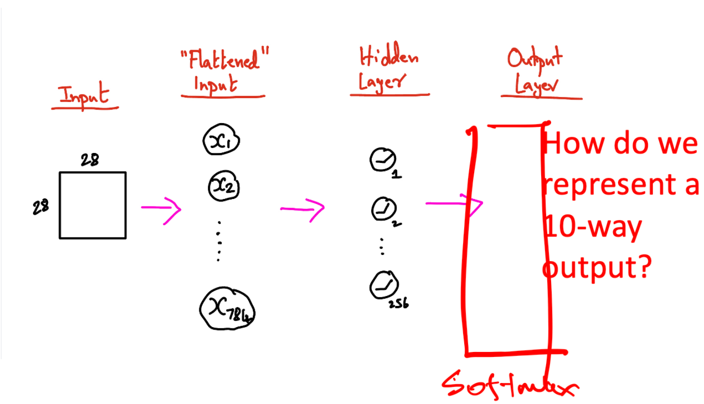
Lecture 4
Deep Learning for Computer Vision – Convolutional Neural Networks and Transfer Learning

15.S04: Hands-on Deep Learning
Spring 2024

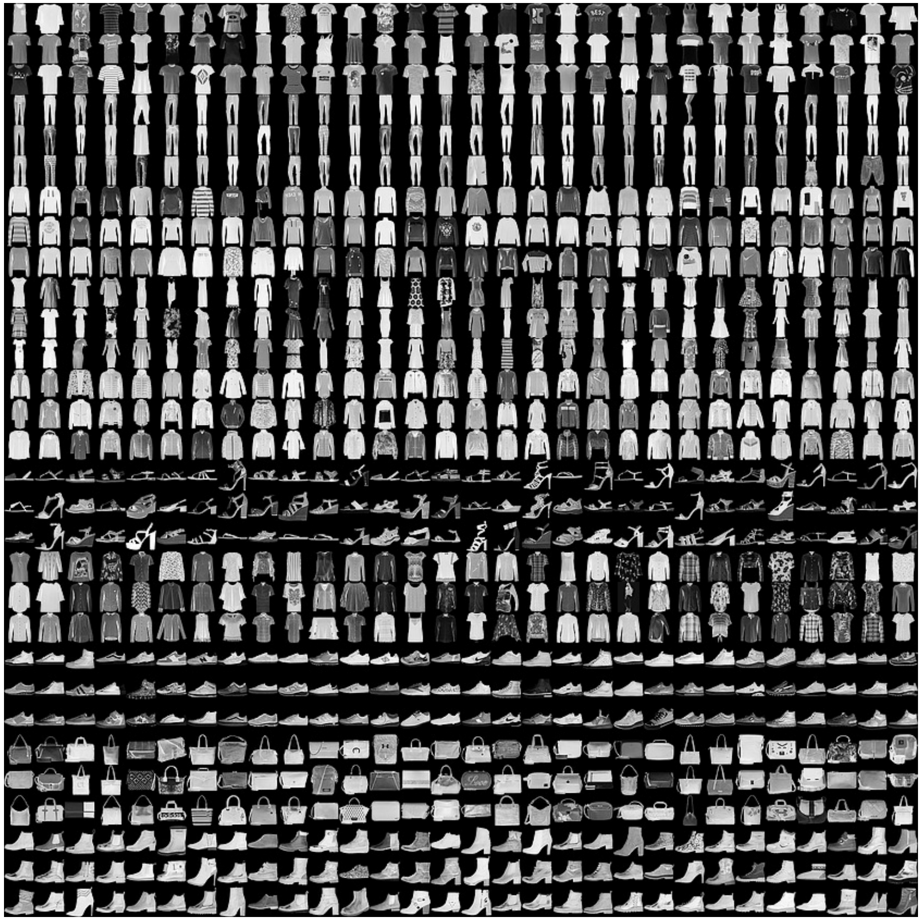
Farias, Ramakrishnan

Fashion MNIST

We saw previously that an NN with a single hidden layer can get to 85% + accuracy on this dataset.



How can we do better?



Source: https://www.kaggle.com/datasets/zalando-research/fashionmnist

Handling Images "Naturally"

- When we flatten the image matrix into a long vector and feed it to a dense layer, several "undesirable" things happen:

Handling Images "Naturally"

- When we flatten the image matrix into a long vector and feed it to a dense layer, several "undesirable" things happen:
  - We need to learn "too many" parameters
    - Flattening a 3024×3024 – pixel color image (from your phone) and connecting to a single 100-neuron dense layer generates approximately ? parameters.

Handling Images "Naturally"

- When we flatten the image matrix into a long vector and feed it to a dense layer, several "undesirable" things happen:
  - We need to learn "too many" parameters
    - Flattening a 3024×3024 – pixel color image (from your phone) and connecting to a <u>single</u> 100-neuron dense layer generates approximately 2.7 billion parameters.

Handling Images "Naturally"

- When we flatten the image matrix into a long vector and feed it to a dense layer, several "undesirable" things happen:
  - We need to learn "too many" parameters
    - Flattening a 3024×3024 – pixel color image (from your phone) and connecting to a single 100-neuron dense layer generates approximately 2.7 billion parameters.
    - This is computationally demanding, very data-hungry and increases the risk of overfitting

Handling Images "Naturally"

- When we flatten the image matrix into a long vector and feed it to a dense layer, several "undesirable" things happen:
  - We need to learn "too many" parameters
  - We lose the *local spatial adjacency* relationships between pixels that define features of the image.

Handling Images "Naturally"

- When we flatten the image matrix into a long vector and feed it to a dense layer, several "undesirable" things happen:
  - We need to learn "too many" parameters
  - We lose the *local spatial adjacency* relationships between pixels that define features of the image.
  - We don't learn once and reuse repeatedly
    - If a feature of the image (e.g., a vertical line or a circle) appears in different places in the image, the network should "learn it once and use it again and again" rather learn it separately each time.

Handling Images "Naturally"

- When we flatten the image matrix into a long vector and feed it to a dense layer, several "undesirable" things happen:
  - We need to learn "too many" parameters
  - We lose the *local spatial adjacency* relationships between pixels that define features of the image.
  - We don't learn once and reuse repeatedly

- Convolutional layers were developed to address these shortcomings

Convolutional Layers

Convolutional Layers and Filters

- *A convolutional filter* is a small square matrix of numbers

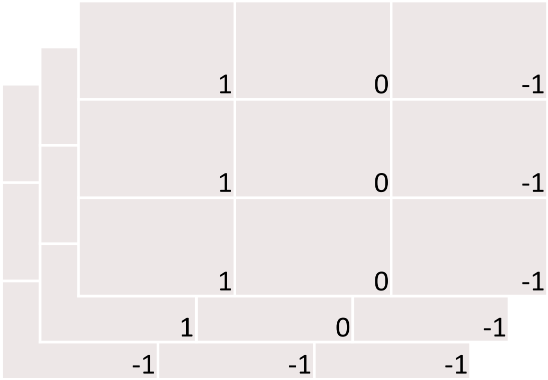
1	1	1
0	0	0
-1	-1	-1

Convolutional Layers and Filters

- *A convolutional filter* is a small square matrix of numbers

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

- A *convolutional layer* is composed of one or more *convolutional filters*



The Convolutional Filter

By choosing the numbers in a filter carefully and "applying" the filter to an image, different features of the image can be detected (as we will demonstrate shortly)

The Convolutional Filter

By choosing the numbers in a filter carefully and "applying" the filter to an image, different features of the image can be detected (as we will demonstrate shortly)

This filter can detect horizontal edges!

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

This filter can detect vertical edges

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

Applying a convolutional filter to an image is called the “convolution operation”

Input

0	0	0	0	0	0	0
5	5	5	5	5	5	5
10	10	10	10	10	10	10
15	15	15	15	15	15	15
10	10	10	10	10	10	10
5	5	5	5	5	5	5
0	0	0	0	0	0	0

*

Filter

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

=

Output

Applying a convolutional filter to an image is called the "convolution operation"

Input

0	0	0	0	0	0	0
5	5	5	5	5	5	5
10	10	10	10	10	10	10
15	15	15	15	15	15	15
10	10	10	10	10	10	10
5	5	5	5	5	5	5
0	0	0	0	0	0	0

*

Filter

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

=

Output

The convolution operation

- "Overlay" the filter onto the top-left of the image

Applying a convolutional filter to an image is called the “convolution operation”

Input

0	0	0	0	0	0	0
5	5	5	5	5	5	5
10	10	10	10	10	10	10
15	15	15	15	15	15	15
10	10	10	10	10	10	10
5	5	5	5	5	5	5
0	0	0	0	0	0	0

*

Filter

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

=

Output

The convolution operation

- “Overlay” the filter onto the top-left of the image
- Multiply matching elements and add up:
  - 0*1 + 0*1 + 0*1 + 5*0 + 5*0 + 5*0 + 10* -1 + 10* -1 + 10*-1 = -30

Applying a convolutional filter to an image is called the “convolution operation”

Input

0	0	0	0	0	0	0
5	5	5	5	5	5	5
10	10	10	10	10	10	10
15	15	15	15	15	15	15
10	10	10	10	10	10	10
5	5	5	5	5	5	5
0	0	0	0	0	0	0

*

Filter

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

=

Output

0				

The convolution operation

- “Overlay” the filter onto the top-left of the image
- Multiply matching elements and add up:
  - 0*1 + 0*1 + 0*1 + 5*0 + 5*0 + 5*0 + 10* -1 + 10* -1 + 10*-1 = -30
- Run through a ReLU*: max(0, -30) = 0. This number is the top-left cell of the output

