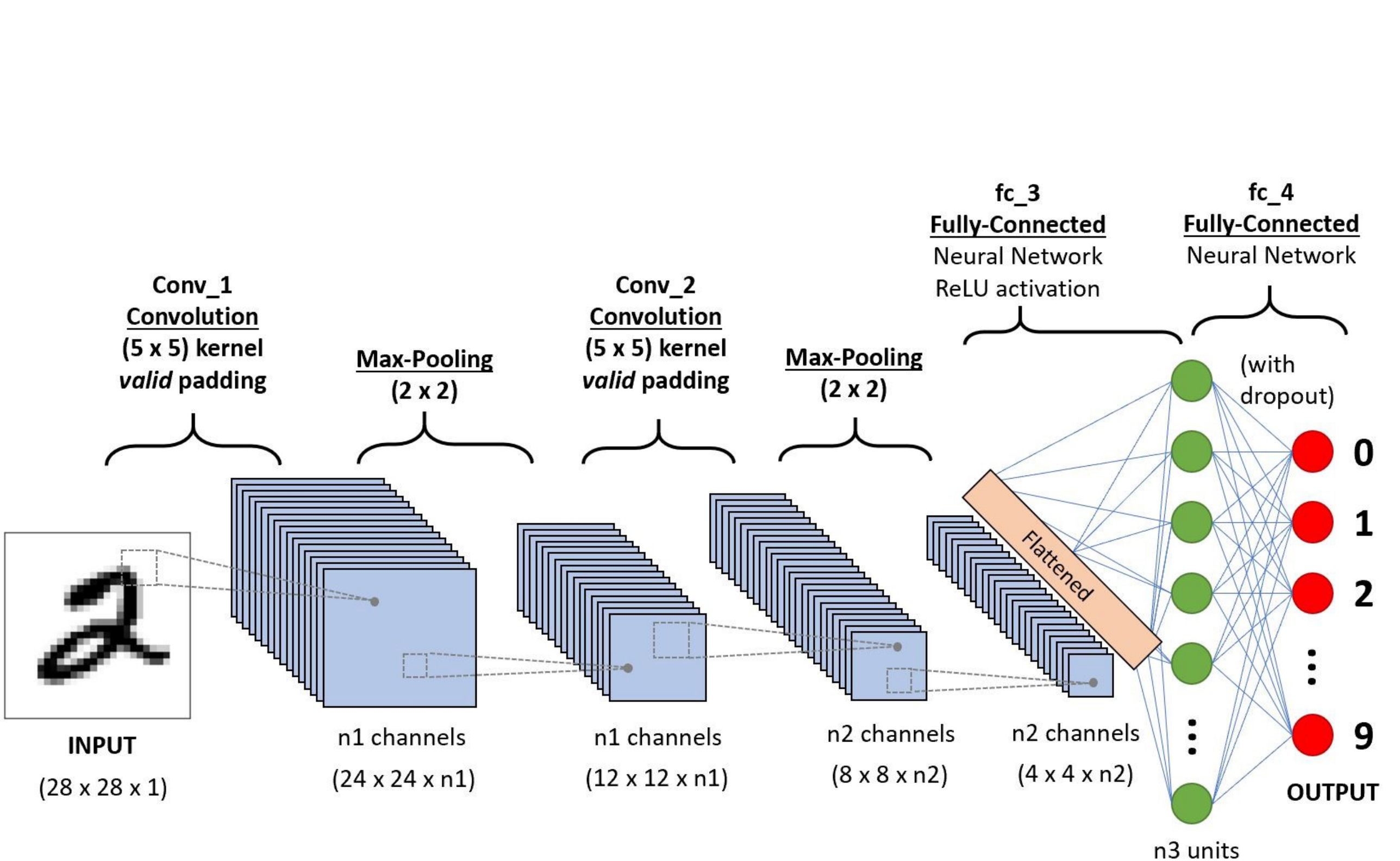
Lecture #9: Convolutional Neural Networks

COMP3806: Intelligent Systems
Inzamam Rahaman


Conv_1
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
Conv_2
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
fc_3
Fully-Connected
Neural Network
ReLU activation
fc_4
Fully-Connected
Neural Network
(with dropout)
Flattened
0
1
2
9
OUTPUT
INPUT
(28 x 28 x 1)
n1 channels
(24 x 24 x n1)
n1 channels
(12 x 12 x n1)
n2 channels
(8 x 8 x n2)
n2 channels
(4 x 4 x n2)
n3 units

Feed-foward Neural Networks

- Feed forward neural networks take in rows of data as input

- However, a lot of interesting data isn't natively in row form

- For example, images are natively expressed as 2D or 3D tensors

- Can we still use neural networks for such data?

Image Storage

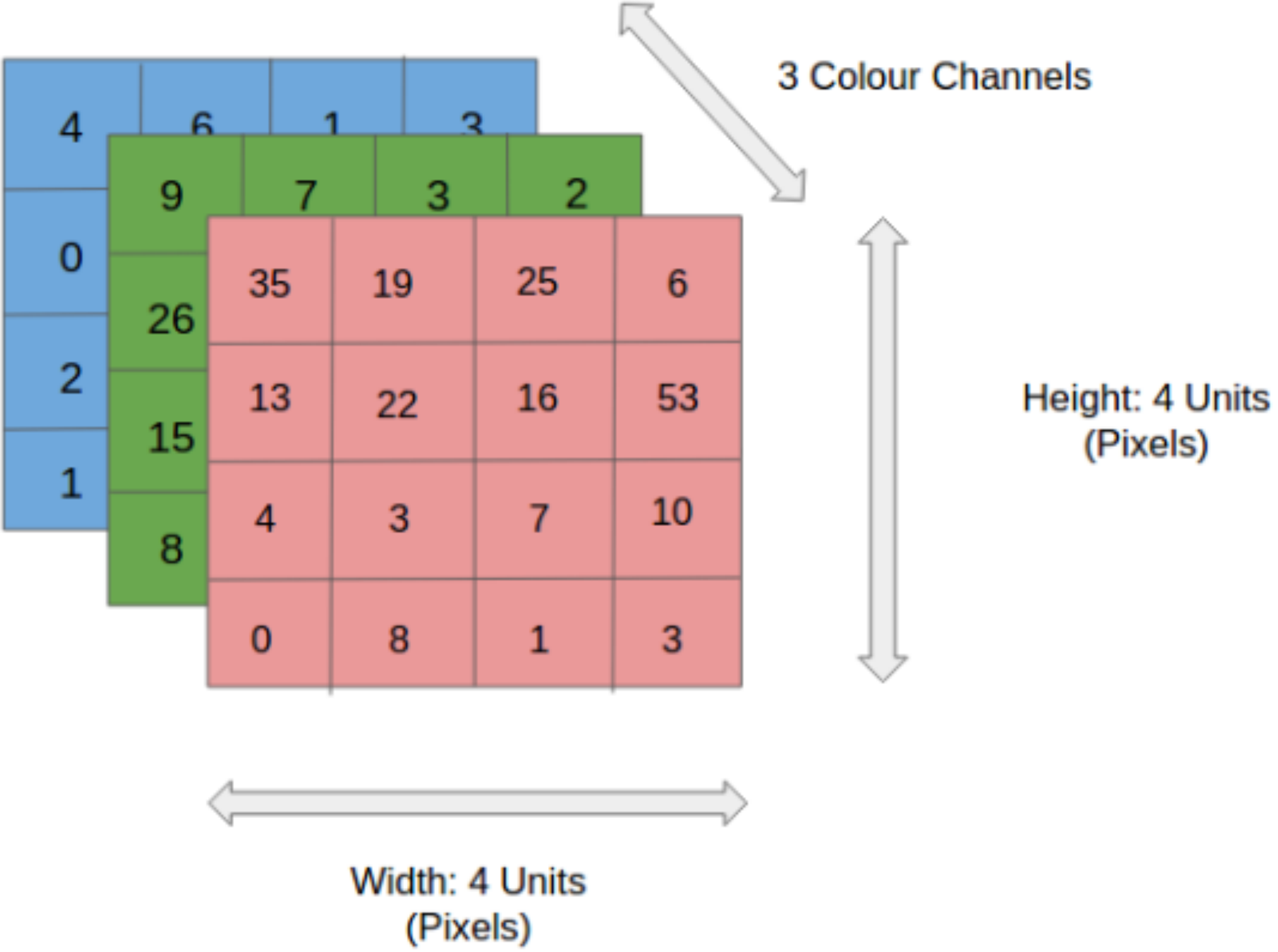
- Images can be conceived as rank 2 tensors - matrices - if they are black/white images

