

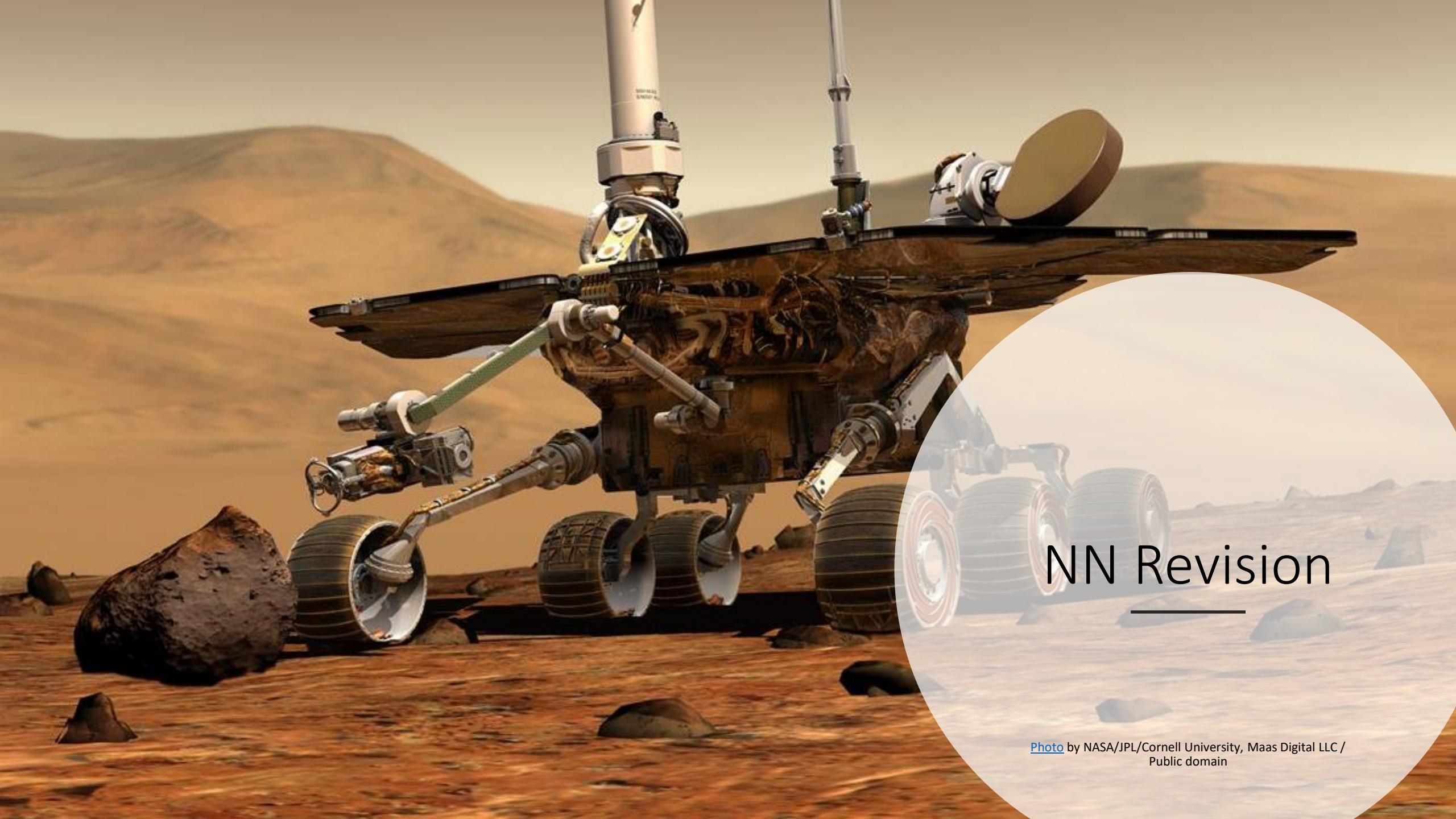
Introduction to Convolutional Neural Networks

[Photo](#) by NASA/JPL/Cornell University, Maas Digital LLC /
Public domain

Outline

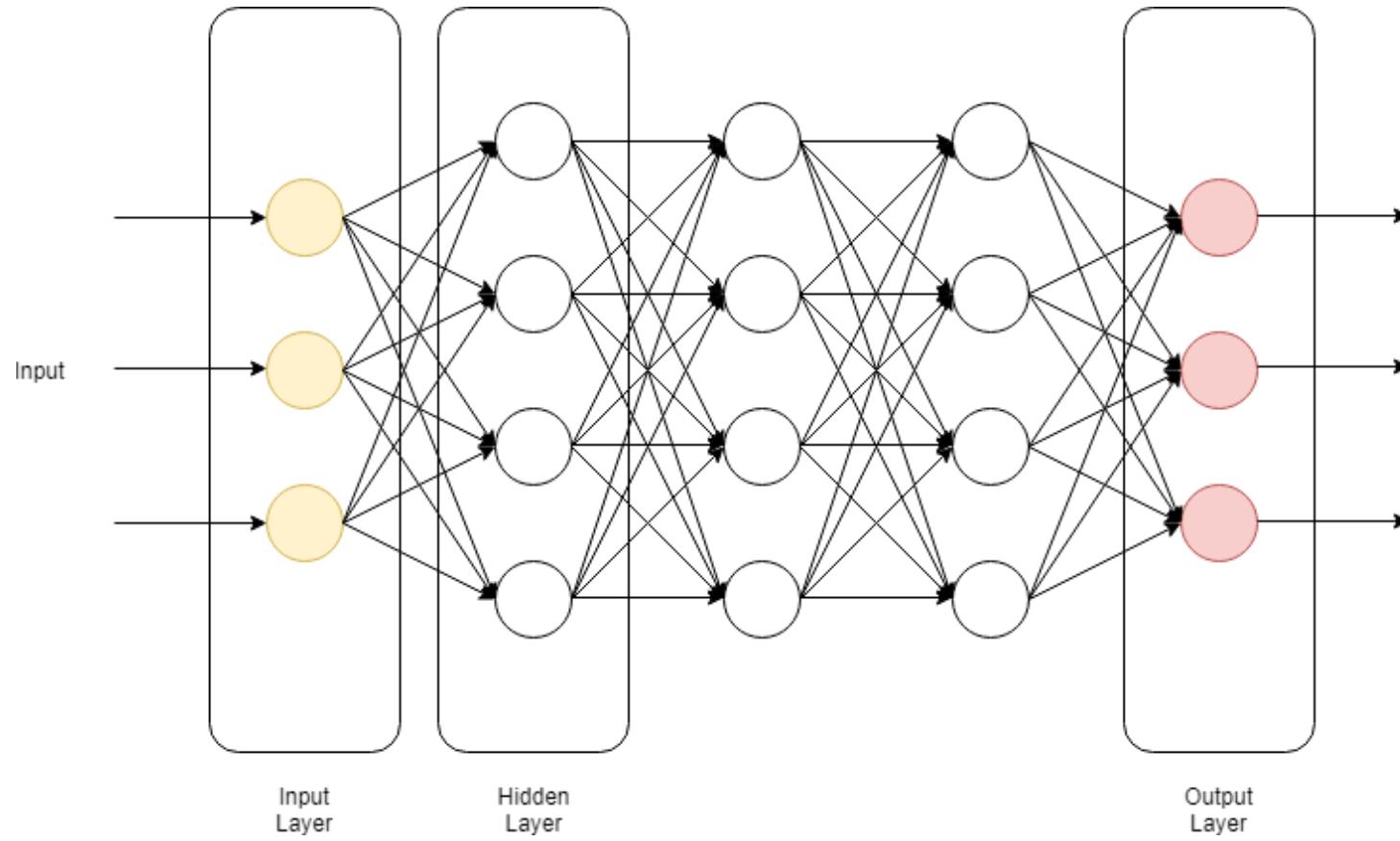
- NN revision
- CNN Visualisation - How does computer sees?
- Motivational Example
- Understanding Convolution
- Train Your Own CNN
- Transfer Learning
- Visualising Network Layers
- Where to From Here?





NN Revision

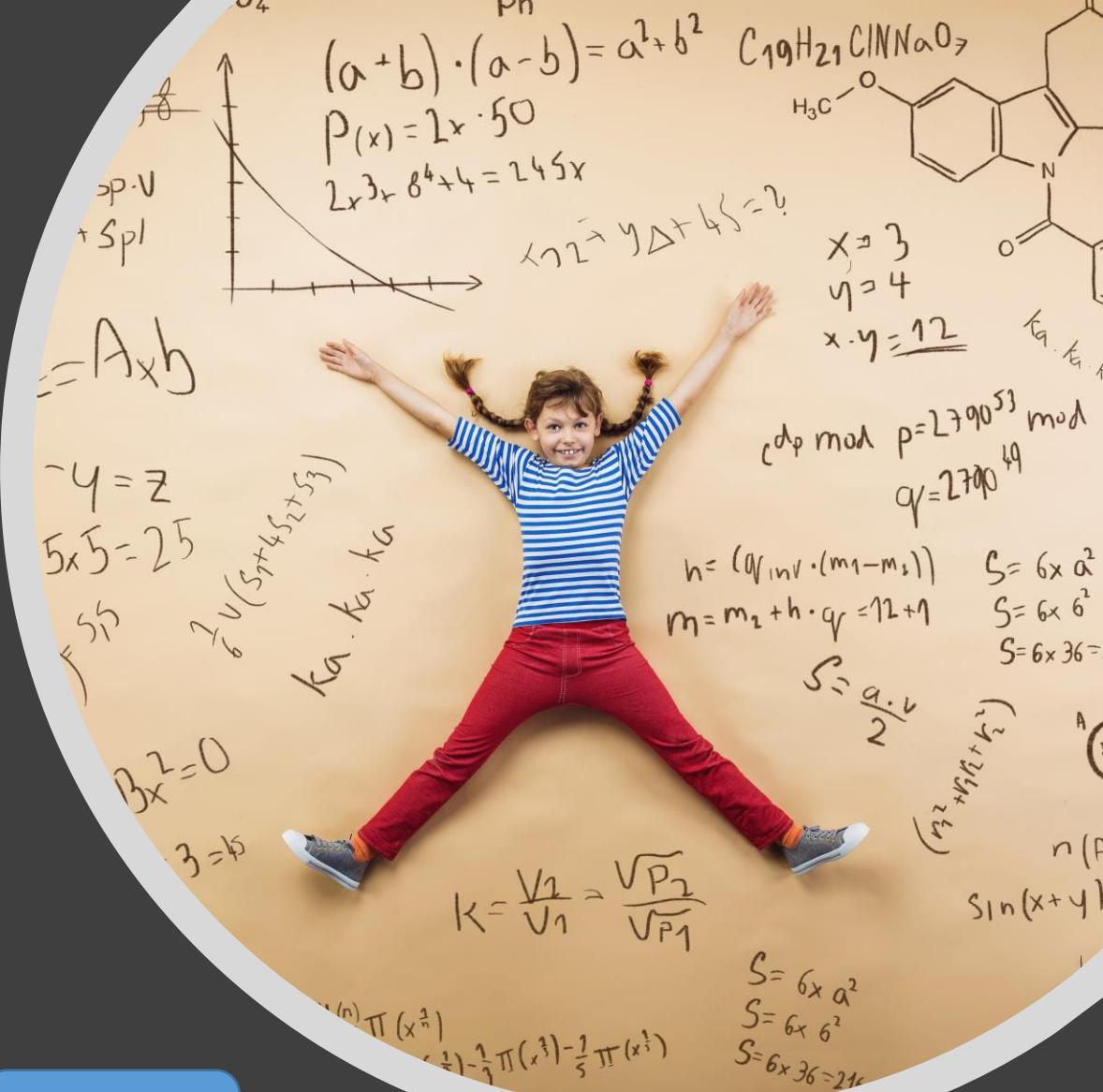
[Photo](#) by NASA/JPL/Cornell University, Maas Digital LLC /
Public domain



Example of a Neural Network

What have we learnt so far?