*strictly speaking, the ReLU isn’t part of the convolution operation but we include it here for convenience

The Convolution Operation

Input

0	0	0	0	0	0	0
5	5	5	5	5	5	5
10	10	10	10	10	10	10
15	15	15	15	15	15	15
10	10	10	10	10	10	10
5	5	5	5	5	5	5
0	0	0	0	0	0	0

*

Filter

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

=

Output

0	0			

Slide the window one step to the right and repeat this process to get the second number of the output

The Convolution Operation

Input

0	0	0	0	0	0	0
5	5	5	5	5	5	5
10	10	10	10	10	10	10
15	15	15	15	15	15	15
10	10	10	10	10	10	10
5	5	5	5	5	5	5
0	0	0	0	0	0	0

*

Filter

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

=

Output

0	0	0	0	0

Slide the window one step to the right and repeat this process to get the second number of the output

When done with the first row, move to the start of the second row and continue as before …

The Convolution Operation

Input

0	0	0	0	0	0	0
5	5	5	5	5	5	5
10	10	10	10	10	10	10
15	15	15	15	15	15	15
10	10	10	10	10	10	10
5	5	5	5	5	5	5
0	0	0	0	0	0	0

*

Filter

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

=

Output

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
30	30	30	30	30
30	30	30	30	30

Slide the window one step to the right and repeat this process to get the second number of the output

When done with the first row, move to the start of the second row and continue as before …

… till you reach the bottom-right corner

By choosing the numbers in a filter carefully and applying the convolution operation, different features of the image can be detected

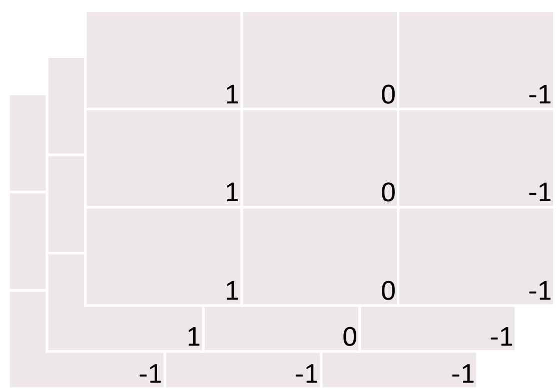
Switch to HODL-Lec-3-Convolution-Example.xlsx

Optional:

Check out https://setosa.io/ev/image-kernels/ to practice with different filters

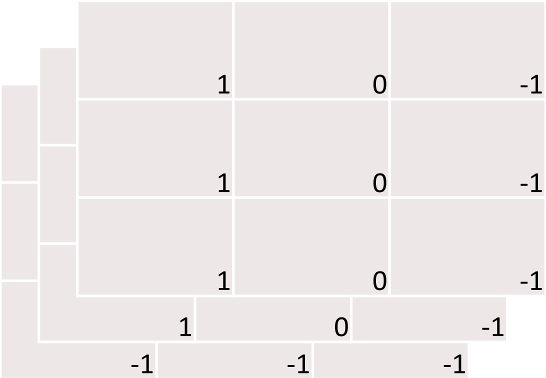
Convolutional Layers

A *convolutional layer* is composed of one or more convolutional filters



Convolutional Layers

A *convolutional layer* is composed of one or more convolutional filters



Each filter can be thought of as a <u>specialist</u> for detecting a particular feature (e.g., a horizontal line, an arc, a vertical line)

Applying a Convolutional *Layer* to an image

Input image

0	0	0	0	0	0	0
5	5	5	5	5	5	5
10	10	10	10	10	10	10
15	15	15	15	15	15	15
10	10	10	10	10	10	10
5	5	5	5	5	5	5
0	0	0	0	0	0	0

Conv layer with 2 filters

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

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

Output

Applying a Convolutional Layer to an image

Input image

0	0	0	0	0	0	0
5	5	5	5	5	5	5
10	10	10	10	10	10	10
15	15	15	15	15	15	15
10	10	10	10	10	10	10
5	5	5	5	5	5	5
0	0	0	0	0	0	0

Conv layer with 2 filters

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

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

Output

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
30	30	30	30	30
30	30	30	30	30

Applying a Convolutional Layer to an image

Input image

0	0	0	0	0	0	0
5	5	5	5	5	5	5
10	10	10	10	10	10	10
15	15	15	15	15	15	15
10	10	10	10	10	10	10
5	5	5	5	5	5	5
0	0	0	0	0	0	0

Conv layer with 2 filters

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

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

Output

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
30	30	30	30	30
30	30	30	30	30

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Applying a Convolutional Layer to an image

Input image

0	0	0	0	0	0	0
5	5	5	5	5	5	5
10	10	10	10	10	10	10
15	15	15	15	15	15	15
10	10	10	10	10	10	10
5	5	5	5	5	5	5
0	0	0	0	0	0	0

Conv layer with 2 filters

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

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

Output

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
30	30	30	30	30
30	30	30	30	30

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

These two 5x5 matrices can be represented as a tensor of shape _______?

Applying a Convolutional Layer to an image

Input image

0	0	0	0	0	0	0
5	5	5	5	5	5	5
10	10	10	10	10	10	10
15	15	15	15	15	15	15
10	10	10	10	10	10	10
5	5	5	5	5	5	5
0	0	0	0	0	0	0

Conv layer with 2 filters

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

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

Output

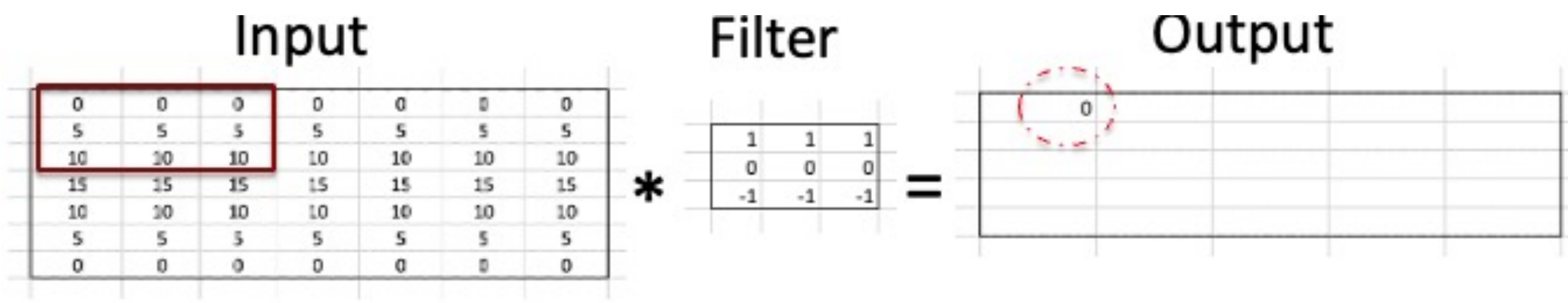
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
30	30	30	30	30
30	30	30	30	30

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

These two 5x5 matrices can be represented as a tensor of shape 5 x 5 x 2 or 2 x 5 x 5

Applying a Convolutional Filter to a color image

We know how to apply a convolutional filter to a 2-d tensor (e.g., a grayscale image)



How should we apply a convolutional filter to a rank-3 tensor (e.g., a color image)?

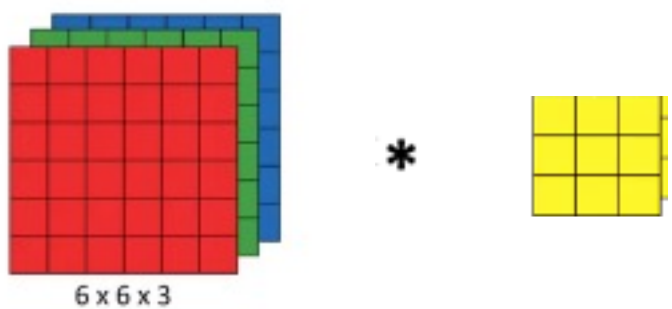
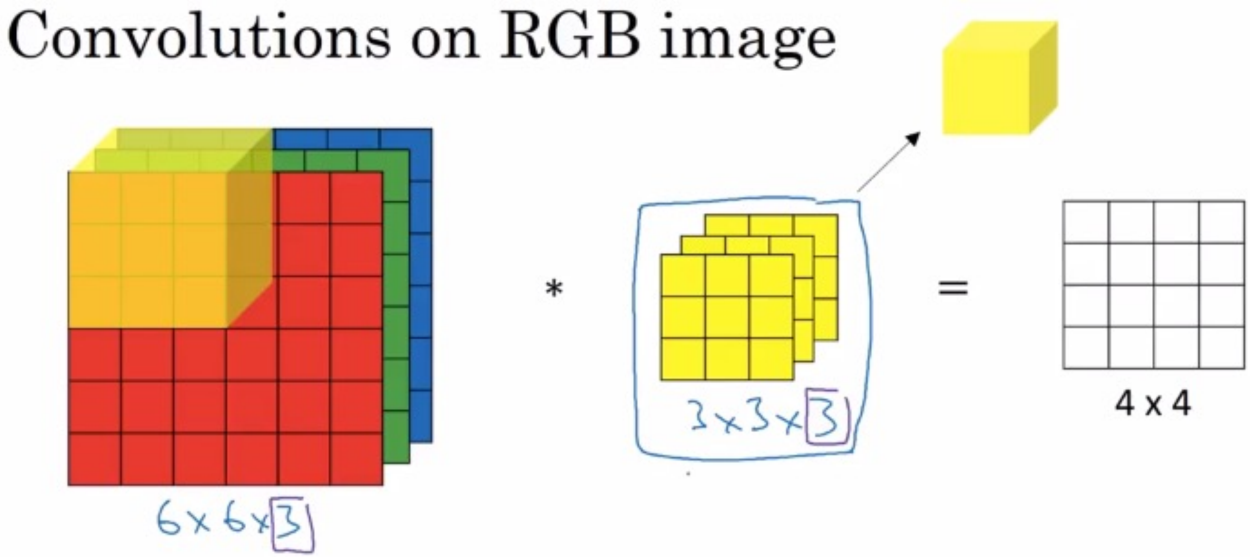


Image source: deeplearning.ai

Applying a Convolutional Filter to a color image



- We make the filter rank-3 as well and give it the same depth as the input
- The other aspects of the convolution operation are unchanged

Image source: deeplearning.ai

Please see Chapter 8.1 of the textbook for more detail on how convolutional filters and layers work

The Big Idea

- These filters seem excellent but how are we supposed to come up with the numbers in each filter?