- Or rank 3 tensors when they are colour images

  - Each pixel has RBG triple

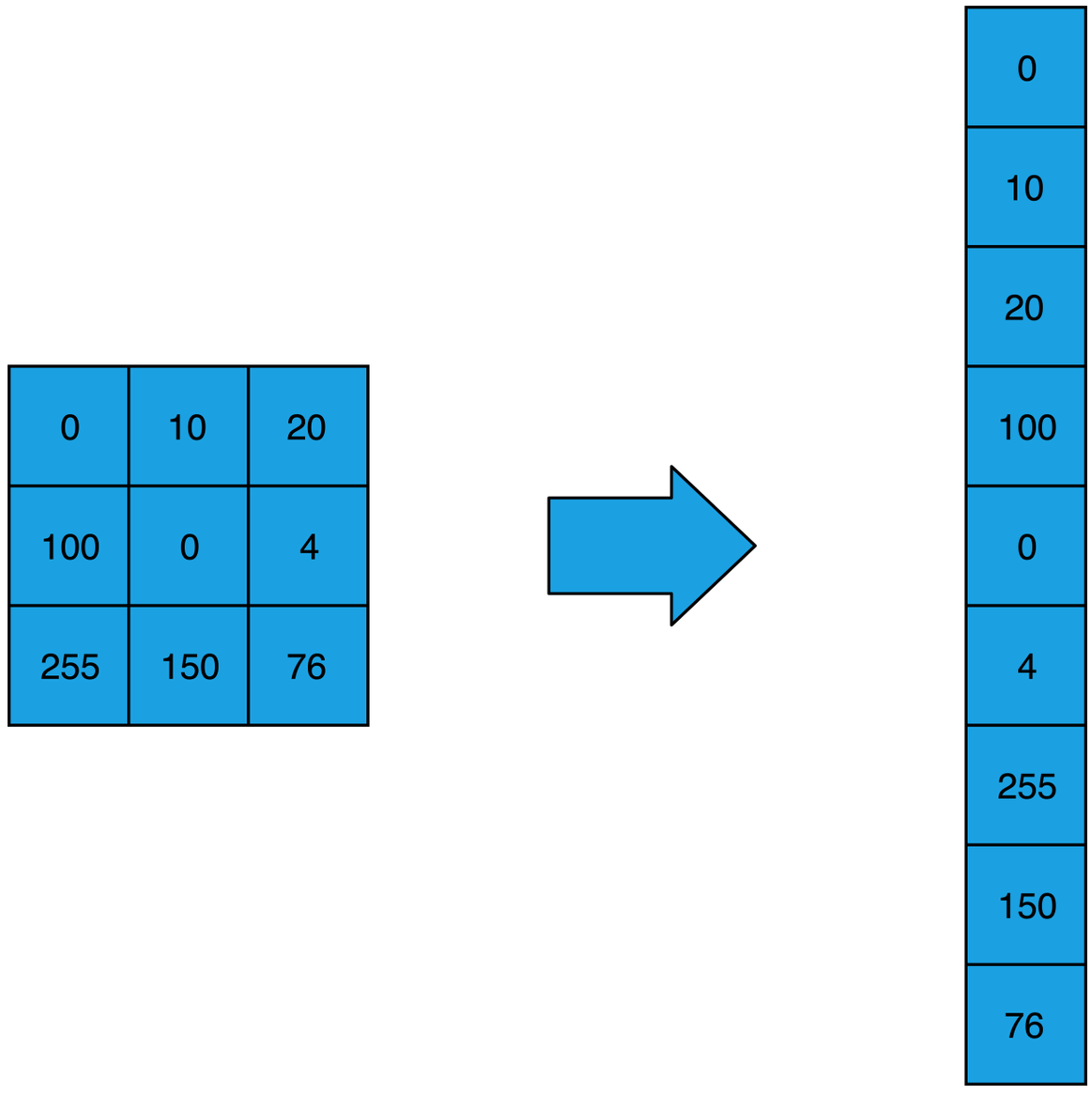
  - Each colour is noted as a channel - so a $W \times H$ colour image is represented as a $W \times H \times 3$ tensor

  - Think of the data as being a cuboid shape


3 Colour Channels
Height: 4 Units
(Pixels)
Width: 4 Units
(Pixels)
4 6 1 3
9 7 3 2
0 35 19 25 6
26 13 22 16 53
2 15 4 3 7 10
1 8 0 8 1 3

