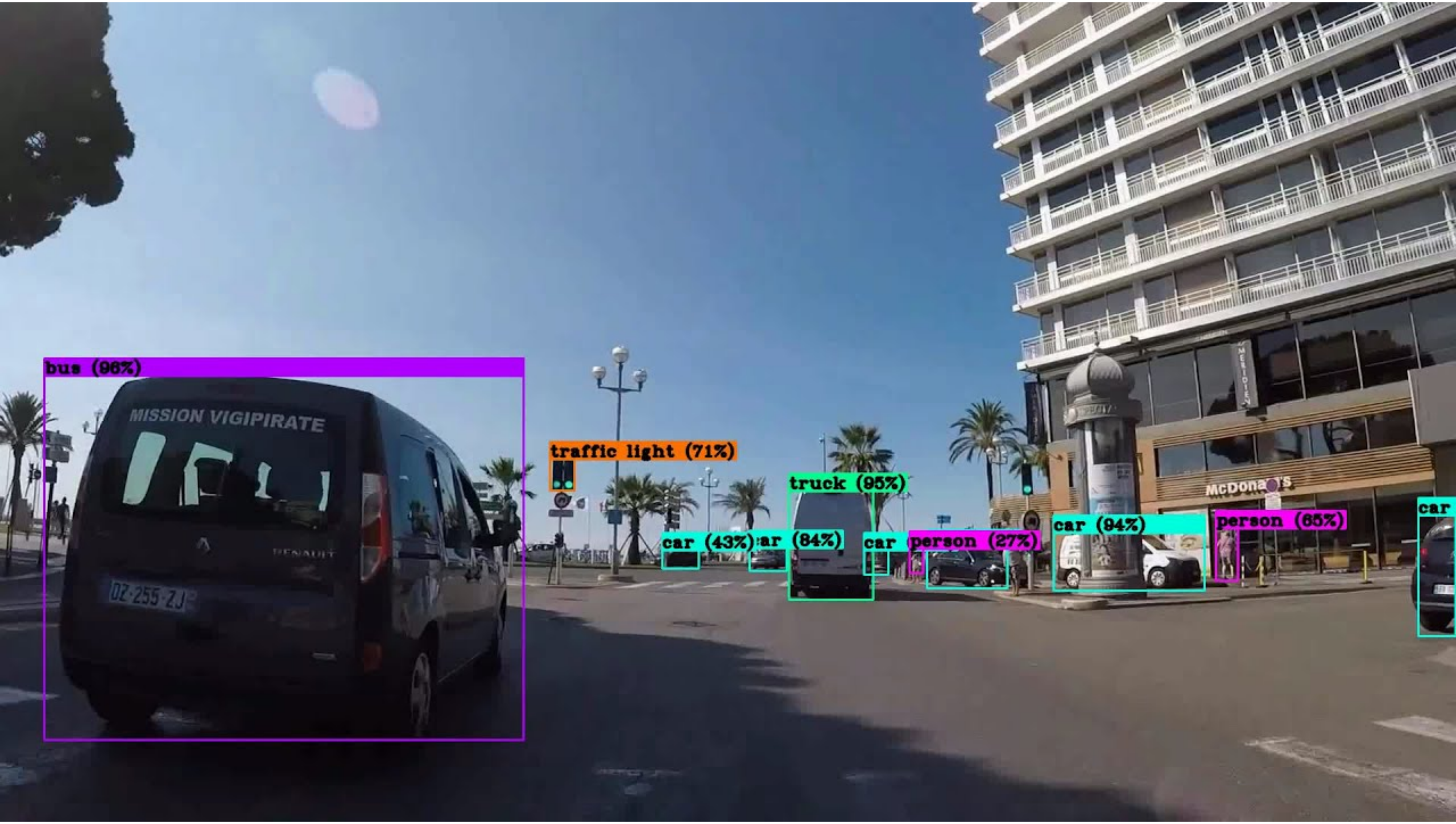


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.


bus (96%)
MISSION VIGIPIRATE
DZ-255-ZJ
traffic light (71%)
truck (95%)
car (43%)
ar (84%)
car
person (27%)
car (94%)
person (65%)
car
McDonald's