The Big Idea

- These filters seem excellent but how are we supposed to come up with the numbers in each filter?

- In fact, convolutional filters used to be designed by hand. Computer Vision researchers invested a lot of effort in devising filters that could detect various types of image features

The Big Idea

- These filters seem excellent but how are we supposed to come up with the numbers in each filter?
- In fact, convolutional filters used to be designed by hand. Computer Vision researchers invested a lot of effort in devising filters that could detect various types of image features
- As we figured out how to train deep networks with lots of weights, a big idea emerged: **think of the numbers in the filter as <u>weights</u> and simply learn them from the data, just like we learn all the other weights**

Later conv layers “see” more of the original input than the earlier layers

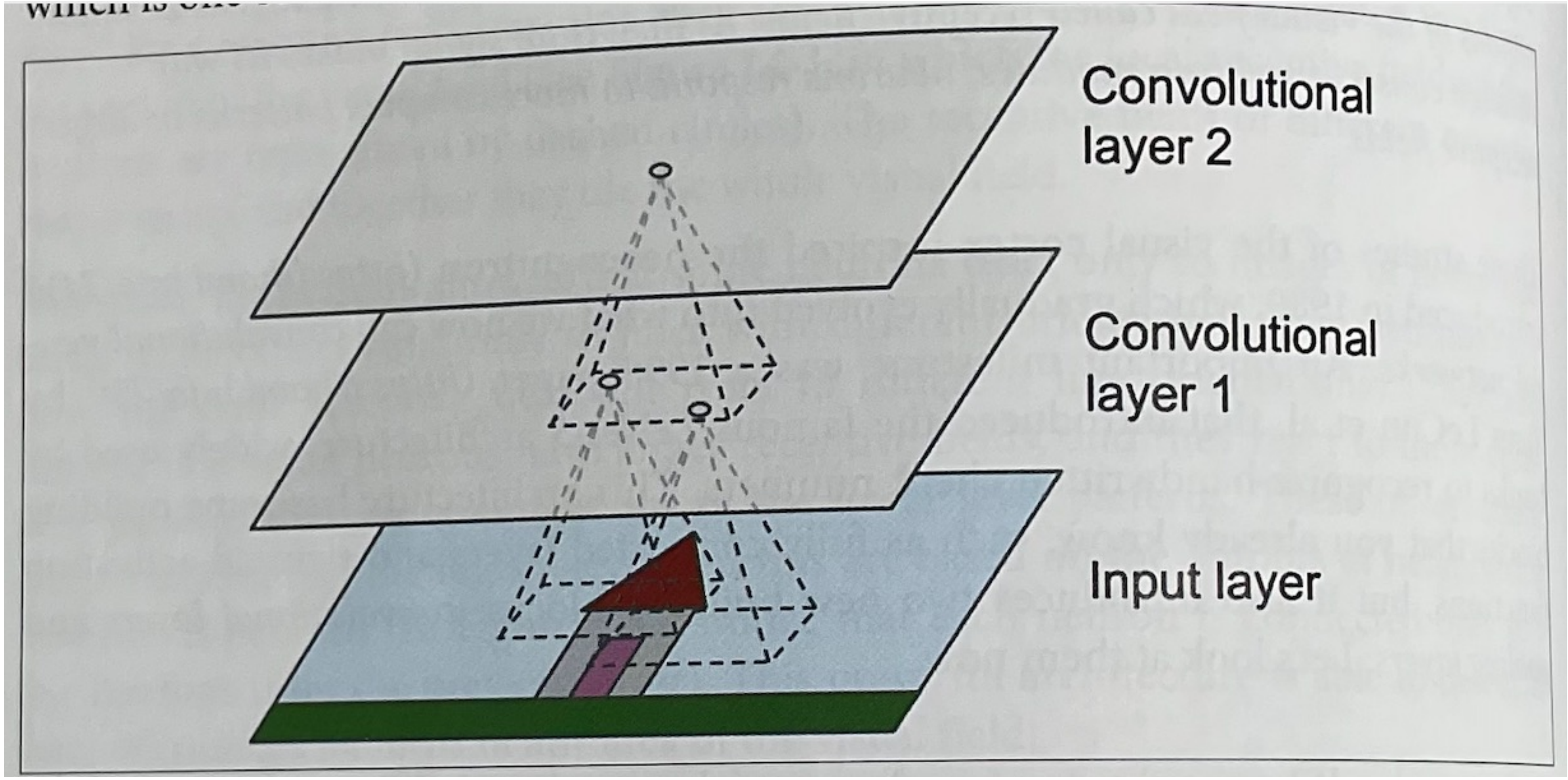
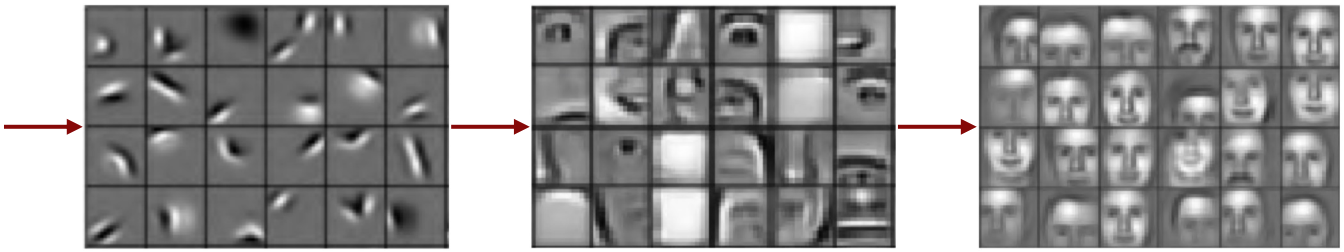


Image credit: Fig 14-2 from https://www.amazon.com/Hands-Machine-Learning-Scikit-Learn-TensorFlow/dp/1492032646

As a result, *stacked* convolutional layers can learn increasingly complex features



lines => edges, circles => faces!

Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations by Lee at al (2009)

Pooling Layers

Pooling Layers

Pooling layers (also called down-sampling or subsampling layers) reduce the size of the tensor coming out of a convolutional layer

Pooling Layers

Pooling layers (also called down-sampling or subsampling layers) reduce the size of the tensor coming out of a convolutional layer

23	14	0	109
9	43	58	15
105	85	35	25
24	69	17	7

2x2 max pooling →

43	109
105	35

- In *max-pooling*, we take the maximum value from each 2x2 box

Pooling Layers

Pooling layers (also called down-sampling or subsampling layers) reduce the size of the tensor coming out of a convolutional layer

23	14	0	109
9	43	58	15
105	85	35	25
24	69	17	7

2x2 average pooling

22.2	45.5
70.75	21

- In *max-pooling*, we take the maximum value from each 2x2 box
- In *average pooling*, we take the average of each 2x2 box

Pooling Layers

Pooling layers (also called down-sampling or subsampling layers) reduce the size of the tensor coming out of a convolutional layer

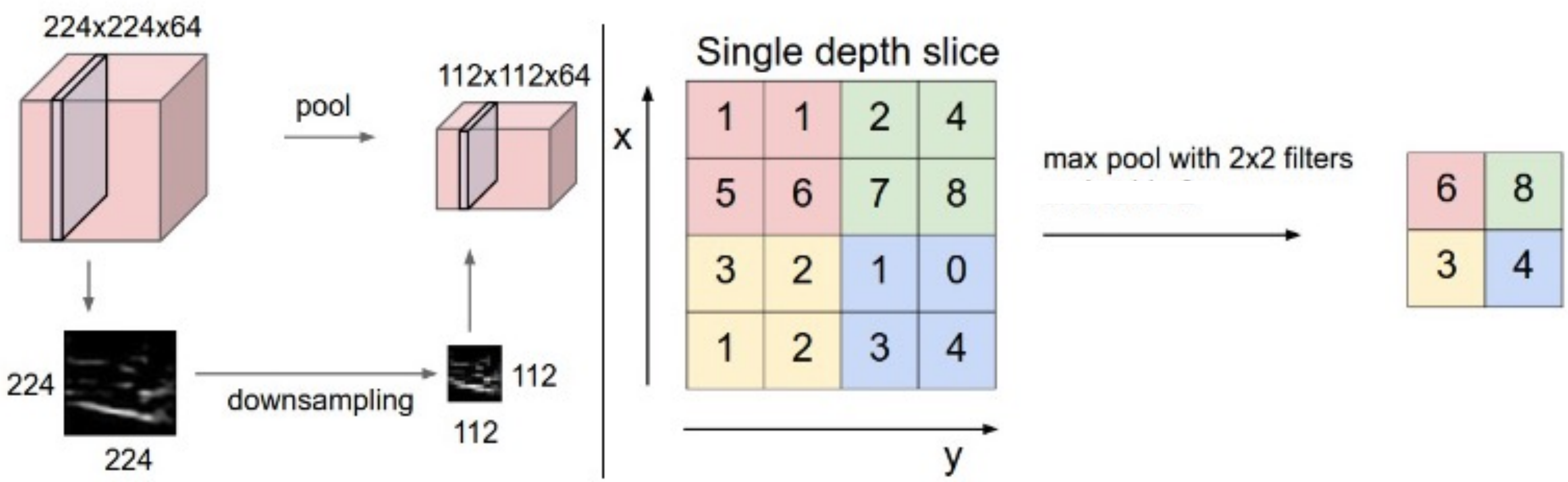
23	14	0	109
9	43	58	15
105	85	35	25
24	69	17	7