Feed-forward neural networks

- One strategy is to “flatten” higher rank tenors into vectors

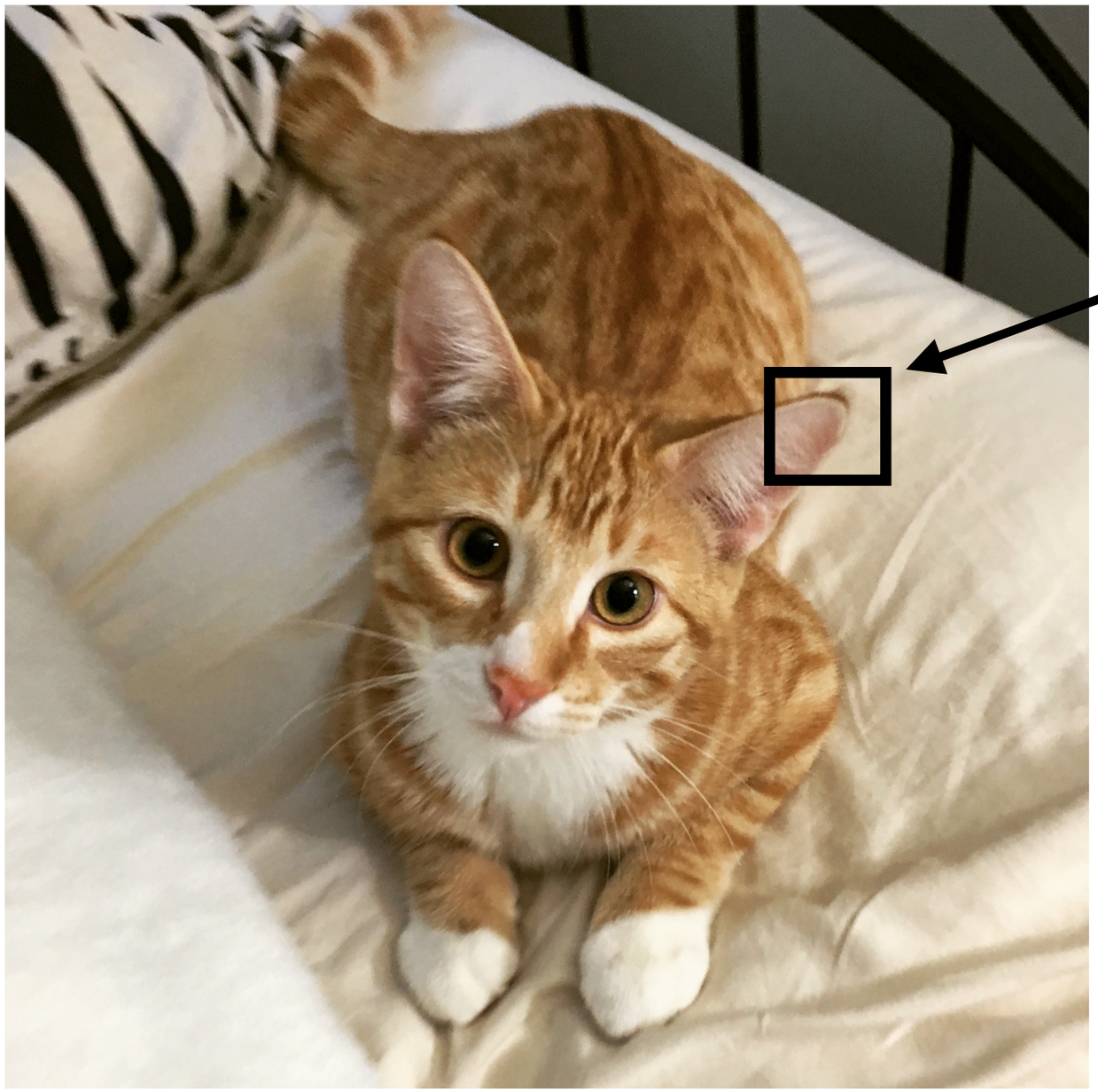


Feed-foward neural networks

- Flattened data is then used as input
- Sometimes this can actually work okay, for example, on the MNIST dataset, this can yield okay performance
- But for many more complicated problems - such as distinguishing between dogs and cats, this would not work!
- Why?

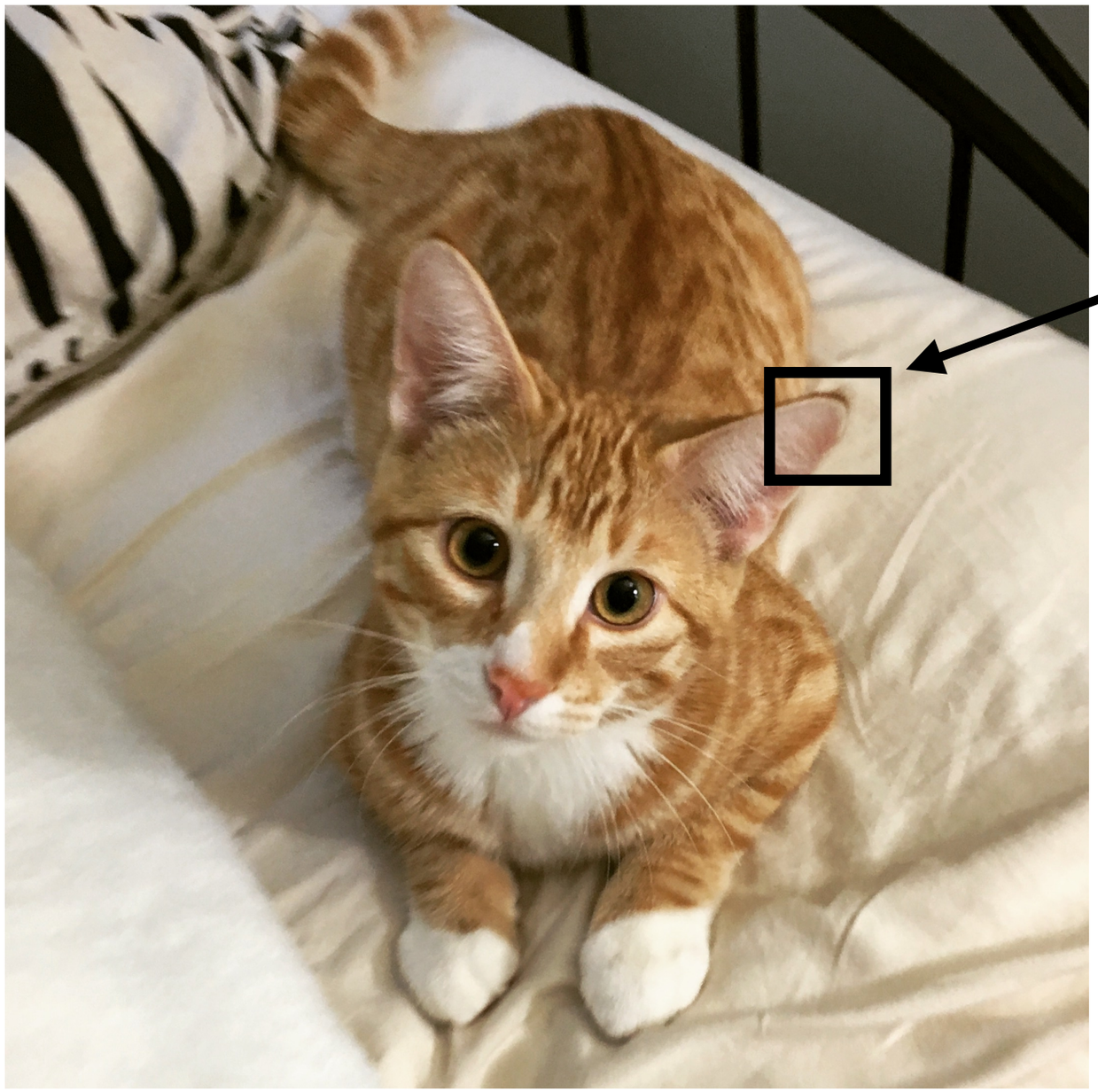
Spatial Locality and Images

Spatial Locality and Images



Individual Pixels rarely encode meaningful features!