- Machine learning is about
 - Input – ??? – Targets
 - Finding the ??? to link inputs to target
- Required many examples to achieve this.
- From the previous slides we can see that there are many layers to deep learning neural networks.
- These layers learn the relationship between input and target from being exposed to many examples

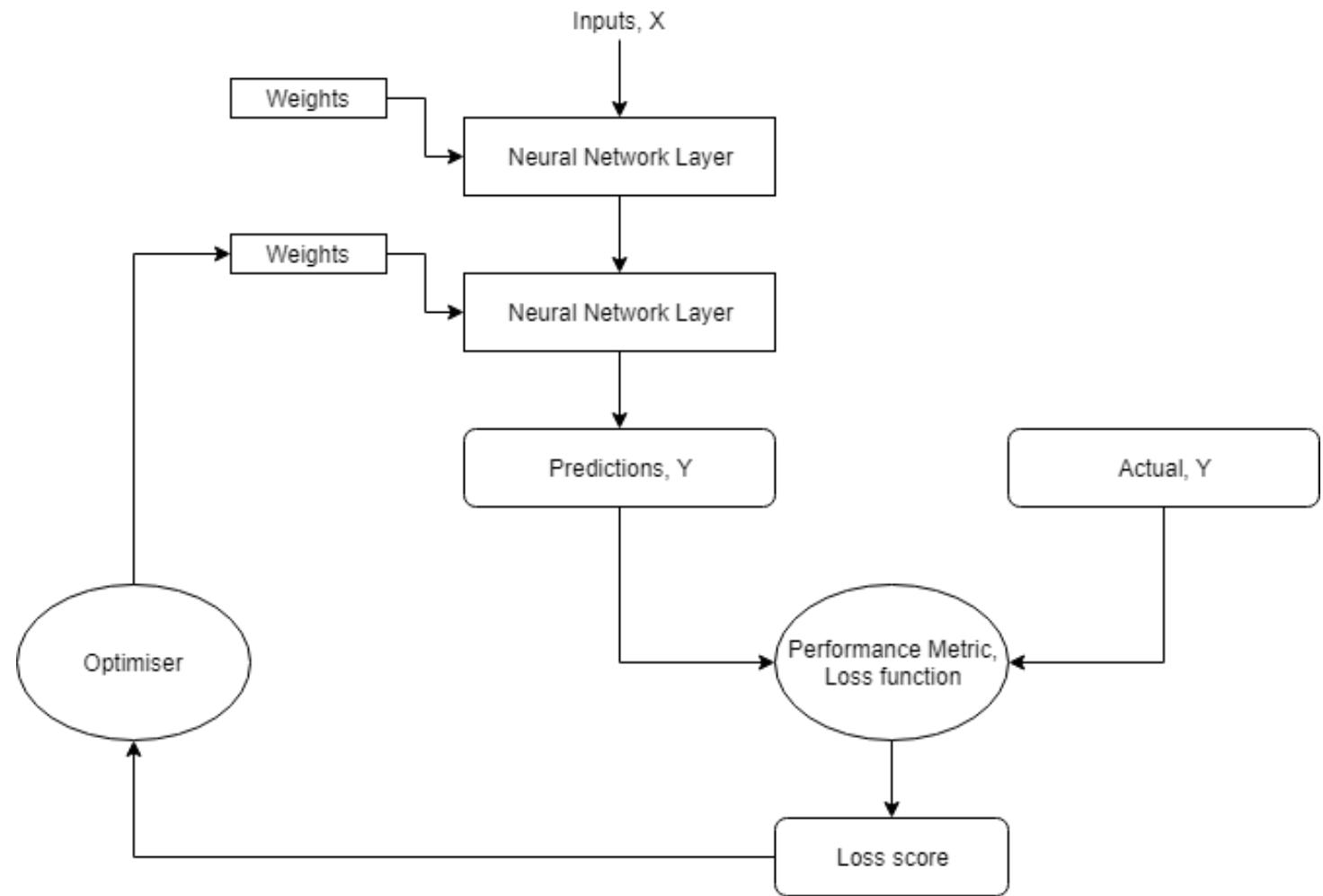


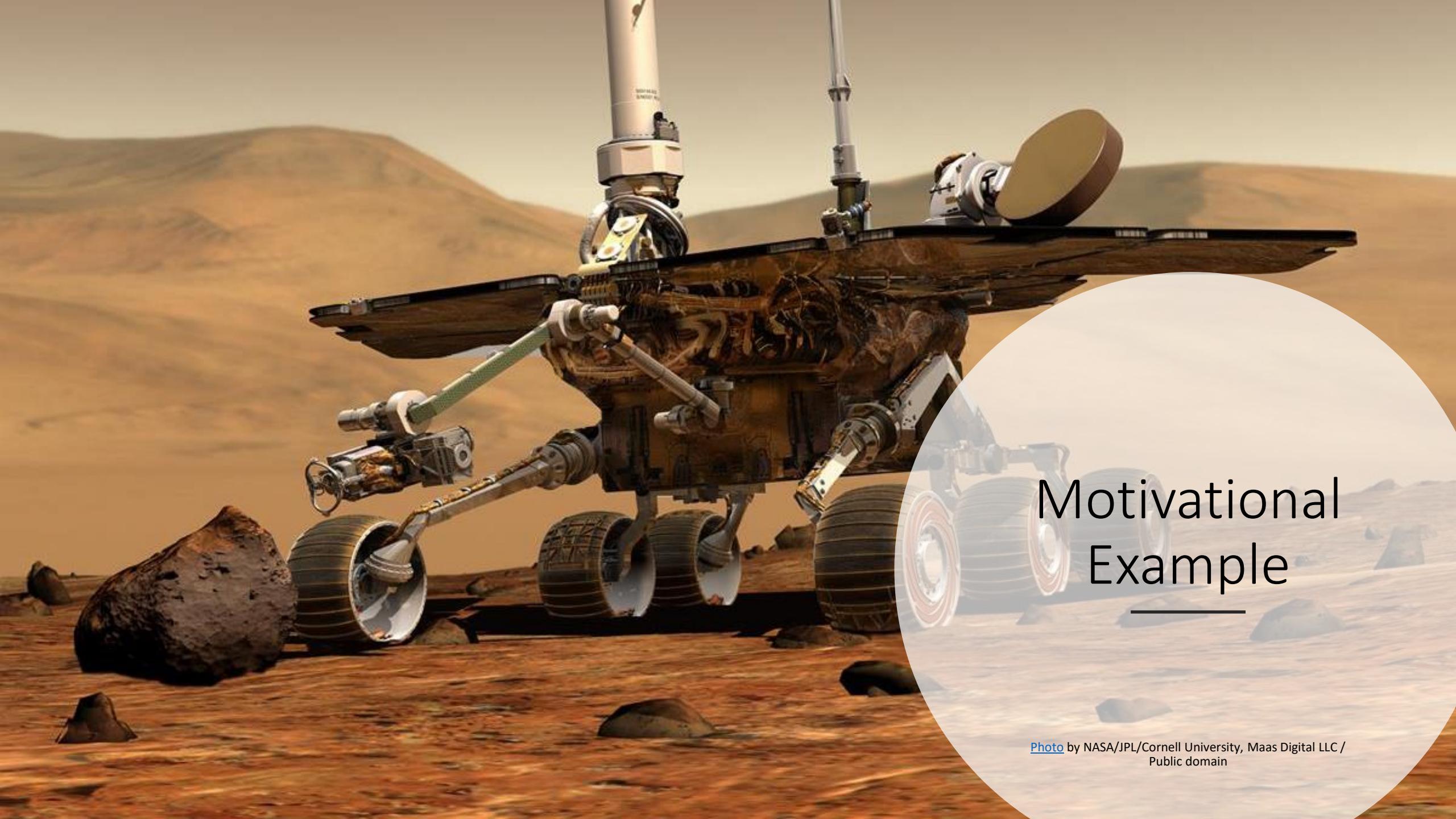
VGG16

- Example of a NN architecture
- [VGG16](#)
- NB:
 - Fully connected layer not shown

Layer (type)	Output Shape	Param #
<hr/>		
input_1 (InputLayer)	(None, None, None, 3)	0
block1_conv1 (Conv2D)	(None, None, None, 64)	1792
block1_conv2 (Conv2D)	(None, None, None, 64)	36928
block1_pool (MaxPooling2D)	(None, None, None, 64)	0
block2_conv1 (Conv2D)	(None, None, None, 128)	73856
block2_conv2 (Conv2D)	(None, None, None, 128)	147584
block2_pool (MaxPooling2D)	(None, None, None, 128)	0
block3_conv1 (Conv2D)	(None, None, None, 256)	295168
block3_conv2 (Conv2D)	(None, None, None, 256)	590080
block3_conv3 (Conv2D)	(None, None, None, 256)	590080
block3_pool (MaxPooling2D)	(None, None, None, 256)	0
block4_conv1 (Conv2D)	(None, None, None, 512)	1180160
block4_conv2 (Conv2D)	(None, None, None, 512)	2359808
block4_conv3 (Conv2D)	(None, None, None, 512)	2359808
block4_pool (MaxPooling2D)	(None, None, None, 512)	0
block5_conv1 (Conv2D)	(None, None, None, 512)	2359808
block5_conv2 (Conv2D)	(None, None, None, 512)	2359808
block5_conv3 (Conv2D)	(None, None, None, 512)	2359808
block5_pool (MaxPooling2D)	(None, None, None, 512)	0
<hr/>		
Total params: 14,714,688		
Trainable params: 14,714,688		
Non-trainable params: 0		