2x2 average pooling →

22.2	45.5
70.75	21

- Reduces the number of entries significantly (e.g., 75% for 2x2 pooling)
- The output from the pooling layer is fed to the next layer as usual

Pooling Layers



https://cs231n.github.io/convolutional-networks/#conv

Intuition behind max pooling

- Max pooling acts like an “OR” condition: if a feature exists anywhere in its input, max-pooling will pick it up i.e., max-pooling acts like a *feature detector*

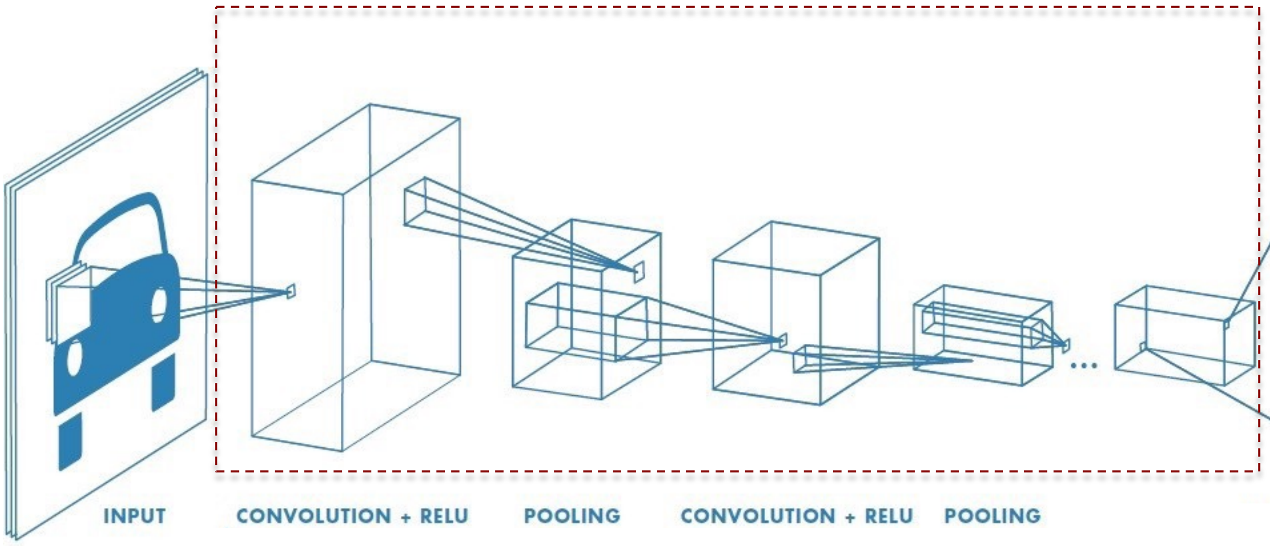
Intuition behind max pooling

- Max pooling acts like an "OR" condition: if a feature exists anywhere in its input, max-pooling will pick it up i.e., max-pooling acts like a *feature detector*

- Since successive convolutional layers can "see" more and more of the original input image, the max-pooling layers that follow them can detect if a feature exists in more and more of the original input image as well

The Architecture of a Basic CNN

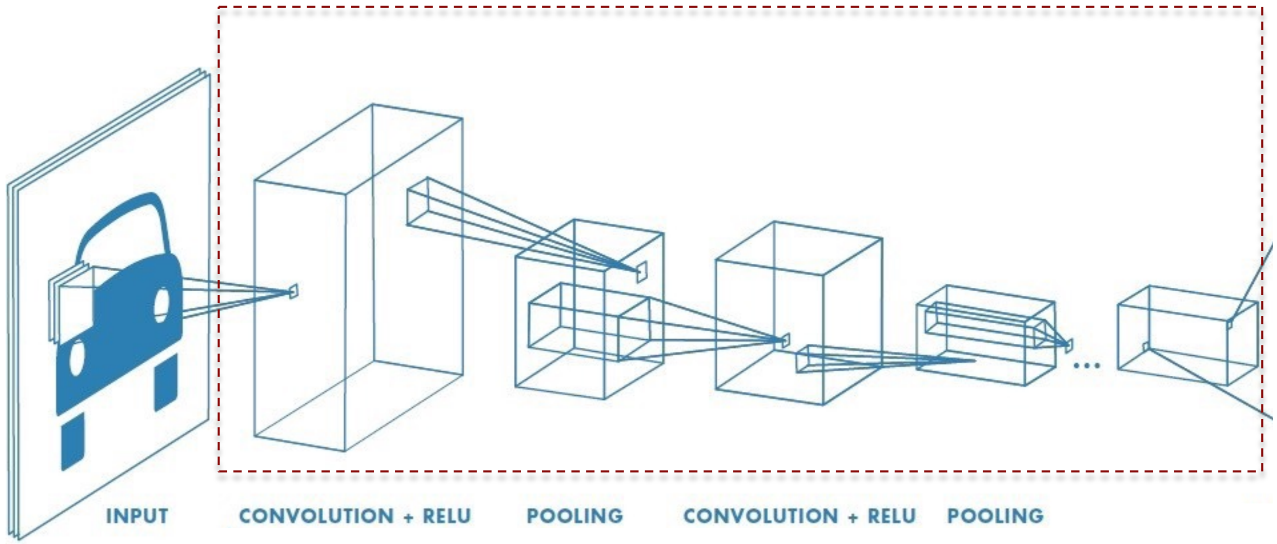
The architecture of a basic CNN



A series of convolutional blocks

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

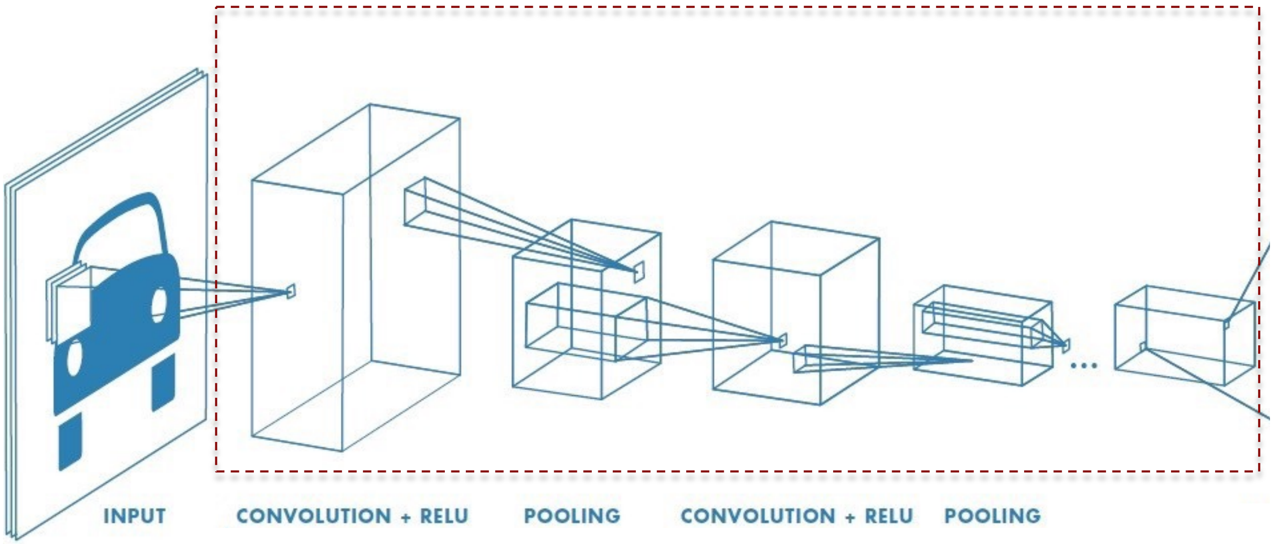
The architecture of a basic CNN



Each *convolutional block* *typically has* 1-2 convolutional layers followed by a pooling layer