Spatial Locality and Images



Individual Pixels rarely encode meaningful features!

Need to “aggregate” pixels to compose features!

NNs and Representation Learning

- Recall that the hidden layers of NNs learn ways to re-represent data into a form that is soluble by a linear model

- If we are going to use NNs anyway, can we use them to engineer features from images directly without use needing to specify them upfront?

- Much like how we used NNs to self-engineer representations of vector data, can use it for arbitrary tensors as well

Can we come up with a type of layer that would capture spatially local information?

Convolutions (in an ML context)

- Convolution operator uses a filter (also called a kernel)
- Act as a sort of sliding window across image data and intermediary tensors to compute features that take into account spatial locality
- Filters contain weights - parameters - of the network that we can tune using gradient descent
    - Filter applies hadamard product and sum as it moves across input
- Uses the same backpropogation algorithm but with more complicated computations!

Dimension of a Convolution

- The dimension of a convolution refers to its movement
  - Think number of loops to iterate over input
- A 2D convolution moves across width and then height
  - Used for image data
- A 1D convolution moves across the vector
  - Used for Univariate Time Series encoded as a single vector
- A 3D moves across width, height, and depth
  - Used for more complicated scenarios such as videos

2D Filter properties

- Filters have several properties
  - Width - width of filter
  - Length - length of filter
    - Most filters are square
- In channels - number of channels of input data
- Padding - amount of extra padding placed around input, default is 0
- Stride - the number of blocks moved for the sliding window, default is 1
- Dilation - determines the shape of the filter relative to rectangle, dilation is almost always 0

Convolutional Layers

- Each convolutional layer contains one or more filters
- All filters share the same properties
- The number of filters in a convolutional layer is referred to as its number of out channels
- Just like other layers in feed-forward networks, convolutional layers also feed their output through a non-linearity such as ReLU or σ.

Convolutional Layer

Input width or height

$$W_{out} = \left\lfloor \frac{(W_{in} + 2P - D \times (F - 1) - 1)}{S} \right\rfloor + 1$$

Output width or height

$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0
$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	1	0
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1
0	0	1	1	0
0	1	1	0	0

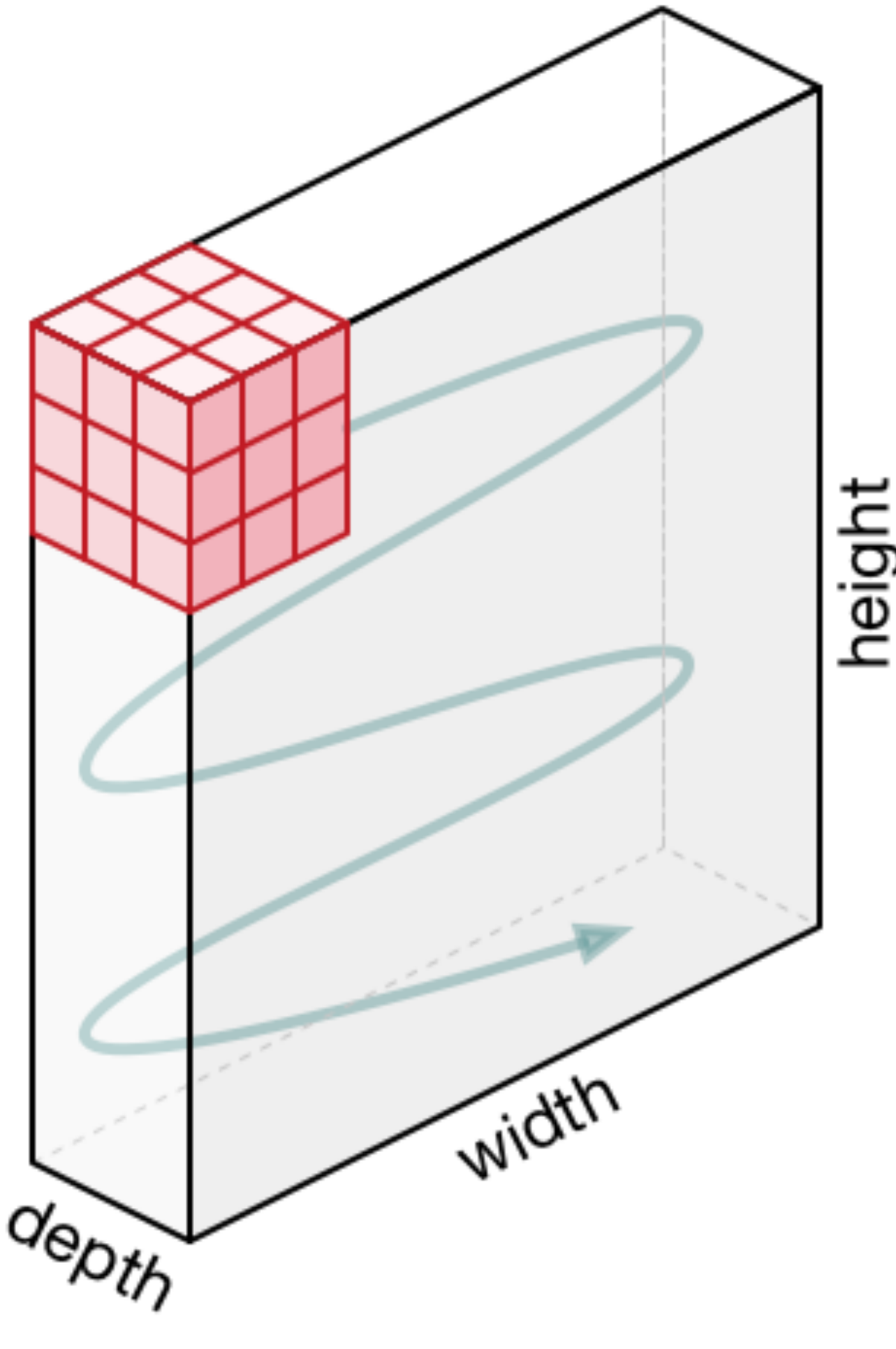
Image

4		

Convolved Feature

Convolutional Layers

- Operation generalises to rank 3 tensors trivially



0	0	0	0	0	0	...
156	155	156	158	158	158	...
153	154	157	159	159	159	...
149	151	155	158	159	159	...
146	146	149	153	158	158	...
145	143	143	148	158	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

$308 + -498 + 164 + 1 = -25$

Bias = 1

Output

-25				...
				...
				...
				...
...	...	...	...	...