Neural Network





Motivational Example

[Photo](#) by NASA/JPL/Cornell University, Maas Digital LLC /
Public domain

Revisit Motivational Example 1

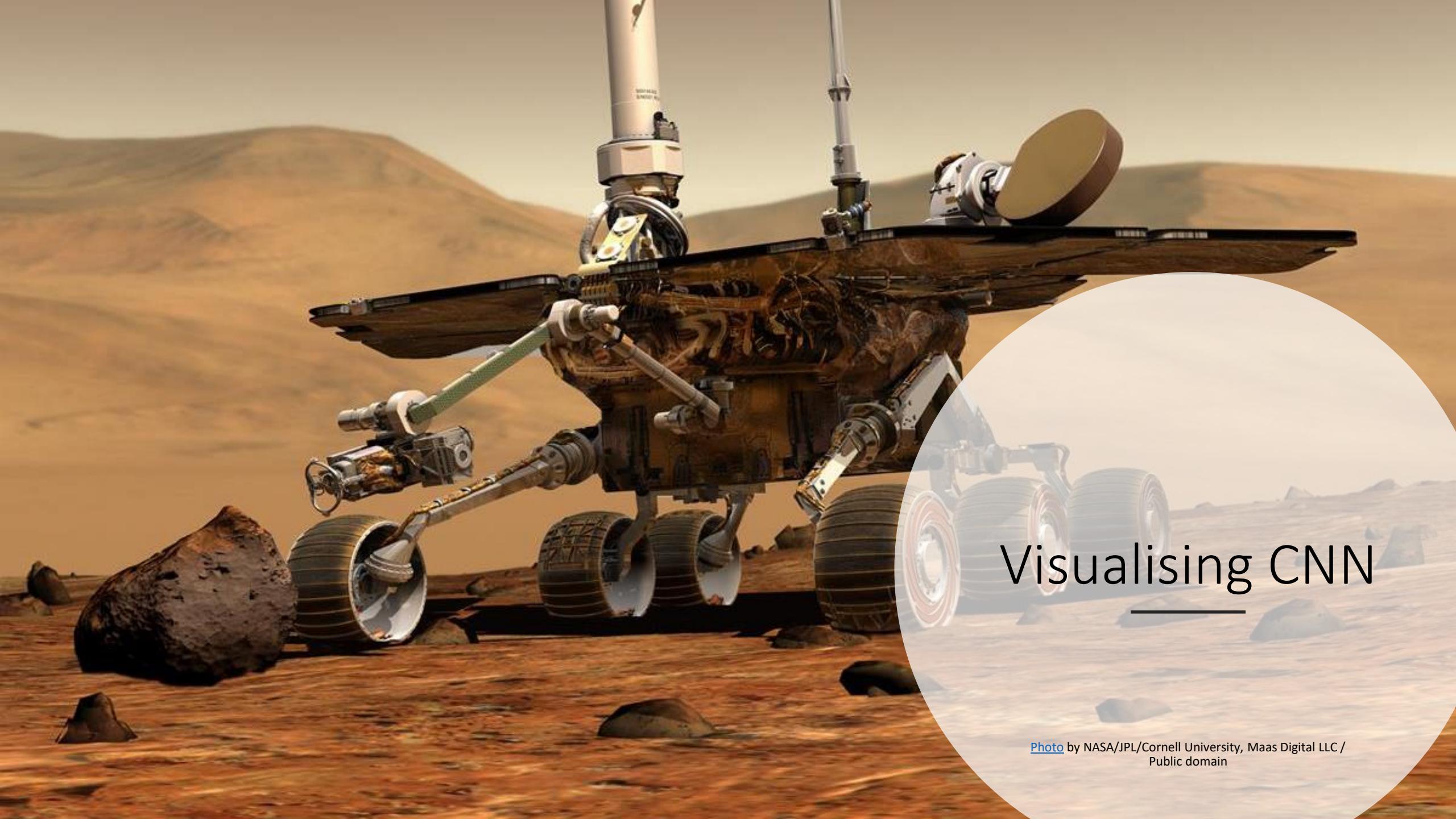
- ANN Model
- Motivational Example 1.ipynb





Motivational Example 2

- ANN vs CNN
- Motivational Example
2.ipynb



Visualising CNN

[Photo](#) by NASA/JPL/Cornell University, Maas Digital LLC /
Public domain

VGG16

- Example of a NN architecture
- [VGG16](#)
- NB:
 - Fully connected layer not shown

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, None, None, 3)	0
block1_conv1 (Conv2D)	(None, None, None, 64)	1792
block1_conv2 (Conv2D)	(None, None, None, 64)	36928
block1_pool (MaxPooling2D)	(None, None, None, 64)	0
block2_conv1 (Conv2D)	(None, None, None, 128)	73856
block2_conv2 (Conv2D)	(None, None, None, 128)	147584
block2_pool (MaxPooling2D)	(None, None, None, 128)	0
block3_conv1 (Conv2D)	(None, None, None, 256)	295168
block3_conv2 (Conv2D)	(None, None, None, 256)	590080
block3_conv3 (Conv2D)	(None, None, None, 256)	590080
block3_pool (MaxPooling2D)	(None, None, None, 256)	0
block4_conv1 (Conv2D)	(None, None, None, 512)	1180160
block4_conv2 (Conv2D)	(None, None, None, 512)	2359808
block4_conv3 (Conv2D)	(None, None, None, 512)	2359808
block4_pool (MaxPooling2D)	(None, None, None, 512)	0
block5_conv1 (Conv2D)	(None, None, None, 512)	2359808
block5_conv2 (Conv2D)	(None, None, None, 512)	2359808
block5_conv3 (Conv2D)	(None, None, None, 512)	2359808
block5_pool (MaxPooling2D)	(None, None, None, 512)	0
<hr/>		
Total params: 14,714,688		
Trainable params: 14,714,688		
Non-trainable params: 0		

Image Kernels

Explained Visually

[Tweet](#) [Like 903](#) [Share](#)

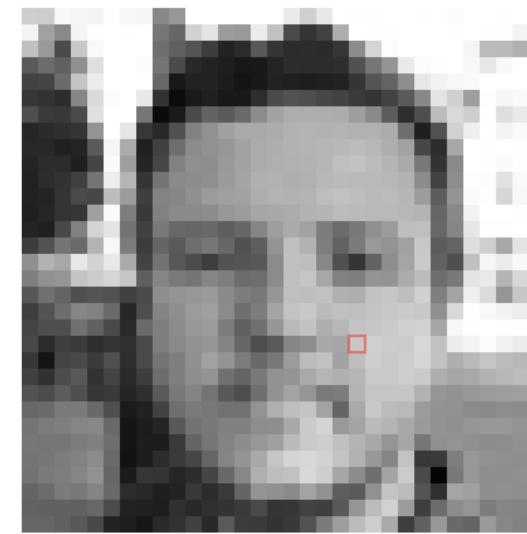
By [Victor Powell](#)

An image kernel is a small matrix used to apply effects like the ones you might find in Photoshop or Gimp, such as blurring, sharpening, outlining or embossing. They're also used in machine learning for 'feature extraction', a technique for determining the most important portions of an image. In this context the process is referred to more generally as "convolution" (see: [convolutional neural networks](#).)

To see how they work, let's start by inspecting a black and white image. The matrix on the left contains numbers, between 0 and 255, which each correspond to the brightness of one pixel in a picture of a face. The large, granulated picture has been blown up to make it easier to see; the last image is the "real" size.

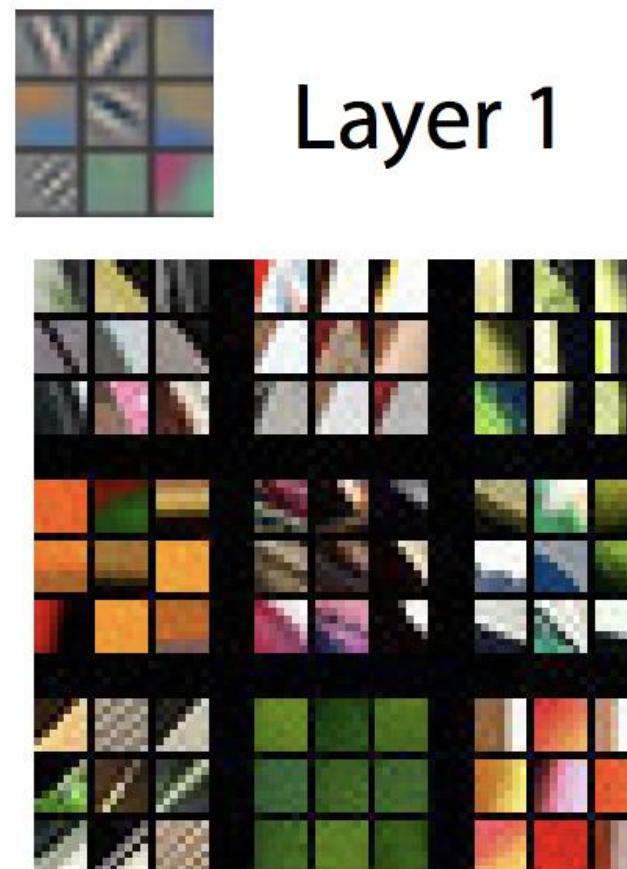
Image Kernels

208 205 247 245 244 253 247 245 136 151 255 255 255 255 255 255 234 207 231 255 254 254 255 255 255 255 255 255 255 255 247
244 161 137 244 254 255 254 255 118 103 209 228 155 153 236 193 74 52 66 173 255 254 254 255 255 255 254 254 254 254 244 184
192 154 75 200 249 255
90 196 96 143 223 255
67 69 107 196 236 255
55 51 45 134 218 251 255
79 59 56 75 224 255
38 43 47 52 147 230 229 56 41 81 129 142 180 169 189 172 178 179 179 177 177 172 173 172 172 158 137 92 46 70 187 217 206 254 222 233 255
40 43 33 36 90 245 171 32 65 110 139 145 151 161 171 174 178 179 182 184 187 183 173 182 71 45 161 235 254 255 254 249 255
37 44 44 31 69 250 158 70 129 142 152 161 171 175 177 178 182 191 194 188 180 170 129 31 137 255 254 250 254 254
34 45 51 64 116 237 181 53 116 138 140 142 154 164 176 178 179 177 183 188 185 185 183 178 140 66 141 254 252 225 249 255
34 39 52 74 71 188 156 63 131 134 144 153 162 161 173 179 178 179 189 190 190 185 187 182 156 93 148 250 254 214 247 255
32 38 52 54 159 250 126 57 129 138 138 140 151 158 166 166 171 178 180 187 186 185 185 182 180 102 136 242 255 254 254
36 32 72 129 212 228 115 65 121 104 102 140 94 103 134 158 162 125 128 121 143 155 190 191 191 124 134 230 252 253 255 255
61 82 116 107 179 247 124 60 101 90 111 119 123 81 94 147 191 178 126 98 123 133 147 181 200 92 100 222 207 187 227 215
144 172 231 210 232 170 67 115 88 76 62 63 83 88 139 192 192 135 89 93 99 141 185 201 97 76 192 245 235 248 249
127 149 195 204 213 197 99 133 122 117 132 126 102 139 191 197 167 129 127 148 147 171 188 110 121 228 233 180 215 212
87 112 100 79 85 82 65 75 142 148 151 153 138 125 120 149 191 192 173 175 174 193 195 196 190 200 127 163 239 219 149 196 195
63 83 109 134 129 156 39 78 132 142 155 159 130 111 124 164 195 200 186 192 191 195 200 202 200 143 217 253 249 242 238 234
69 70 78 113 97 74 43 106 127 140 152 155 125 97 112 150 185 194 174 183 196 198 202 208 209 185 247 254 255 254 254
72 44 63 59 46 52 49 74 127 137 149 132 103 78 90 134 141 168 166 165 166 207 207 204 203 216 193 238 244 251 242 236 243
58 29 69 73 99 80 46 74 117 127 144 161 164 124 125 120 156 187 183 186 189 206 201 205 214 194 174 185 187 183 193
65 49 77 89 30 68 43 61 109 127 141 147 113 102 121 145 148 189 181 178 181 201 201 205 202 174 169 178 183 186 184
82 76 92 79 54 58 37 47 90 121 132 116 89 79 111 146 163 149 122 126 180 197 197 198 178 149 146 152 155 157 159 168
104 107 122 123 105 79 27 33 66 111 122 120 114 114 147 175 190 194 163 163 101 170 200 187 185 156 146 145 149 137 141 140 145
117 124 127 133 133 125 21 20 37 88 115 121 128 128 141 142 168 202 212 153 164 186 180 168 154 146 144 149 151 151 147 144
119 118 118 125 128 111 21 29 28 58 100 118 131 140 151 159 188 201 205 192 160 168 149 186 119 144 147 143 142 141
117 119 125 130 139 156 18 29 44 58 70 102 130 144 168 197 212 215 210 195 177 132 133 195 57 59 151 145 142 142 141
115 123 126 134 143 102 27 54 58 45 69 105 128 175 189 189 216 206 185 139 111 164 203 74 5 124 151 142 142 143 146
101 108 123 121 132 125 44 40 31 35 57 44 58 58 101 175 189 183 216 206 185 139 111 164 203 74 48 160 162 142 144 145
98 97 97 96 104 76 34 33 39 48 41 49 51 59 74 53 55 66 63 89 150 188 209 156 62 102 140 149 125 133 131 131
102 102 97 88 73 20 30 23 42 50 65 41 90 60 59 31 57 82 123 157 187 205 169 62 96 151 105 101 154 135 130 129