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

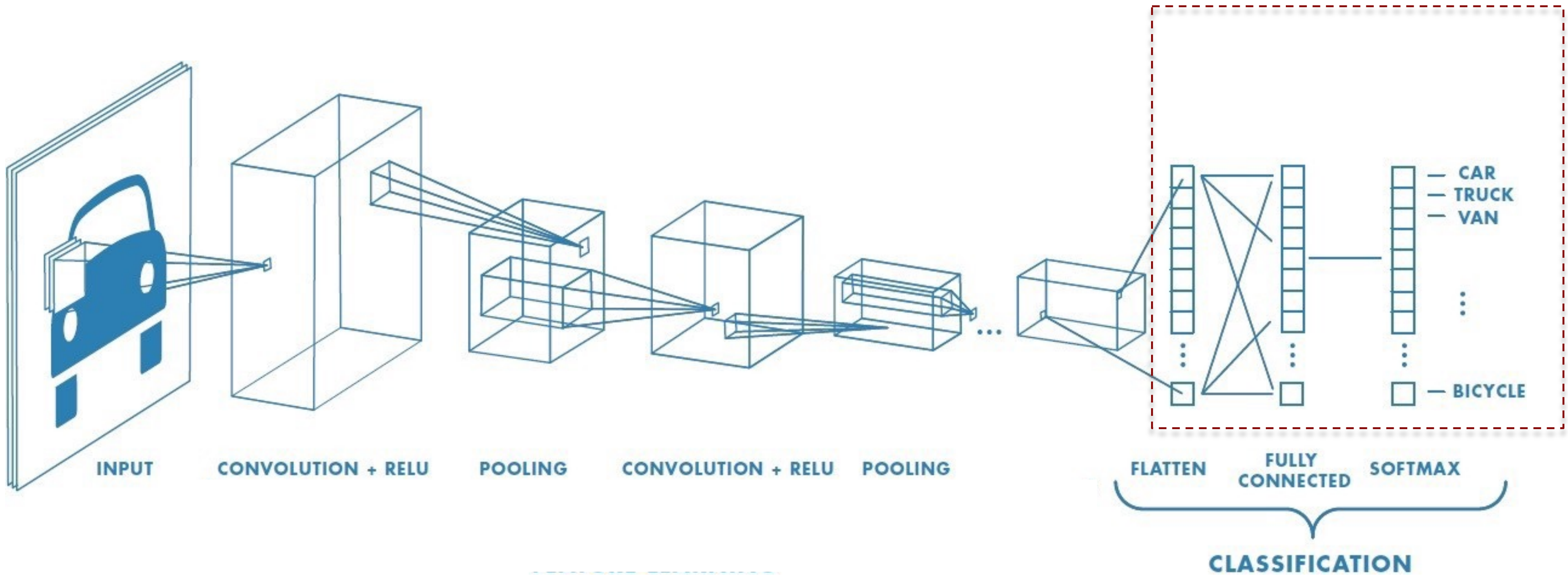
The architecture of a basic CNN



Each block will typically have more depth than the previous block but lower height/width

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

At the end, we flatten the tensor, run it through fully-connected layers, and then the output layer



The final tensor gets flattened into a long vector and sent through 0 or more hidden layers to the output layer

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Colab: Let's solve Fashion MNIST with Convolutional layers!

Next: We will work with <u>color</u> images

Motivating application: A Handbags-Shoes Classifier based on less than 100 images!

Web-scraped dataset of < 100 color images of handbags and shoes

With this tiny dataset, we will build a deep learning network to classify <u>your</u> shoe or handbag <u>in class</u> with high accuracy!

Colab: Let’s build a shoes-handbags classifier with Convolutional layers!

Can we do better? We only have 100 examples of each class

Transfer Learning with Pre-trained Networks

Transfer learning takes advantage of two research trends

Trend 1: Researchers have designed NN architectures that are well-matched to different *types* of data. For example:

Type of data	**Architecture**
All	Residual connections
Images	Convolutional layers
Sequences (e.g., natural language, audio, video, gene sequences)	Transformers

Transfer learning takes advantage of two research trends

Trend 1: Researchers have designed NN architectures that are well-matched to different *types* of data. For example:

Type of data	Architecture
All	Residual connections
Images	Convolutional layers
Sequences (e.g., natural language, audio, video, gene sequences)	Transformers

Trend 2: Using these architectural innovations, researchers have trained high-performance DNNs on a variety of **large** real-world datasets. Numerous pretrained models are available!

Transfer learning involves **customizing** such a pre-trained network to your problem, rather than designing and training a network from scratch.

Can we apply Transfer Learning to build a better Handbags/Shoes Classifier?*

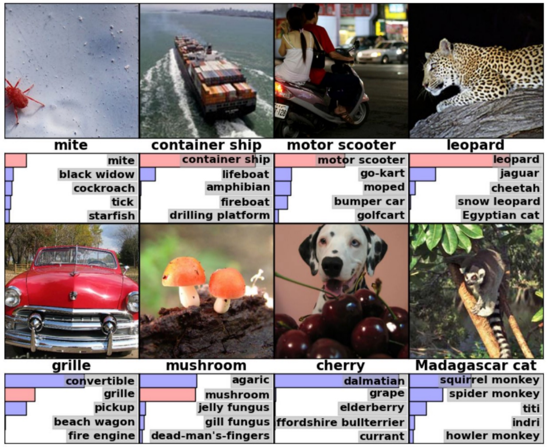
Handbags and shoes are "everyday objects" and you can look around and see if there are any networks that have been trained on a dataset of images of "everyday objects"

*The first thing to do, of course, is to see if someone has already built such a classifier and put it on GitHub ☺

Remember this from lecture 1?

ImageNet Challenge

- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.

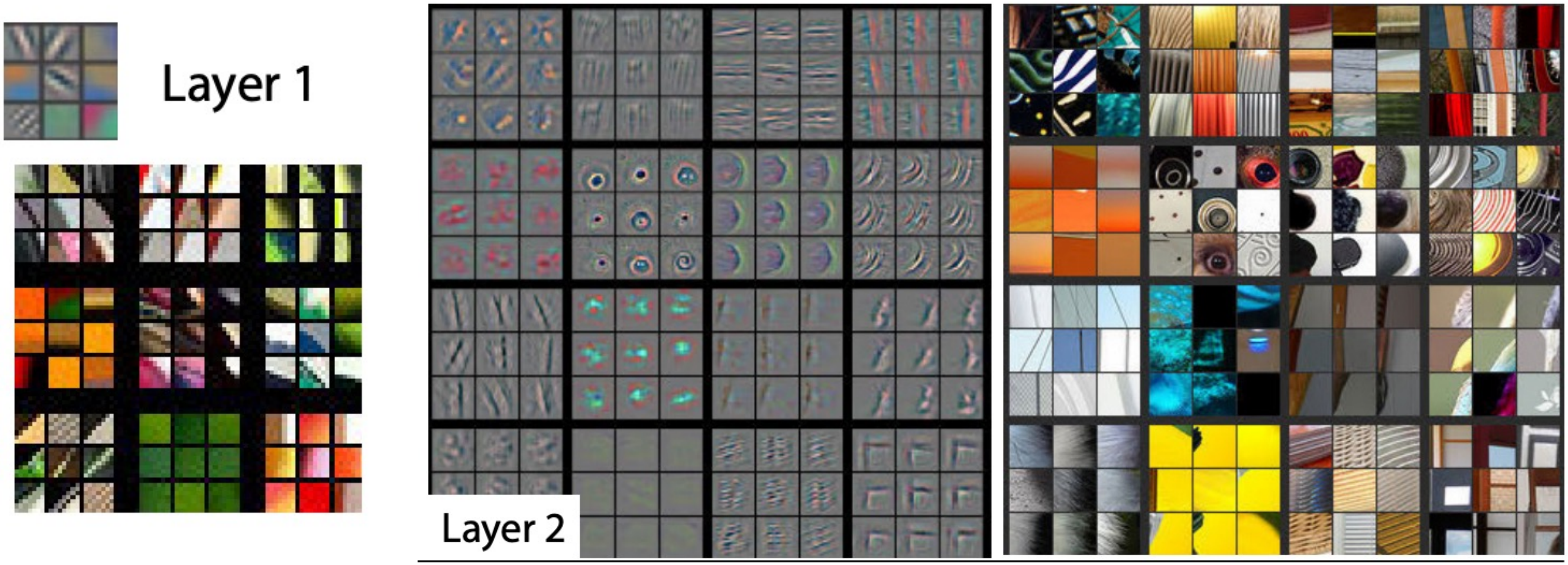


https://www.slideshare.net/xavigiro/image-classification-on-imagenet-d1l4-2017-upc-deep-learning-for-computer-vision/

The ImageNet dataset has millions of images of everyday objects

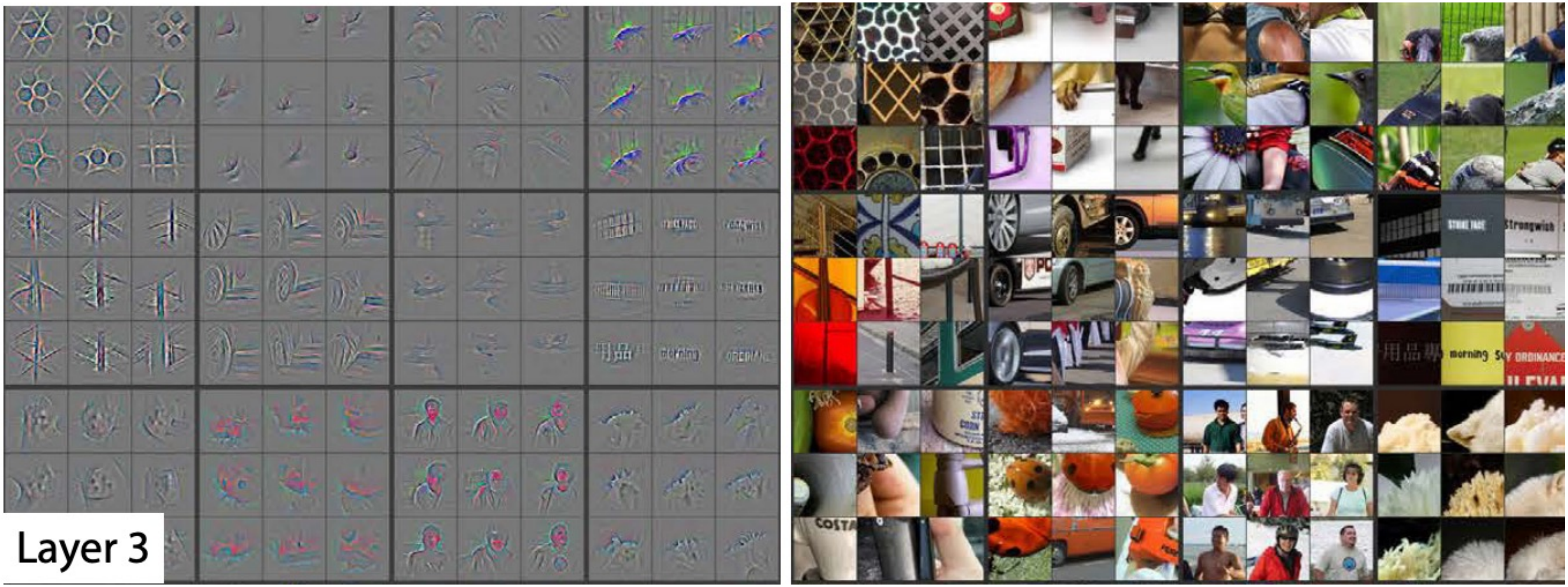
We can look for networks that have been trained on ImageNet