Parameter Sharing

- Aside from the benefits incurred in the filters learning spatial feature maps, the re-use of weights also helps save memory and computation time

- Reducing the number of parameters reduces the variance of the model

- Thereby reduces possibility of overfitting

Pooling

- Feature maps learned by convolutional layer can learn “noise”
- Need to smooth out “noise” by aggregating nearby cells of output
- This is the purpose of the pool operation
- Like a filter that applies an aggregation operation over a square region of its input
- Aggregations: max, average, min, median, mode
- Also done for the practical reasoning of reducing output size

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

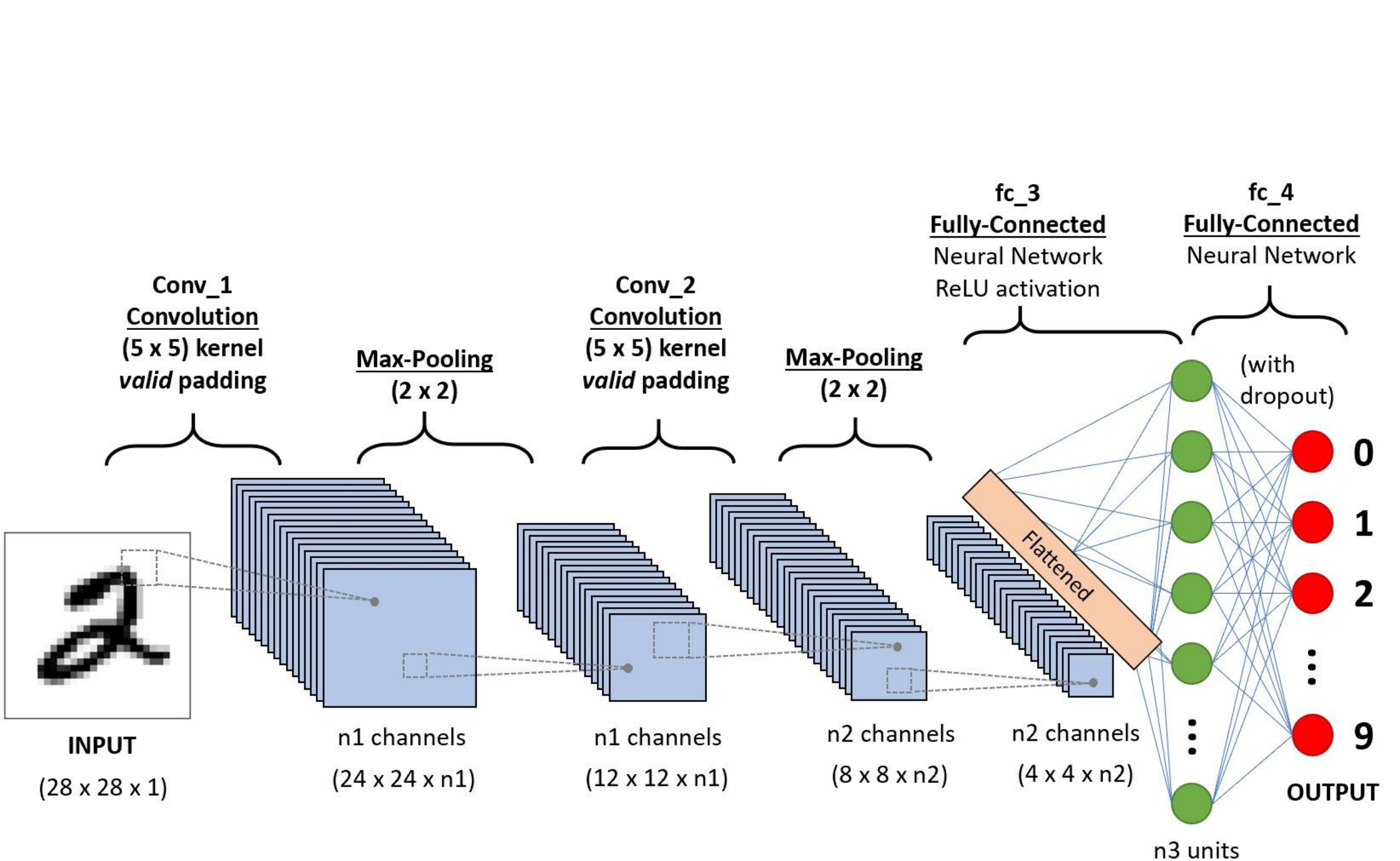
CNNs

- A CNN or CovNet is a neural network with at least one convolutional layer

- Image data is passed through a series of convolutional layers and pooling layers

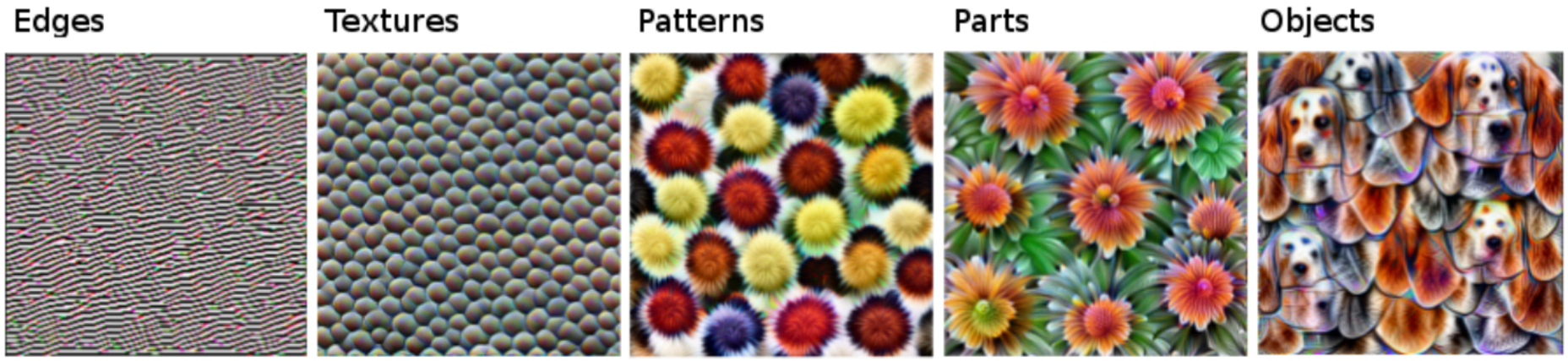
- Eventually, output is flattened, but spatially relevant features would have been localised by the time of flattening

- From flatten layer onward, the network is a feed-forward (most of the time)


Conv_1
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
Conv_2
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
fc_3
Fully-Connected
Neural Network
ReLU activation
fc_4
Fully-Connected
Neural Network
(with dropout)
Flattened
0
1
2
9
OUTPUT
INPUT
(28 x 28 x 1)
n1 channels
(24 x 24 x n1)
n1 channels
(12 x 12 x n1)
n2 channels
(8 x 8 x n2)
n2 channels
(4 x 4 x n2)
n3 units

Representation Learning and CNNs

- As aforementioned, we are using convolutions to learn features that are captured by spatial locality
- Do we have anyway to know what these features are?