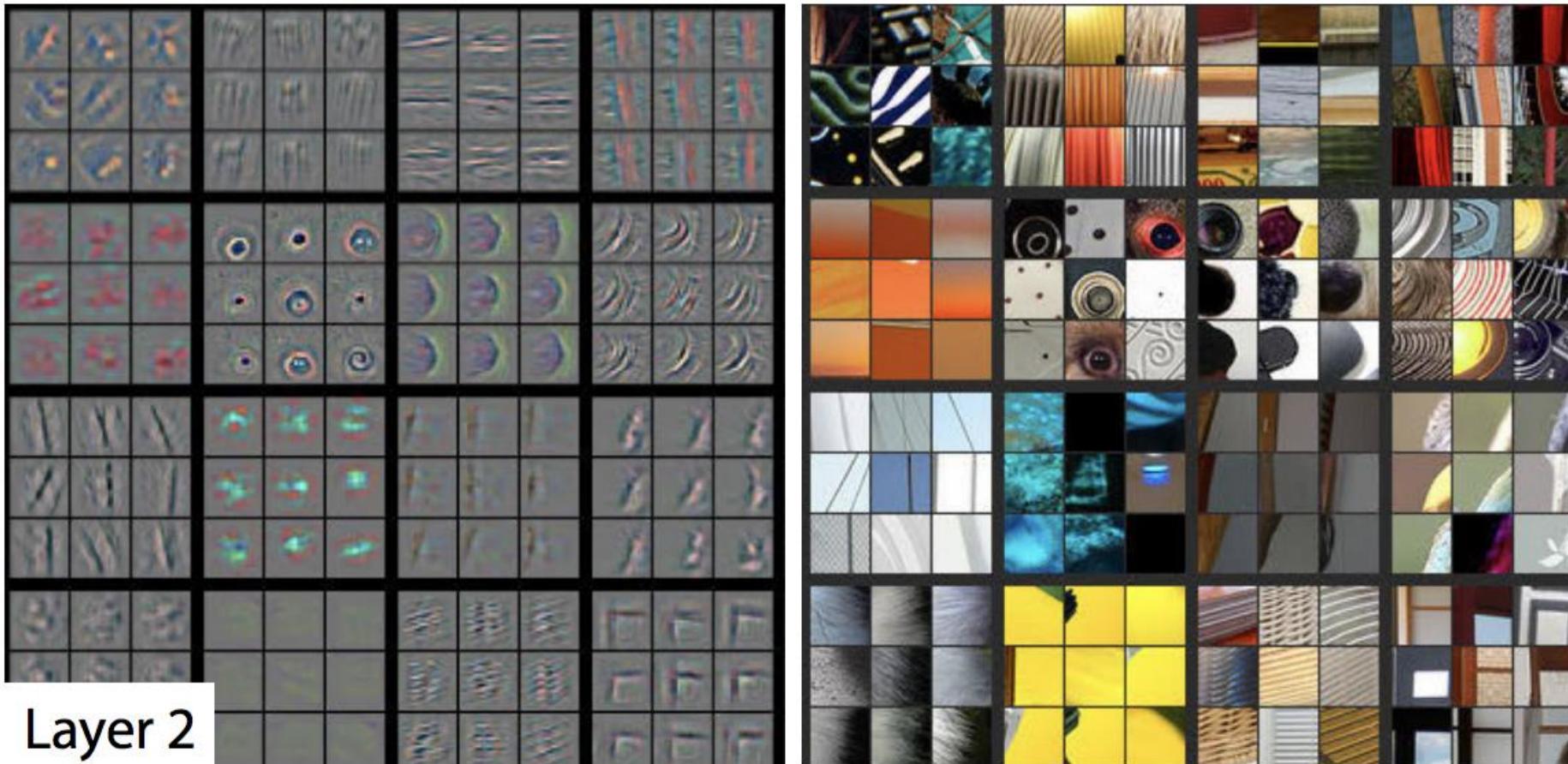
Source: <http://setosa.io/ev/image-kernels/>

Visualizing and Understanding Convolutional Networks by Matthew D. Zeiler and Rob Fergus (2013)



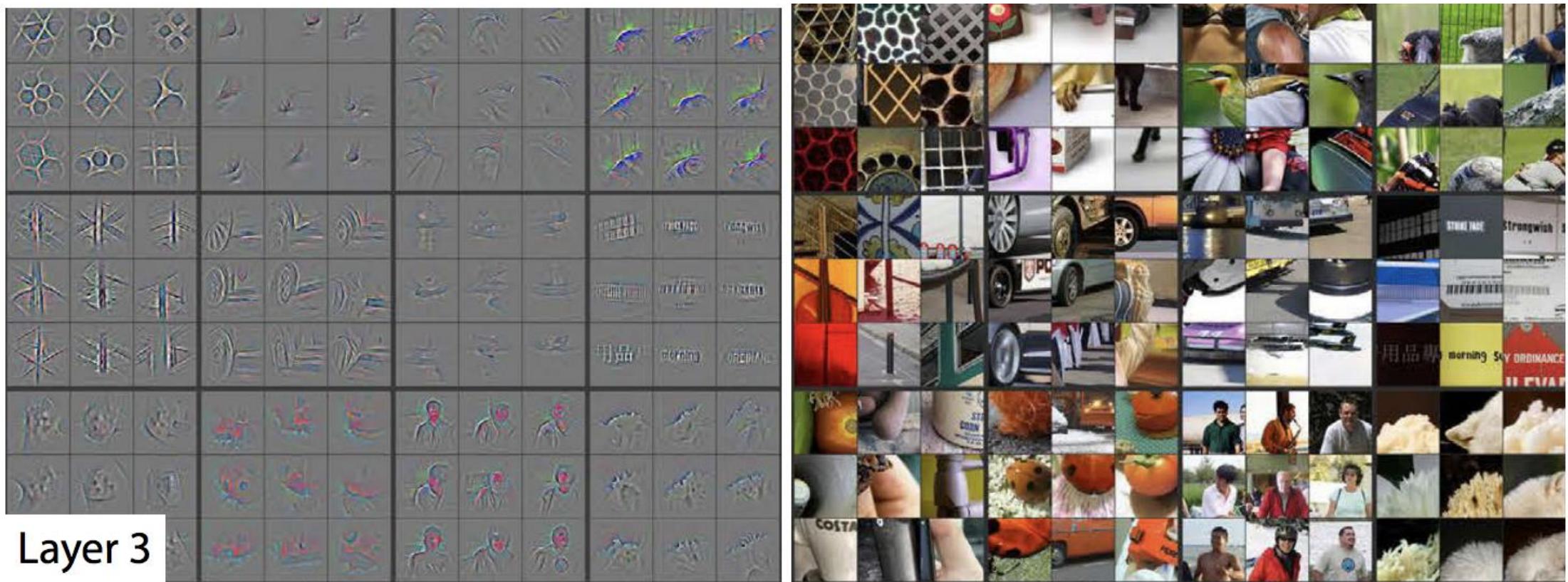
Source: <https://arxiv.org/abs/1311.2901>

Visualizing and Understanding Convolutional Networks by Matthew D. Zeiler and Rob Fergus (2013)



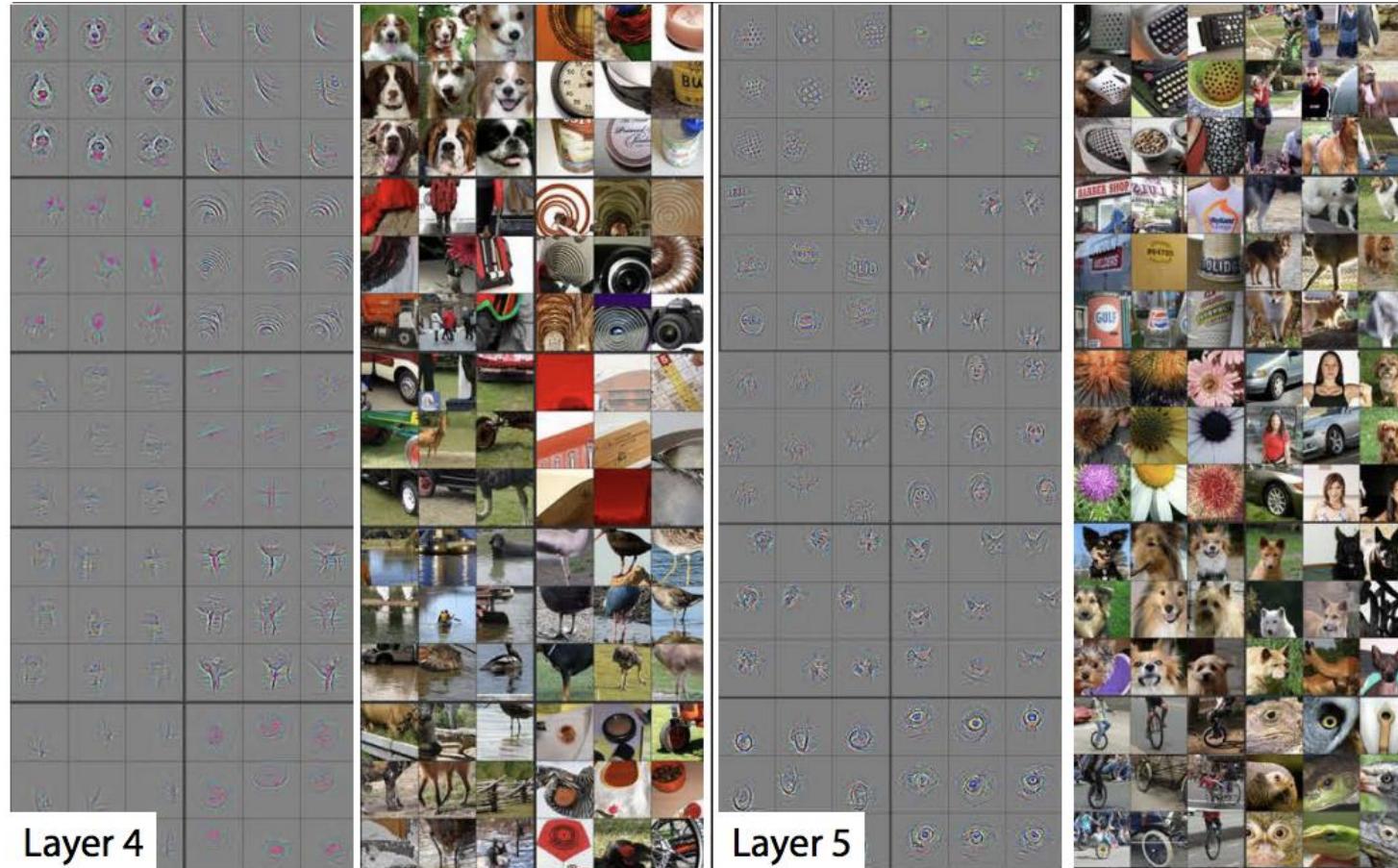
Source: <https://arxiv.org/abs/1311.2901>