A network trained on ImageNet and that does well on it has essentially developed a smart, hierarchical representation of these objects across its layers



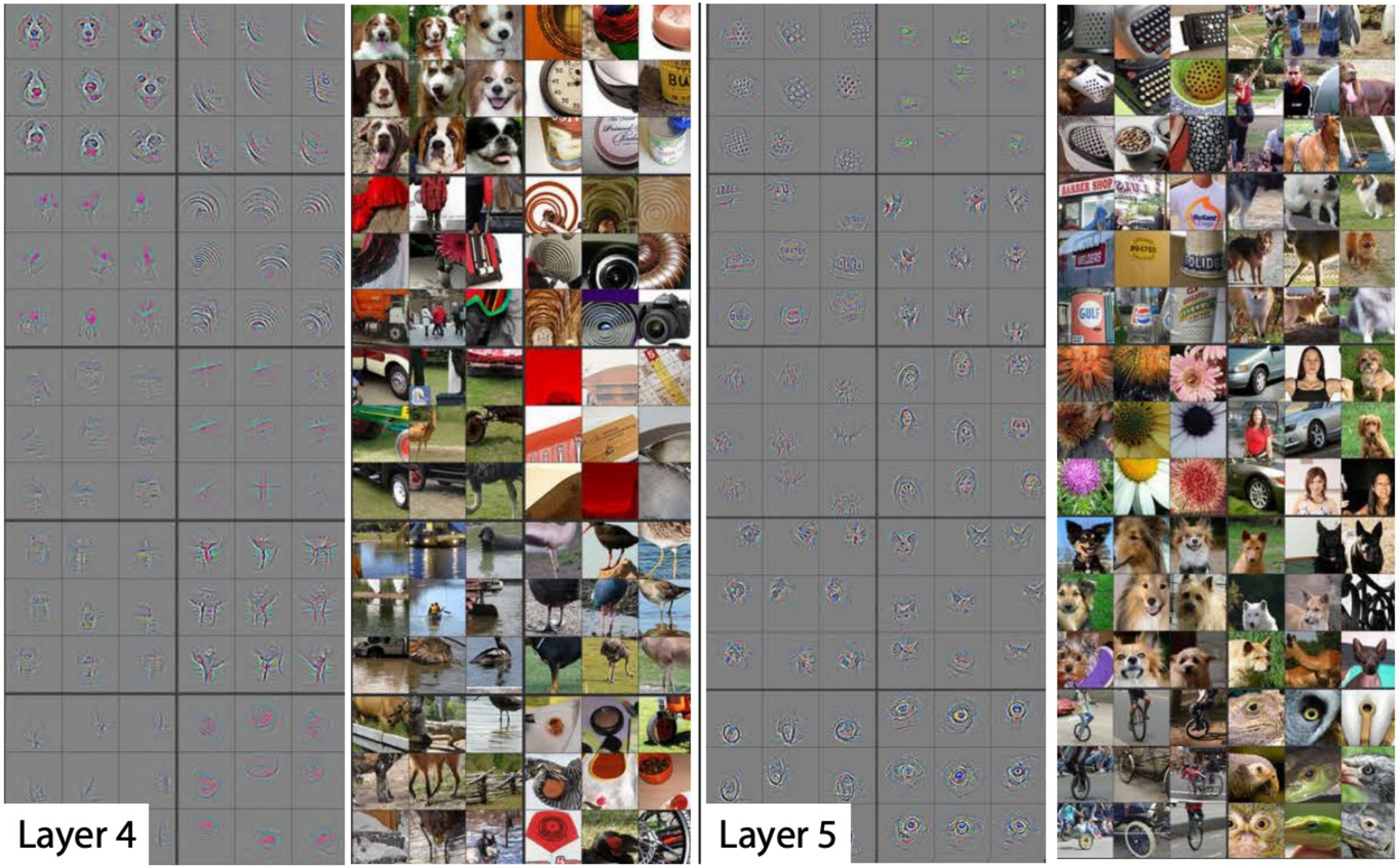
https://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf

A network trained on ImageNet and that does well on it has essentially developed a smart, hierarchical representation of these objects across its layers



https://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf

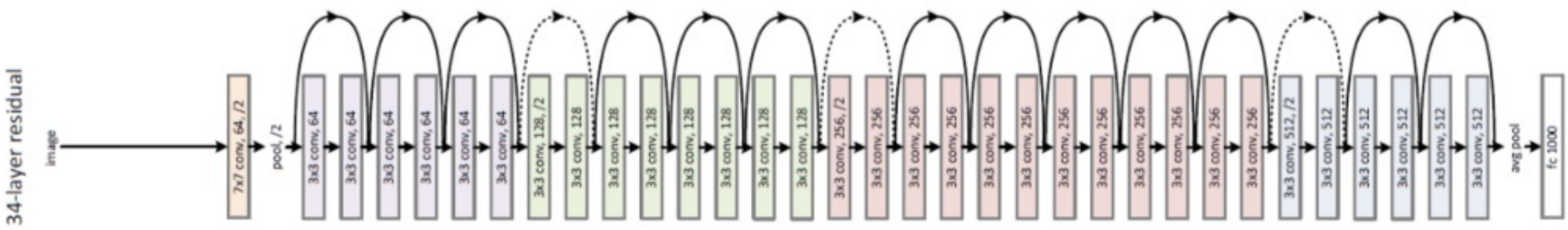
A network trained on ImageNet and that does well on it has essentially developed a smart, hierarchical representation of these objects across its layers



https://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf

The ResNet family of networks is famous for doing very well on ImageNet

ResNet 34*

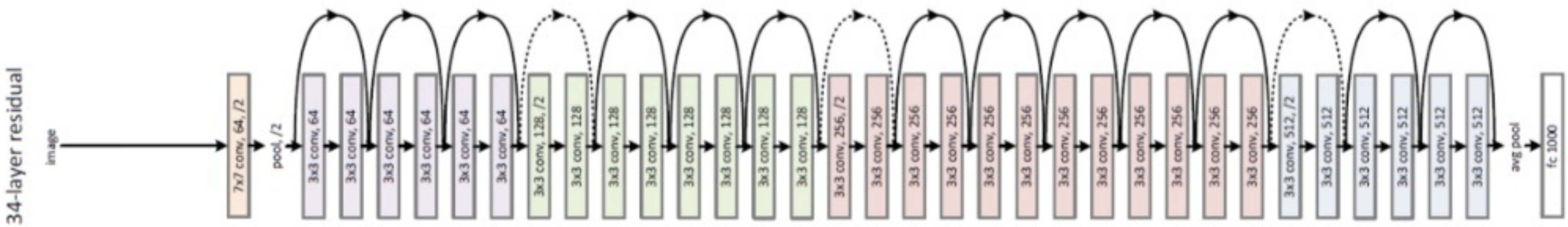


We'd expect the (learned) weights and biases of ResNet layers to embody "knowledge" (like illustrated in the previous slides) about the characteristics of the millions of everyday images that the network was trained on

*https://arxiv.org/abs/1512.03385

Can we use ResNet as is?

ResNet 34*



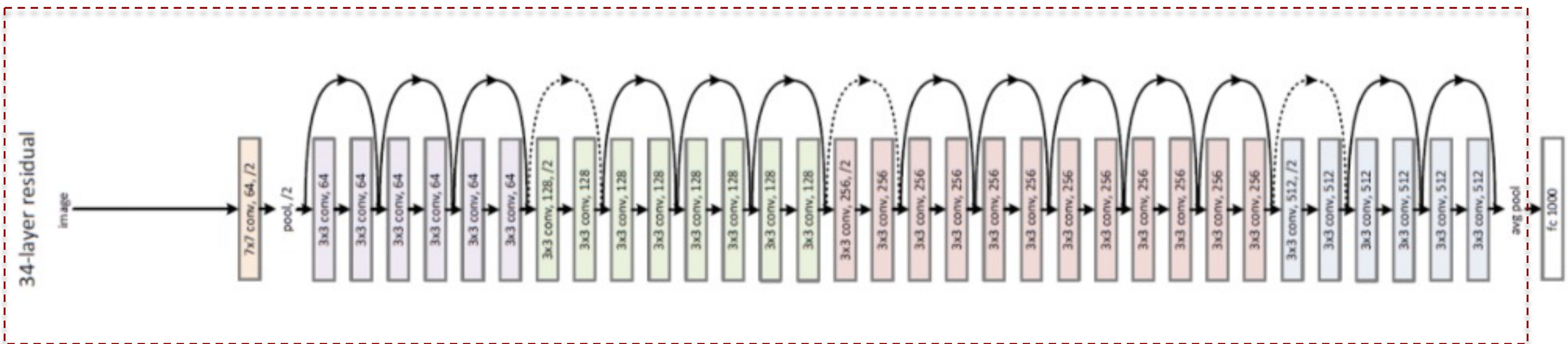
Not really.