Edges
Textures
Patterns
Parts
Objects

Visual Cortex Similarities

- This is actually similar to how our visual cortex functions

- Data passes from the retina to through the optic nerve. Then from optic nerve to a region called the lateral geniculate nucleus

- V1 (primary visual cortex) is a brain region with simple and complex cells arranged in a 2 matrix

- Inferotemporal Cortex takes signal from V1->V2->V3->V4

Visual Cortex Differences

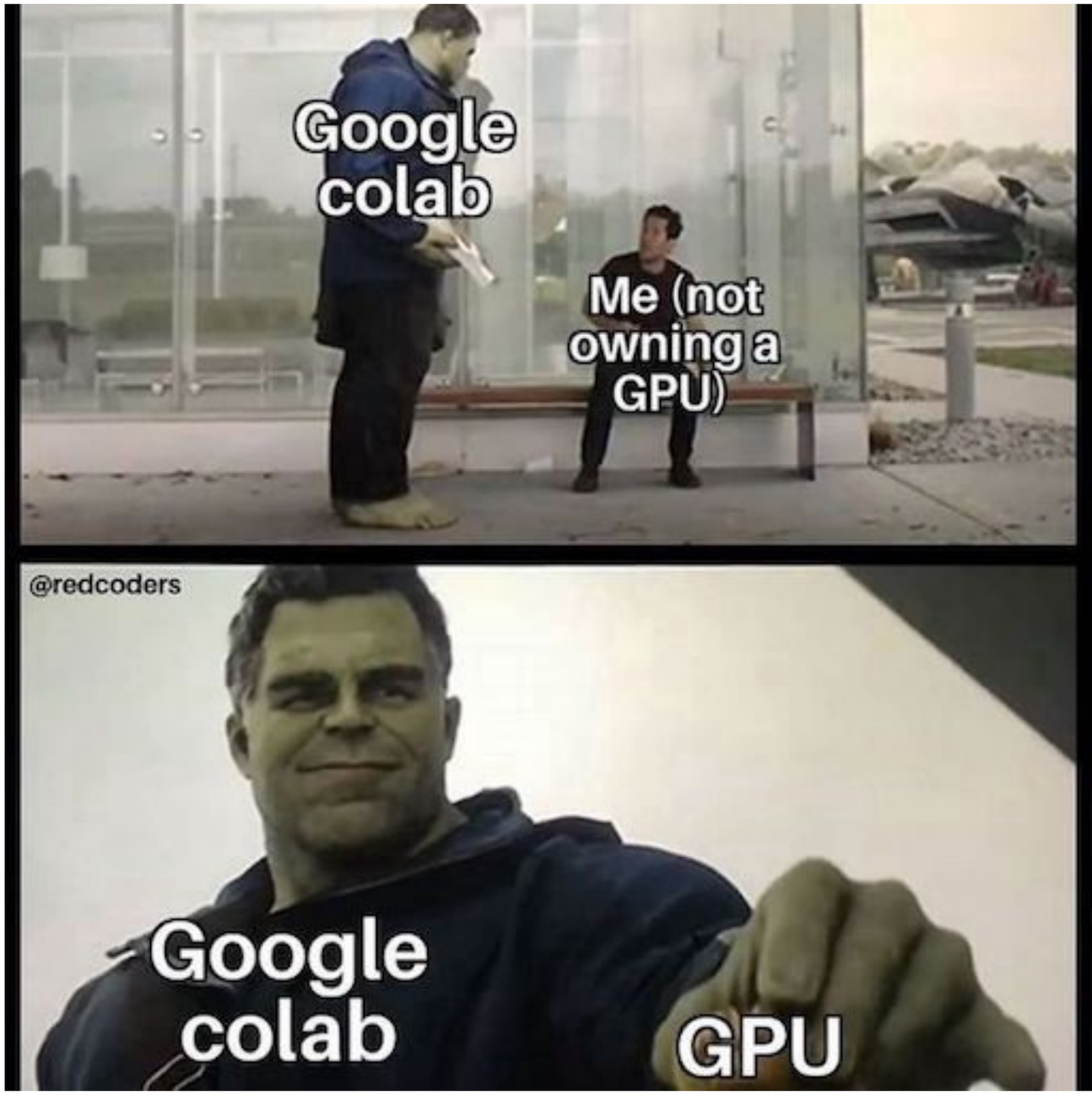
- Human eye has inconsistent resolution across retina. Central region called fovea has highest resolution

- Image data is sampled incrementally (and non-uniformly) in movements called saccades

- Visual cortex is affected by feedback from higher processes. Signal from other sensory organs are integrated in visual discernment

Training CNNs

- Sometimes, Feed-forward NNs are okay to train on CPUs
- However, CNNs are much more difficult to train without a GPU
- Will look into using Google Colab in the labs next week to train simple CNNs



Pre-training and Transfer Learning

- Even with a GPU, large CNNs are difficult to train
- Several published architectures are trained on several benchmark datasets such as ImageNet, MNIST, FashionMNIST, etc…
  - Curated to be representative of several typical scenarios
- Sometimes the spatial features maps learnt by these architectures and encoded in their weights can apply to other datasets
- Can we re-use these weights?
  - In a slightly adjusted architecture?

Pre-training

- Chose an architecture
- Instead of randomising weights, use weights determined after training the network on another dataset
- We then use backpropogation to refine these weights for our specific instance
- Run for fewer epochs

Transfer Learning

- Similar in that we re-use weights
- But instead of retraining the entire network, we freeze some of the lower layers
- In the case of CNNs, we can freeze the convolutional layers and only train the feedforward portions of the network

Pre-training vs Transfer Learning

- Pre-training takes more time, but the feature transformations learnt are more refined for problem space

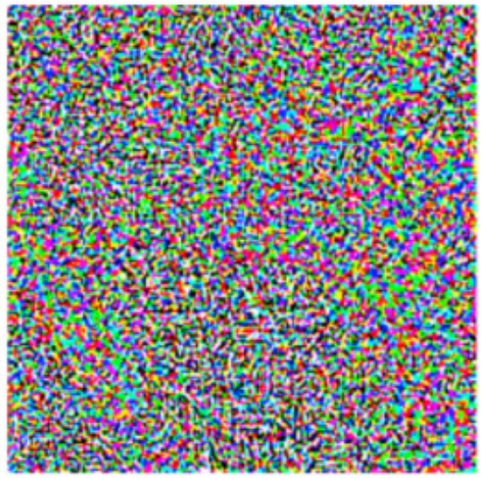
- Transfer learning takes less time, but features are not as refined for problem space

- For many industry cases, can get away with Transfer Learning

Adversarial Attacks and Flaws

- CNNs are not perfect
- Sometimes, the exact mappings the network learns to obtain features or the features themselves are fragile
- Mild corruption that you and I would ignore, a CNN might decide is important - leading to poor decisions
    - Sometimes, dangerous ones….