Visualizing and Understanding Convolutional Networks by Matthew D. Zeiler and Rob Fergus (2013)



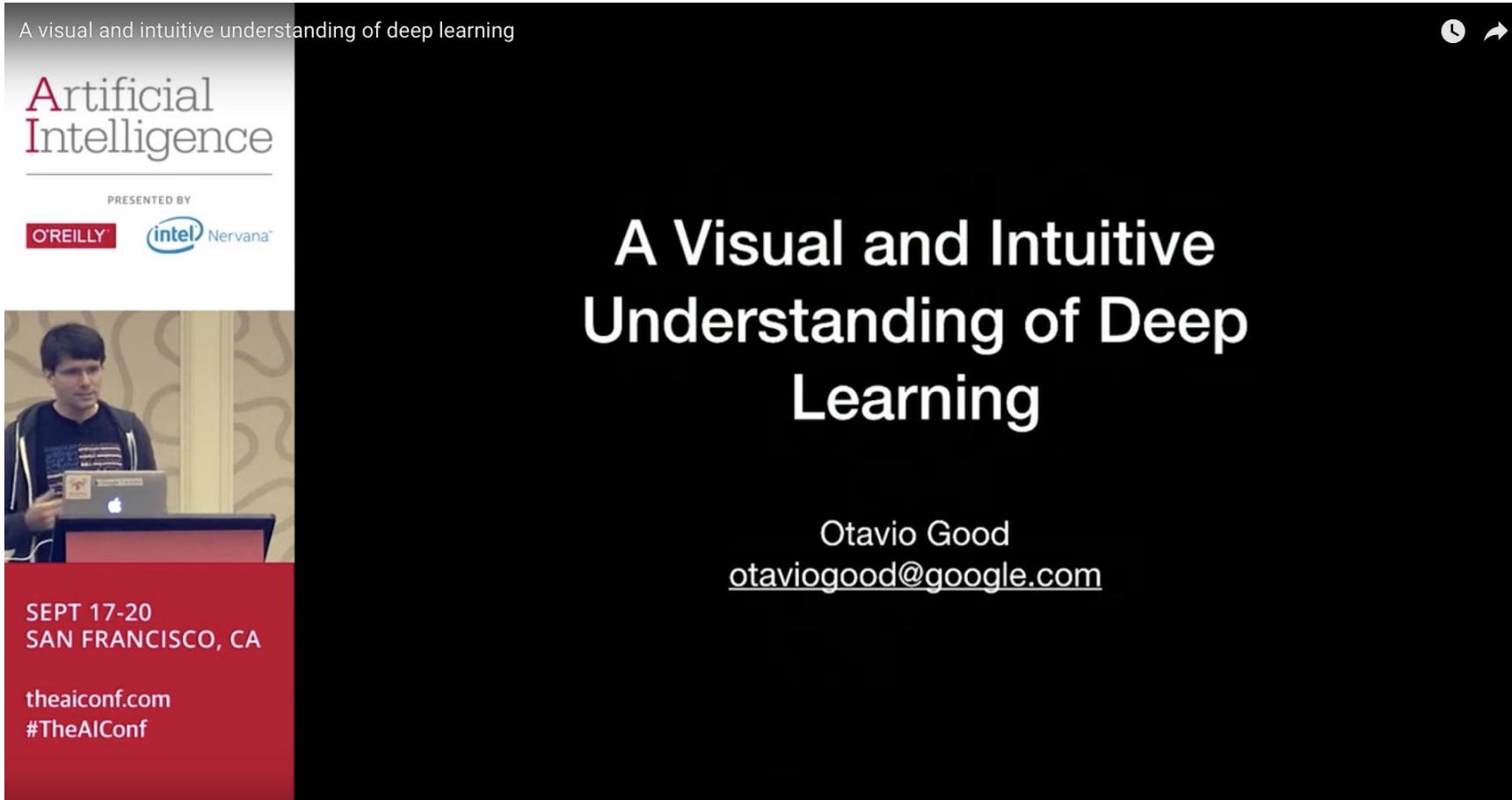
Source: <https://arxiv.org/abs/1311.2901>

Visualizing and Understanding Convolutional Networks by Matthew D. Zeiler and Rob Fergus (2013)



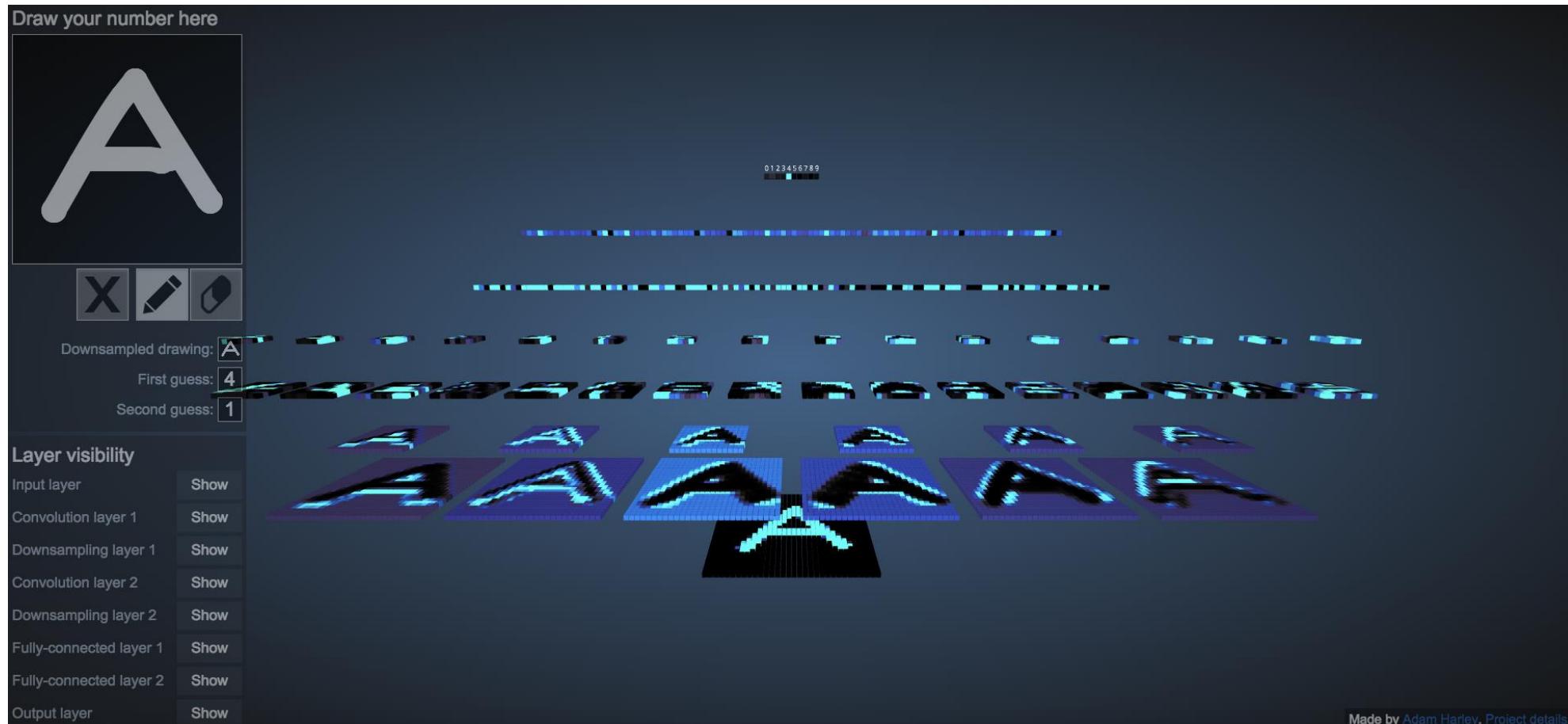
Source: <https://arxiv.org/abs/1311.2901>

Otavio Good



Source: https://www.youtube.com/watch?v=Oqm9vsf_hvU

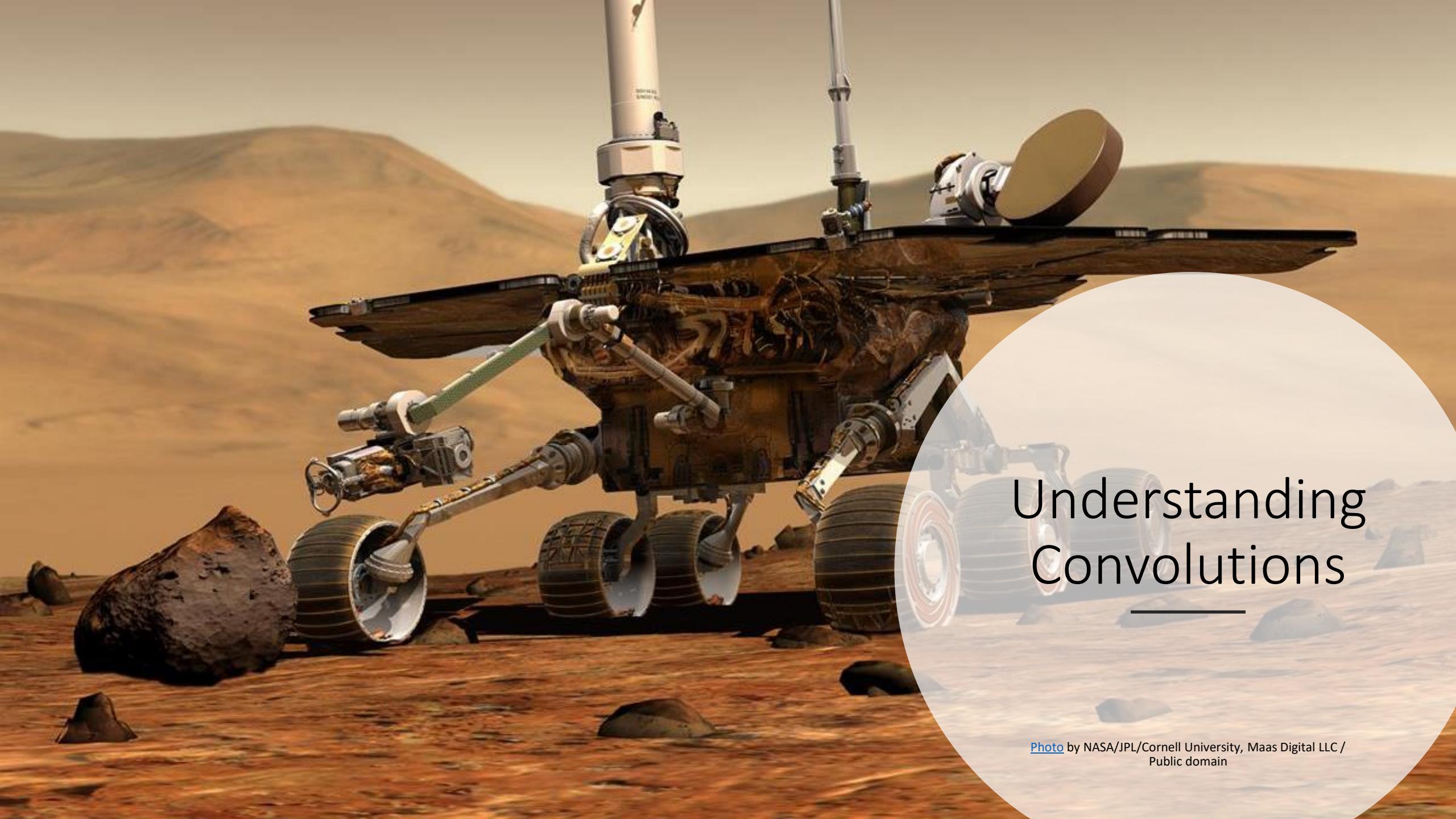
Adam Harley



Source: <http://scs.ryerson.ca/~aharley/vis/conv/>

More NN Visualisation

- [2D Convolutional Network Visualisation](#). Source: An Interactive Node-Link Visualization of Convolutional Neural Networks by [Adam W. Harley](#).
- [Tensor Playground](#)
- [CS231n Layer 2](#)