Remember that ResNet was designed to classify the image into 1000 categories. Our problem is different – we only care about two categories: handbags and shoes.

*https://arxiv.org/abs/1512.03385

But what if we take ResNet and stop just before the last layer?

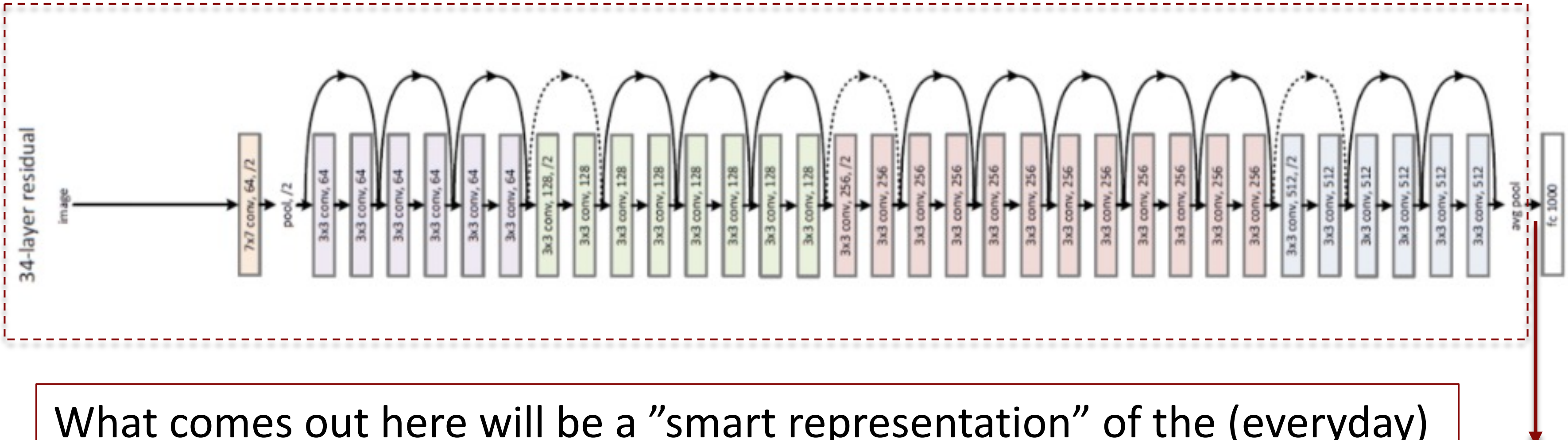
ResNet 34



*https://arxiv.org/abs/1512.03385

We can run our images through this “headless” ResNet

ResNet 34



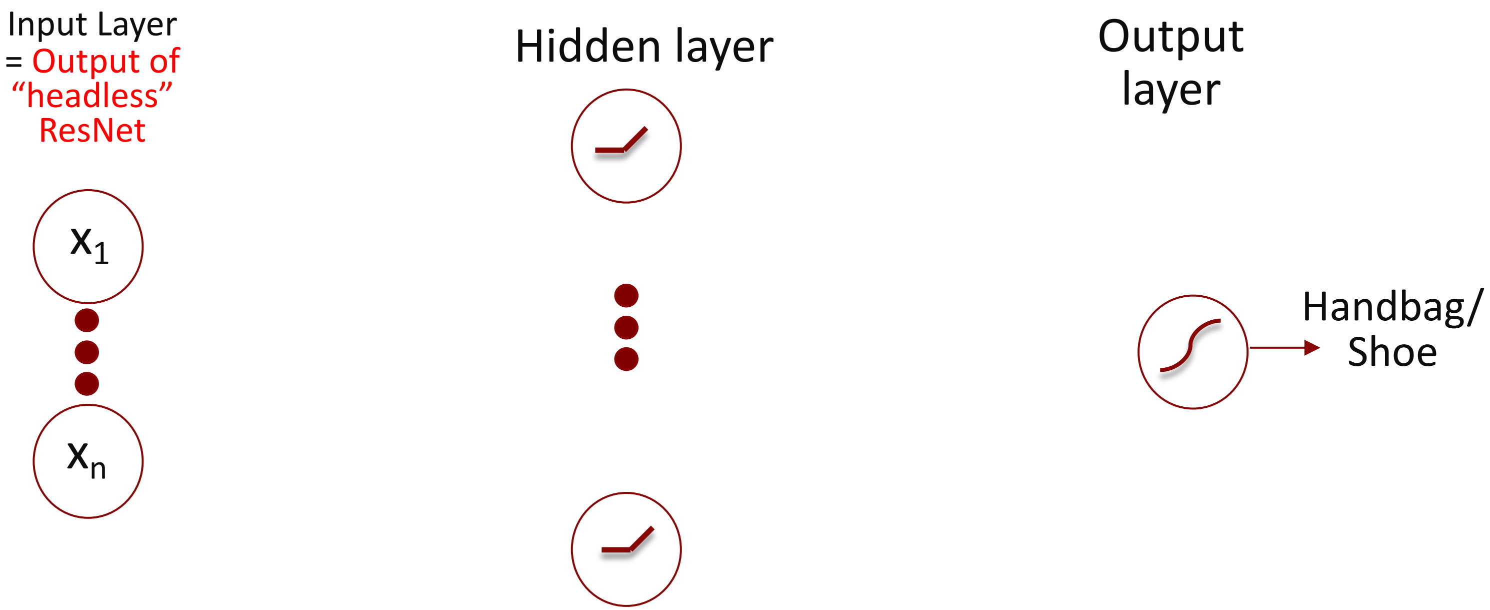
What comes out here will be a ”smart representation” of the (everyday) image that can be used for different kinds of categorization.

Imagine that the image gets ”tagged” with lots of general tags and we can use the tags any way we wish.

*https://arxiv.org/abs/1512.03385

We can think of the output of “headless ResNet” as our transformed smart input and use it to train a simple one-hidden-layer NN

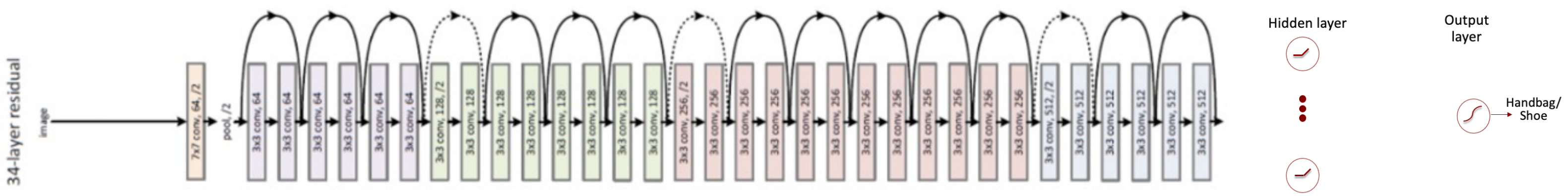
We can think of the output of "headless ResNet" as our transformed smart input and use it to train a simple one-hidden-layer NN



Since the inputs to the hidden layer are at a higher level of abstraction, it may be able to learn to classify handbags from shoes with very few examples