$+ .007 \times$



$=$

“panda”

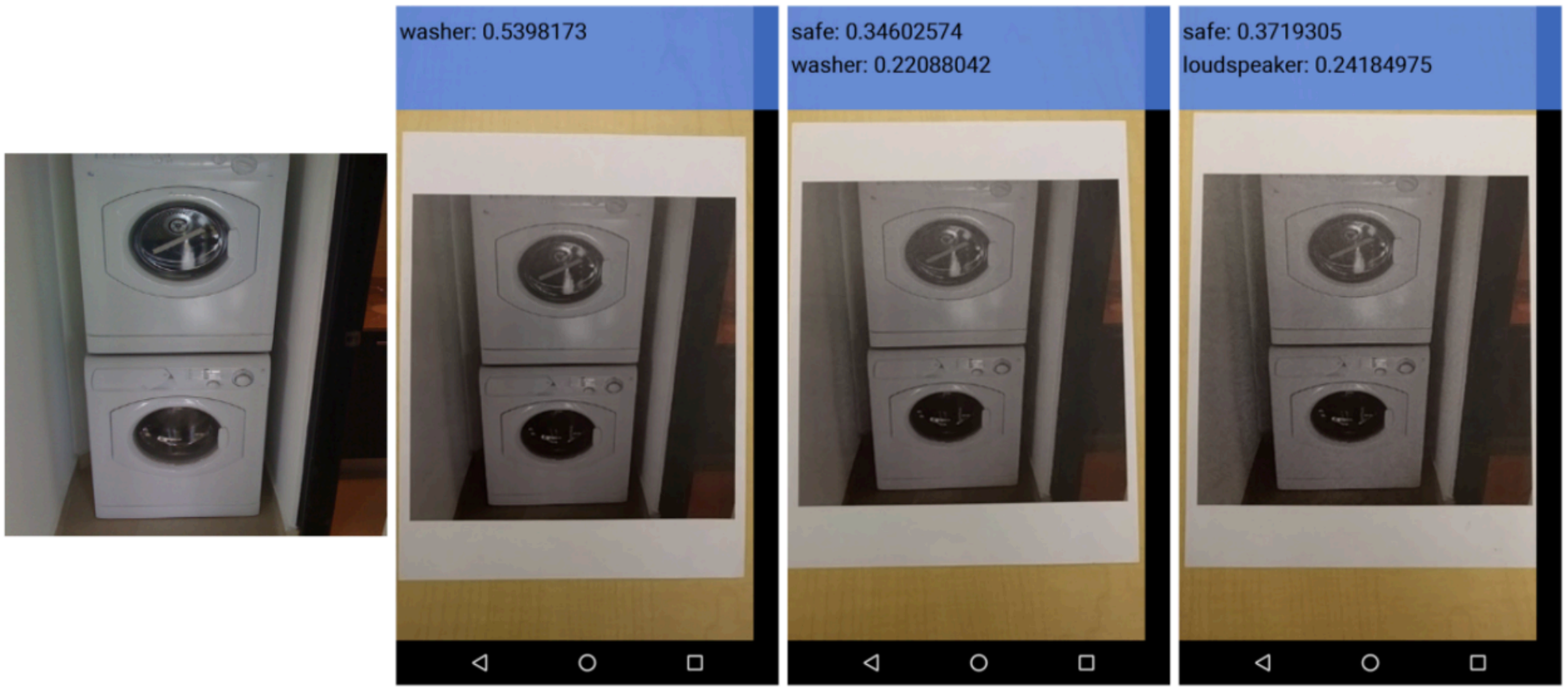
57.7% confidence

noise

“gibbon”

99.3% confidence

From Goodfellow et al. 2015



(a) Image from dataset (b) Clean image (c) Adv. image, $\epsilon = 4$ (d) Adv. image, $\epsilon = 8$

Source: Adversarial Examples in the Physical World. Kurakin et al, ICLR 2017.

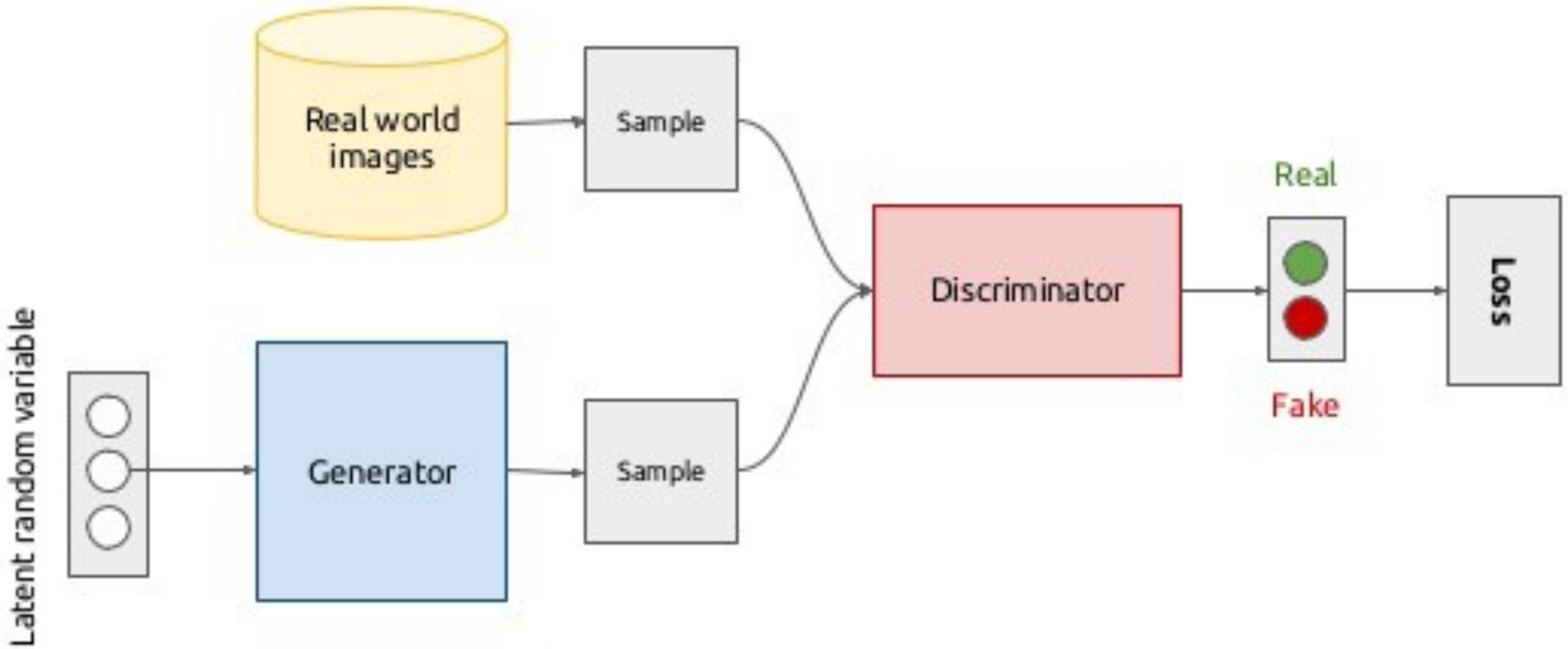
The left image shows real graffiti on a Stop sign, something that most humans would not think is suspicious. The right image shows a physical perturbation applied to a Stop sign. The systems classify the sign on the right as a Speed Limit: 45 mph sign! Source: Robust Physical-World Attacks on Deep Learning Visual Classification.