Understanding Convolutions

[Photo](#) by NASA/JPL/Cornell University, Maas Digital LLC /
Public domain

Convolutional Networks (CNN) Properties

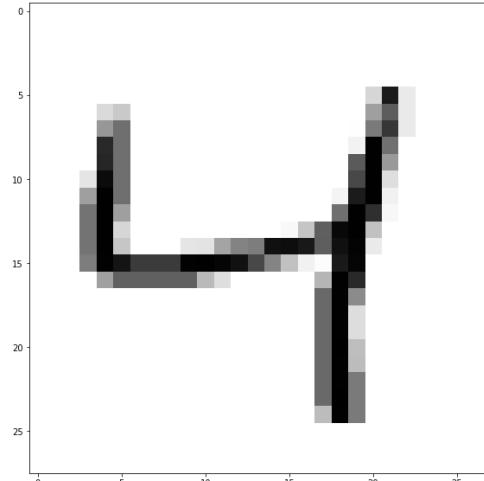
- Local Invariance
 - Once the CNN have learnt the pattern, it can recognise it anywhere.
 - Also called translation invariant
- Compositionality / Learn spatial hierarchies of patterns
 - For example,
 - The first layer may learn simple patterns such as vertical and horizontal edges
 - The second layer may learn more complex patterns that are made up of features from the first layer. This ensures CNN learn efficiently

Convolutional Networks (CNN) Properties

- Operates with 3D tensors
- Height, Width and channels
 - Black and white picture, channels = 1
 - RGB images: channels = 3
- MNIST
 - Input features: 28, 28, 1
 - Filter size: 3 x 3
 - Number of filters 32
 - Output features: 26, 26, 32

Working with Images

- MNIST images
 - Small
 - 28×28 pixels in 1 channel. Grayscale
 - $28 \times 28 = 784$ values
- Colour Images
 - Large
 - $640 \times 480 \times 3$ channels = 921,600 values
- Single fully connected later
 - $(640 \times 480 \times 3)^2 = 849,346,560,000$



Layer Types

- Convolutional (CONV)
- Activation (ACCT or RELU)
- Pooling (POOL)
- Fully connected (FC)
- Batch Normalization (BN)
- Dropout (DO)

Example CNN Architecture

Architecture:

INPUT

CONV2D

MAX POOL

CONV2D

MAX POOL

CONV2D

FC

FC

OUTPUT

Example CNN Architecture

Architecture:

INPUT

CONV2D

MAX POOL

CONV2D

MAX POOL

CONV2D

FC

FC

OUTPUT



Image

- This is an example of an image with
- Height of 10 pixels
- Width of 10 pixels
- Depth of 1 pixel
- This is an example of a black a white picture

10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16

Image

- This is an example of an image with
- Height of 10 pixels
- Width of 10 pixels
- Depth (channel) of 3 pixel
- This would be an example of a colour picture with RGB channel

10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16

Example CNN Architecture

Architecture:

INPUT

CONV2D

MAX POOL

CONV2D

MAX POOL

CONV2D

FC

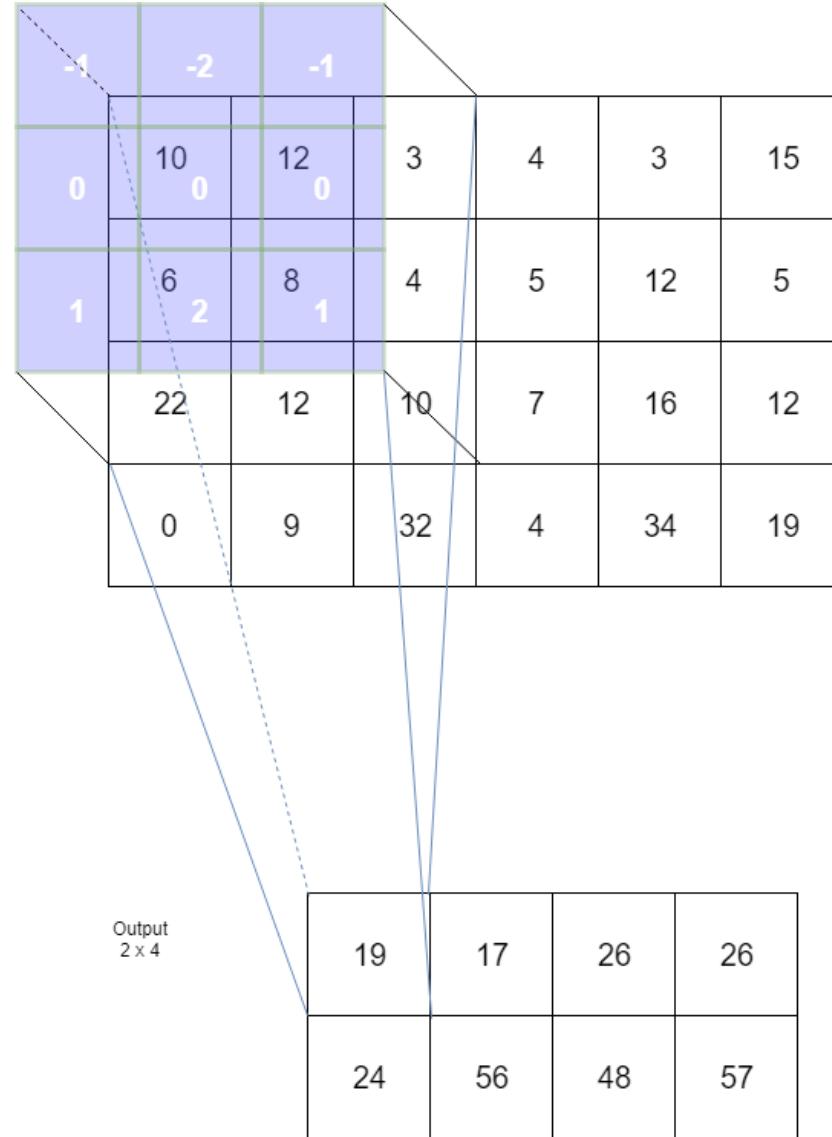
FC

OUTPUT

Convolution Operation

- Sum of Element-wise multiplication
- $(-1 \times 10) + (-2 \times 12) + (-1 \times 3) + (0 \times 6) + (0 \times 8) + (0 \times 4) + (1 \times 22) + (2 \times 12) + (1 \times 10) = 19$
- convolution operation.xlsx
- “stride=1” worksheet

Applying 3 x 3 Filter to the Image



Convolution Example

- This is also an example of applying the kernel filter without padding
- Hence the output size is reduced from 4×6 to 2×4 .

Applying 3×3
Filter to the
Image

-1	-2	-1					
0	10 0	12 0	3	4	3	15	
1	6 2	8 1	4	5	12	5	
22	12	10	7	16	12		
0	9	32	4	34	19		

Output
 2×4

19	17	26	26
24	56	48	57

Convolution Example

- The output is the outcome of one filter
- What happen when you apply multiple filters?
- You will then obtain multiple outputs

Applying 3 x 3 Filter to the Image

-1	-2	-1					
0	10 0	12 0	3	4	3	15	
1	6 2	8 1	4	5	12	5	
22	12	10	7	16	12		
0	9	32	4	34	19		

Output
2 x 4

19	17	26	26
24	56	48	57

Kernel / Filter

```
1 from keras import models
2 from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
3 cnn = models.Sequential()
4 cnn.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
5 cnn.add(MaxPooling2D(2, 2))
6 cnn.add(Conv2D(64, (3, 3), activation='relu'))
7 cnn.add(MaxPooling2D(2, 2))
8 cnn.add(Conv2D(64, (3, 3), activation='relu'))
9 cnn.add(Flatten())
10 cnn.add(Dense(64, activation='relu'))
11 cnn.add(Dense(10, activation='softmax'))
12 cnn.compile(optimizer='rmsprop',
13               loss='categorical_crossentropy',
14               metrics=['accuracy'])
15 cnn.fit(x_train_dat, y_train_dat, epochs=10,
16           batch_size=64,
17           validation_data=(x_val, y_val));
18 print(cnn.summary())
19 test_loss, test_acc = cnn.evaluate(x_test, y_test)
20 print(test_acc)
```