That's the basic idea behind transfer learning …

That's the basic idea behind transfer learning but you can get fancier

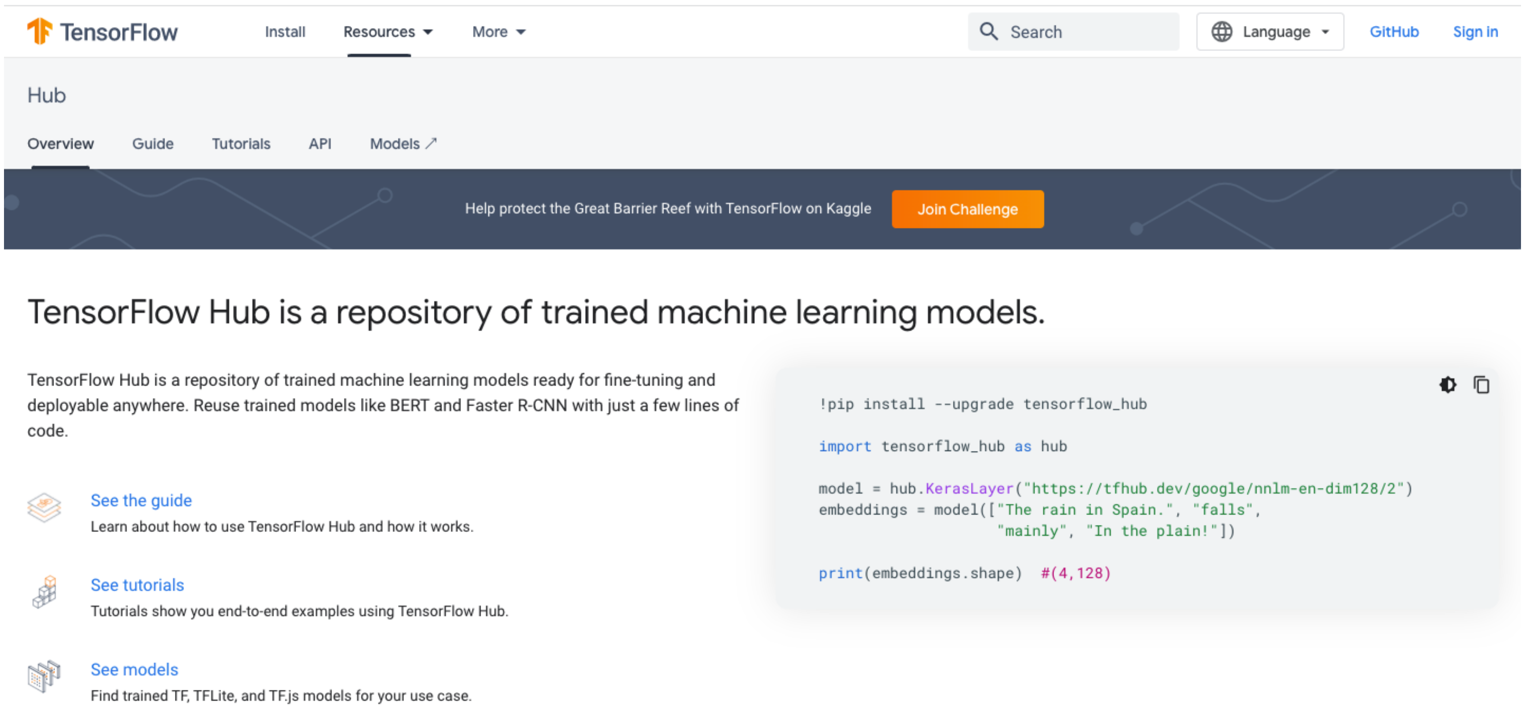


- You can connect up "headless ResNet" with our little network and train the entire network end-to-end. This is called <u>fine-tuning</u>.

- However, you MUST start the training with the weights and biases that came with ResNet, rather than start from scratch.

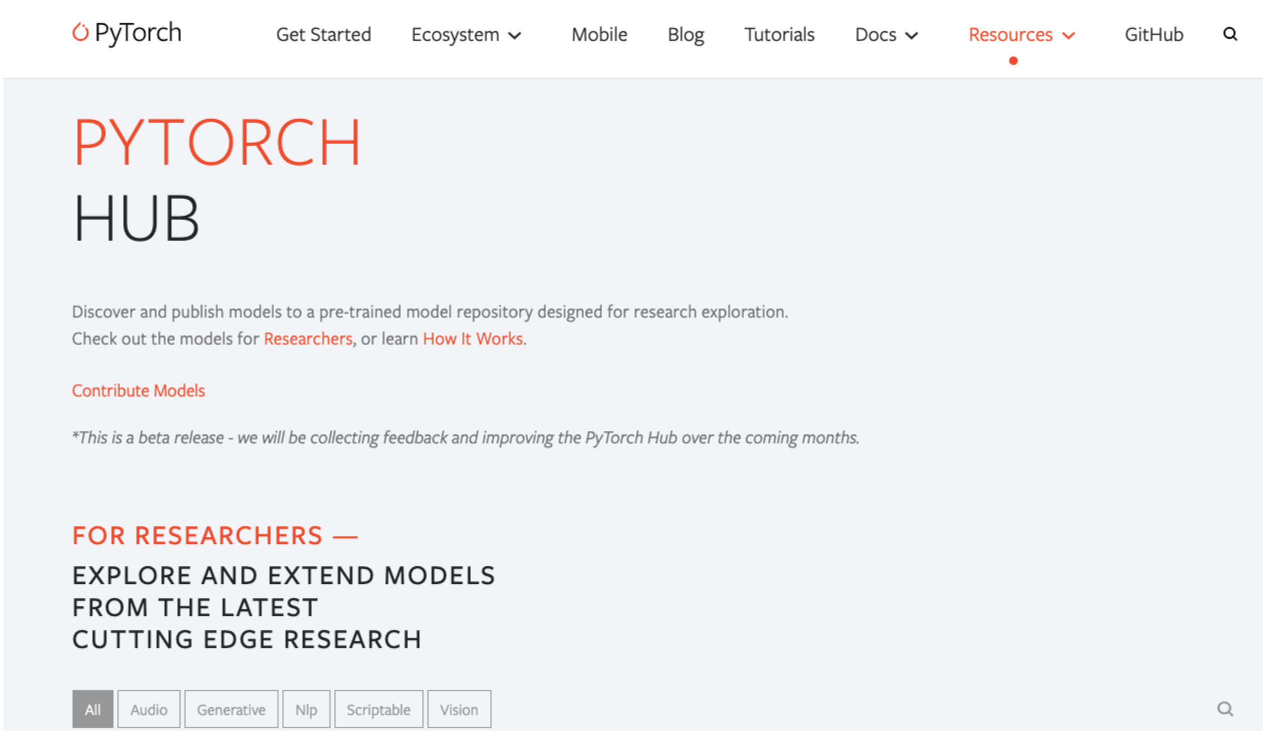
- You will explore this in HW 1

You can find pretrained models in the Tensorflow Hub …



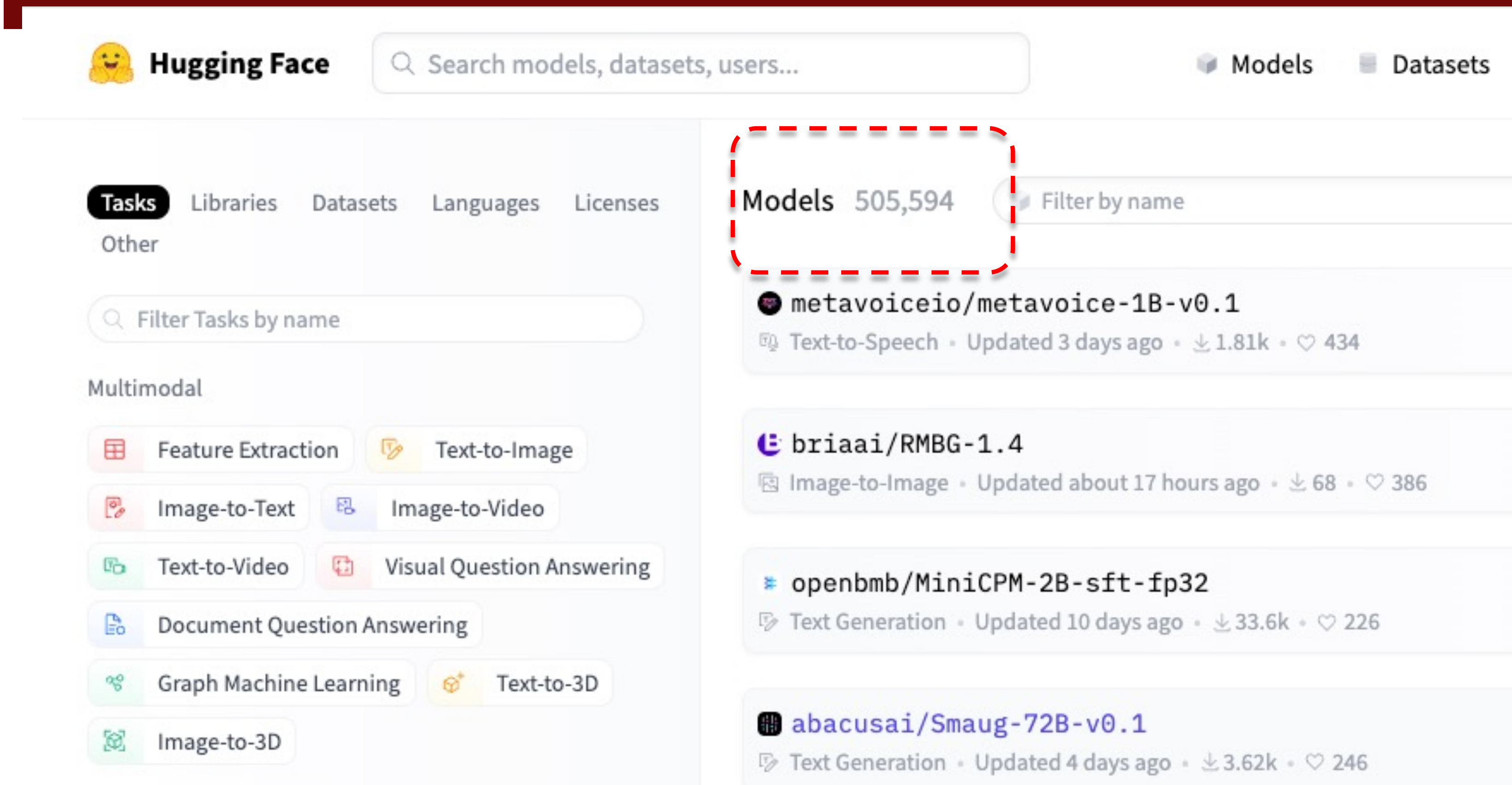
https://www.tensorflow.org/hub

… the PyTorch Hub, …



https://pytorch.org/hub/

… and the Hugging Face Hub



https://huggingface.co/models (as of February 13, 2024)

Back to Colab

Appendix

The convolutional filter is just a slightly modified neuron

- A “traditional” neuron (in the first hidden layer) is connected to all pixels of the input image. In contrast, a “convolutional filter” neuron is connected only to pixels in a small *region* of the input image
- Sliding the filter across the image ⇔ can be thought of as a different filter for each window *but with the same weights*
- Benefits
  - Preserves local adjacency
  - Far fewer parameters
  - Translation invariance – can detect the same pattern regardless of where it appears in an image