Techniques to overcome

- One common sense solution to deliberately pose adversarial examples during the training process

- Another is to use a technique called GANs

Generative adversarial networks (conceptual)



Using GANs

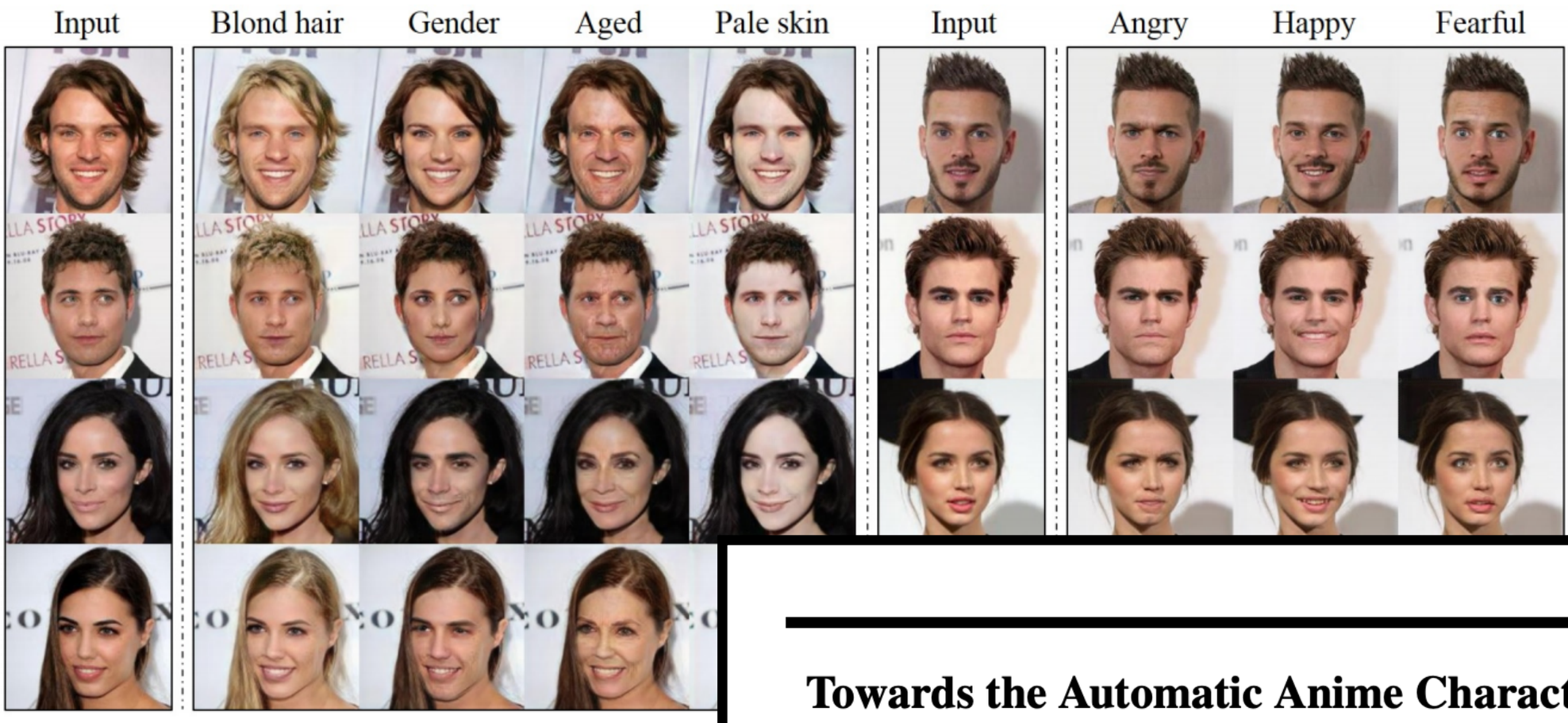
- Use GANs to train discriminator that is then used as pre-trained weights for CNN

- GANs also do some other really interesting things….

- DeepFakes are based on GANs

thispersondoesnotexist.com

Imagined by a GAN (generative adversarial network)
StyleGAN2 (Dec 2019) - Karras et al. and Nvidia
Don't panic. Learn how it works [1] [2] [3]
Help this AI continue to dream | Contact me
Code for training your own [original] [simple]
Art • Cats • Horses • Molecules | News | Friends | Office
Another | Save



Towards the Automatic Anime Characters Creation with Generative Adversarial Networks

Yanghua Jin
School of Computer Science
Fudan University
jinyh13@fudan.edu.cn

Jiakai Zhang
School of Computer Science
Carnegie Mellon University
jiakaiz1@andrew.cmu.edu

Minjun Li
School of Computer Science
Fudan Univerisity
minjunli13@fudan.edu.cn

Yingtao Tian
Department of Computer Science
Stony Brook University
yittian@cs.stonybrook.edu

Huachun Zhu
School of Mathematics
Fudan Univerisity
zhuhc14@fudan.edu.cn

Zhihao Fang
Department of Architecture
Tongji Univerisity
fangzhihao126@gmail.com

Abstract

Preventing Overfitting

- Overfitting is another problem with CNNs
  - Remember high variance
- We can apply ℓ_p regularisation like with other methods
- Can also use Dropout
- In Dropout, we randomly set outputs from activation function to 0
- Shown to be “equivalent” to regularisation
  - But with nicer computational properties