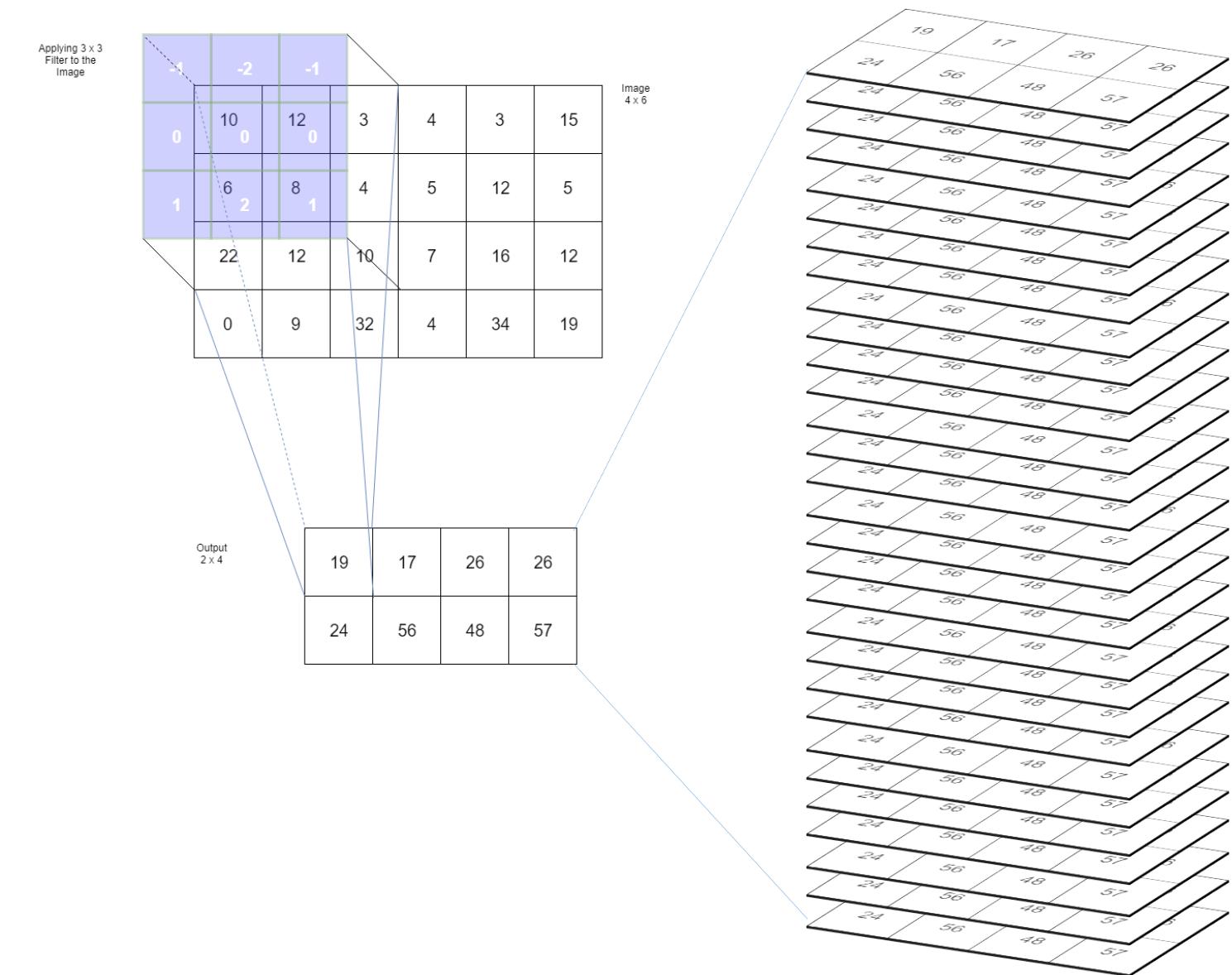
Kernel / Filter

- From 28, 28, 1 to 26, 26, 32
 - The 3x3 filter / kernel and 32 filters resulted in 32 activation map and reduction in dimension from 28, 28 to 26, 26.
 - Change the padding='same', will result in 28, 28

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_3 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_5 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_4 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_6 (Conv2D)	(None, 3, 3, 64)	36928
flatten_2 (Flatten)	(None, 576)	0
dense_5 (Dense)	(None, 64)	36928
dense_6 (Dense)	(None, 10)	650
<hr/>		
Total params: 93,322		
Trainable params: 93,322		
Non-trainable params: 0		

Convolution Example

- In the code example provided above, 32 filters (kernels) were used.
- There will be 32 outputs.
- Hence we have 32 copies of 26×26 image. Also called activation map.
- *Picture provided is just to illustrate the effects of filtering. It is not related to the example*



Common Filters (Kernels)

Examples of Some Common Filters

0	-1	0
-1	5	-1
0	-1	0

Sharpen

0.0625	0.1250	0.0625
0.1250	0.2500	0.1250
0.0625	0.1250	0.0625

Blur

-1	-2	-1
0	0	0
1	2	1

Bottom Sobel

Reference: <http://setosa.io/ev/image-kernels/>

Filters

- A few more things to note that affect the output dimension:
 - The depth of the output relates to the **number of filters** (filters in Keras Conv2D) applied
 - **Strides** (strides=(1, 1) in Keras) specifying the strides of the convolution along the height and width. The default is 1 for both height and width. Larger than 1 means skipping pixel and has the effect of reducing the output
 - **Padding** or more accurately zero-padding, add zero to the external dimension (for top of image it would be row-zero-padding, for sides it would be column-zero-padding) to recover the original dimension. In the example earlier, changing the setting in Keras from padding="valid" to padding="same" recover the original dimension of 28x28.

Strides=(height=1,width=2).
Skipping Column 2.

- You can use strides to reduce the dimensionality
 - convolution operation.xlsx
 - “stride = 2” worksheet

Applying 3 x 3 Filter to the Image

-1 10	-2 12	-1 3	4	3	15
0 6	0 8	0 4	5	12	5
1 22	2 12	1 10	7	16	12
0	9	32	4	34	19

Output
 2×4

19	26
24	48

Applying 3 Filter to th Image

10	12	-1 3	-2 4	-1 3	15
6	8	0 4	0 5	0 12	5
22	12	1 10	2 7	1 16	12
0	9	32	4	34	19

Output
2 x 2

19	26
24	48

10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16

Applying 3x3 filter to 10x10 matrix. Strides=(1,1)

convolution operation.xlsx

“stride=1” worksheet

10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16

Applying 3x3 filter to 10x10 matrix. Strides=(1,1)

10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16

Applying 3x3 filter to 10x10 matrix. Strides=(1,1)

10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16

Applying 3x3 filter to 10x10 matrix. Strides=(1,1)

10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16

Applying 3x3 filter to 10x10 matrix. Strides=(1,1)

10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16

Applying 3x3 filter to 10x10 matrix. Strides=(1,1)

10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16

Applying 3x3 filter to 10x10 matrix. Strides=(1,1)

10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16

Applying 3x3 filter to 10x10 matrix. Strides=(1,1)

Output = 8 x 8

10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16

Applying 3x3 filter to 10x10 matrix. Strides=(2,2)

convolution operation.xlsx

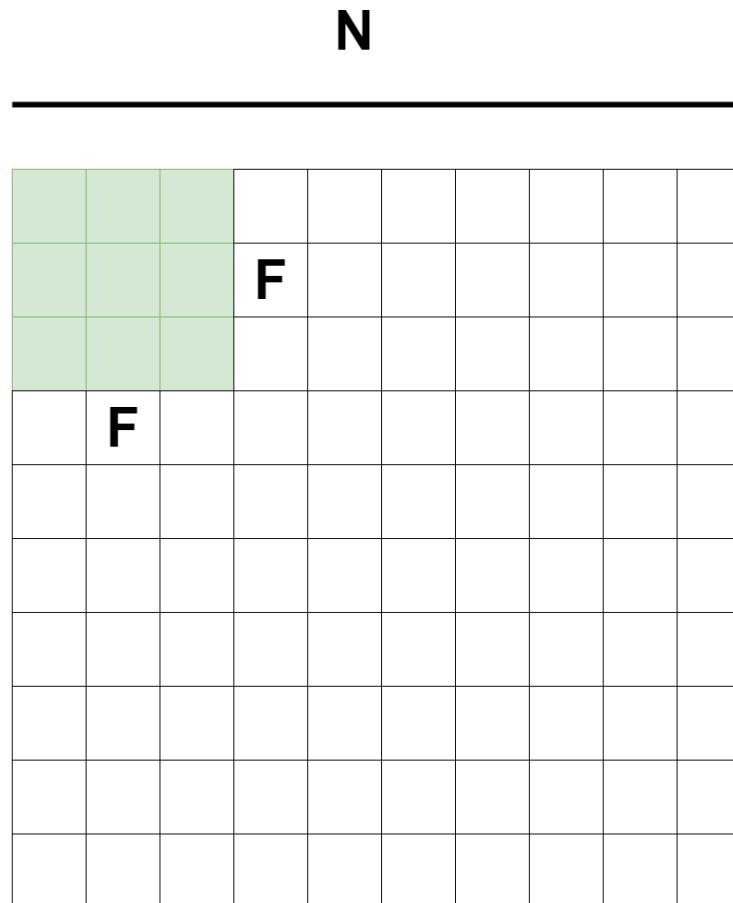
“stride=2” worksheet

10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
10	12	3	4	3	10	12	3	4	3
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16
0	9	32	4	34	0	9	32	4	34
6	8	4	5	12	6	8	4	5	12
22	12	10	7	16	22	12	10	7	16

Applying 3x3 filter to 10x10 matrix. Strides=(2,2)

Output = 4 x 4

Output Size depends on a Few Factors



Output Size:

$$(N - F) / \text{Stride} + 1$$

N $N=10, F = 3$

$$\text{Stride} = 1, (10-3)/1 + 1 = 8$$

$$\text{Stride} = 2, (10-3)/2 + 1 = 4.5$$

Zero Padding

- With zero padding, you are able to recover the original dimension after applying the filter
- convolution operation.xlsx
- “zero padding” worksheet

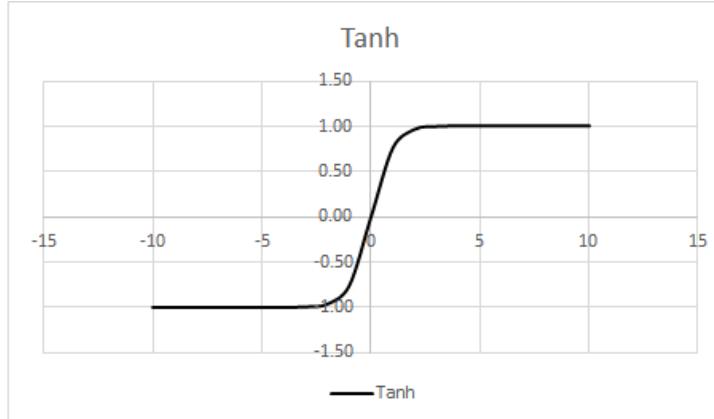
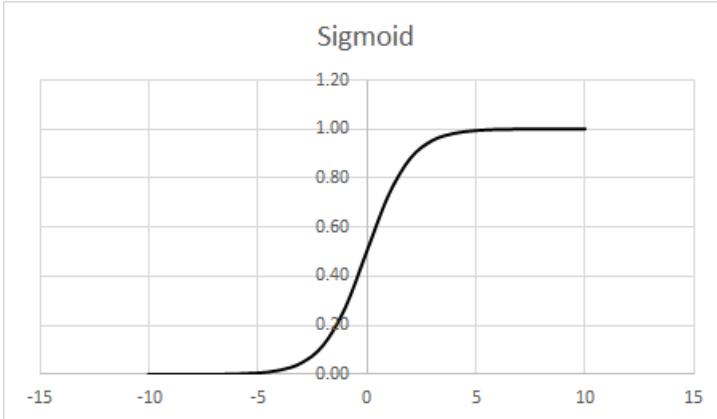
Applying 3 x 3 Filter to the Image

-1	0	-2	0	-1	0	0	0	0	0	0
0	0	0	10	0	12	3	4	3	15	0
1	0	2	6	1	8	4	5	12	5	0
0		22		12		10	7	16	12	0
0	0	0	9	32	4	34	19	0		
0	0	0	0	0	0	0	0	0	0	0

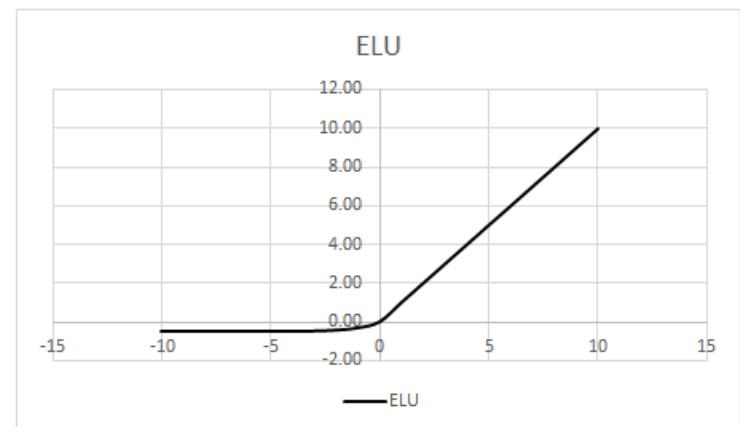
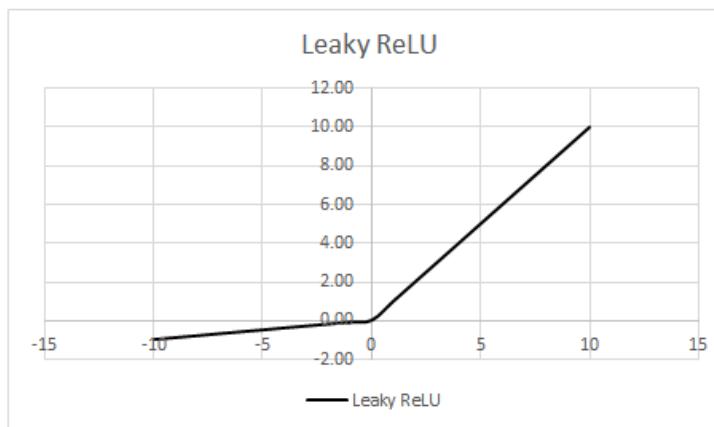
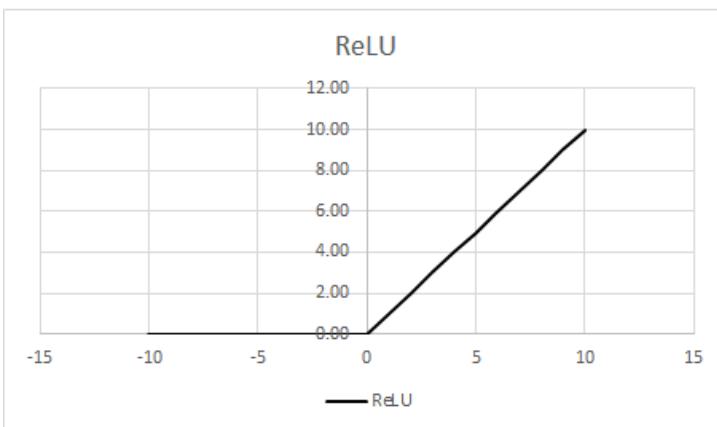
Output
4 x 6

20	26	21	26	34	22
24	19	17	26	26	7
-11	24	56	48	57	50
-56	-56	-39	-40	-51	-40

Common Activation Functions



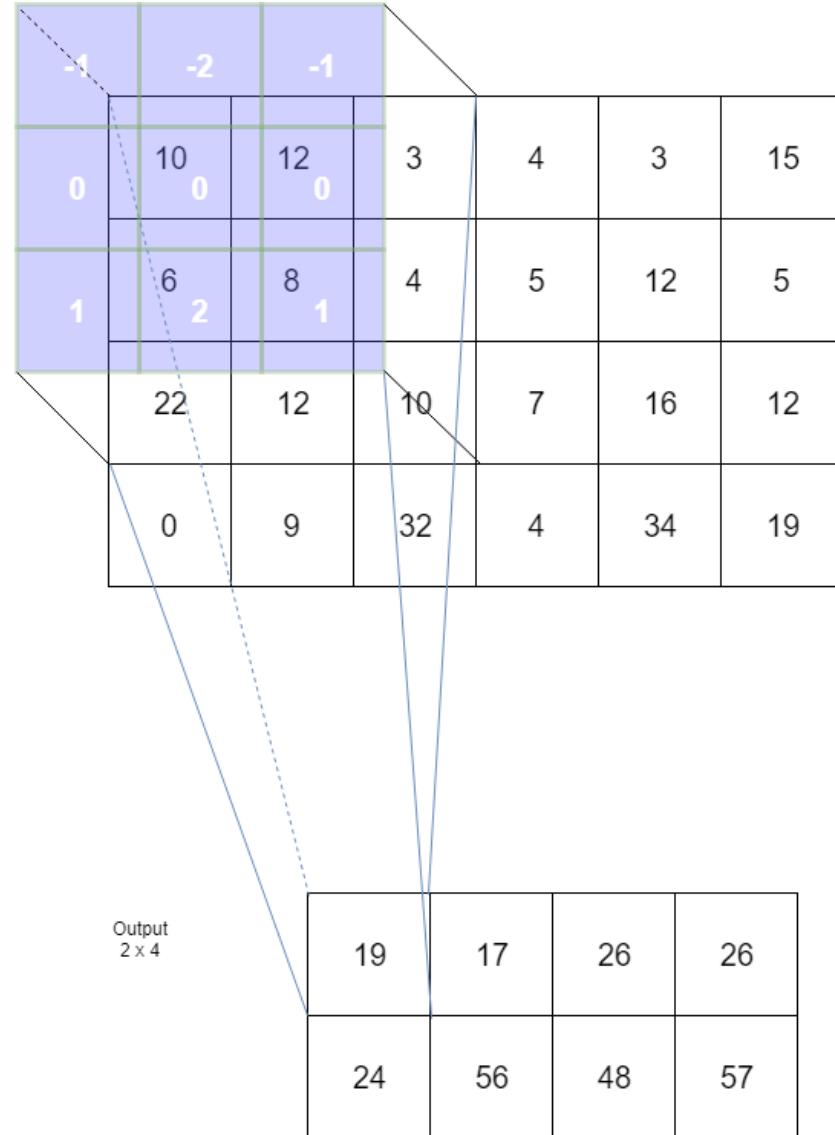
activation fn.xlsx



Activation

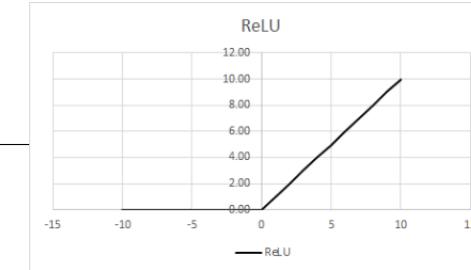
- If we pass the output layer, shown on the right, through a ReLU activation function, you will obtain the same result.
- That is because ReLU zero any negative numbers and leave positive numbers alone.
- Usually follow straight after convolutional layer. Often not specified in architecture

Applying 3 x 3 Filter to the Image



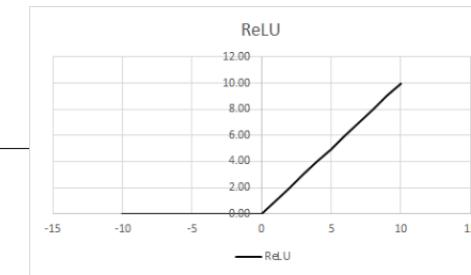
Output + Activation Functions

19	17	26	26
24	56	48	57



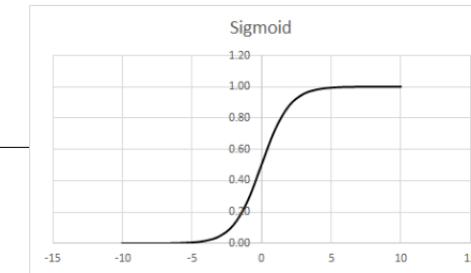
19	17	26	26
24	56	48	57

0.5	0.3	-1.2	0.6
-2.2	-0.2	1.4	-0.4



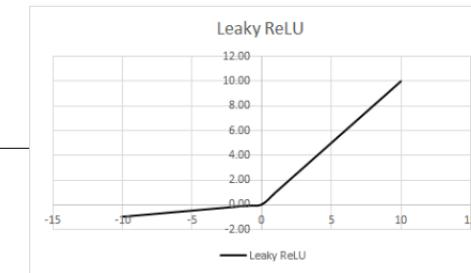
0.5	0.3	0.0	0.6
0.0	0.0	1.4	0.0

0.5	0.3	-1.2	0.6
-2.2	-0.2	1.4	-0.4



0.6	0.6	0.2	0.6
0.1	0.5	0.8	0.4

0.5	0.3	-1.2	0.6
-2.2	-0.2	1.4	-0.4



0.5	0.3	-0.1	0.6
-0.2	-0.0	1.4	-0.0

Example CNN Architecture

Architecture:

INPUT

CONV2D

MAX POOL

CONV2D

MAX POOL

CONV2D

FC

FC

OUTPUT

Pooling Layer

- Average or Max pooling
- Applies to each activation map independently
- Purpose of pooling
 - Reduce representations / spatial size
 - Speed up computation
 - Helps control overfitting

Single Activation Map

10	12	3	4
6	8	4	5
22	12	10	7
0	9	32	4

**Max Pool 2x2 filters
stride = 2**

12	5
22	32

**Max Pool 2x2 filters
stride = 1**

12	12	5
22	12	10
22	32	32

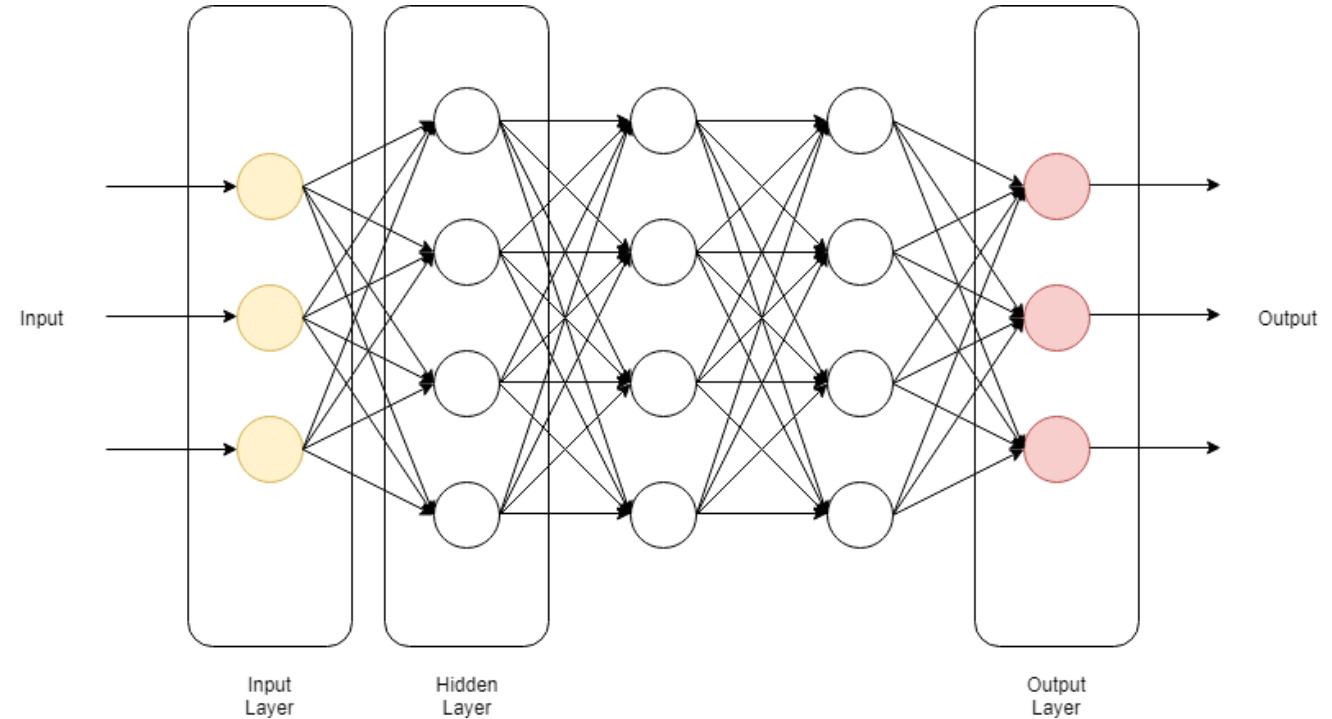
Flatten

- Flatten for the classifier (sigmoid, softmax etc.)
- Classifier requires features to be in vector (1D) form.



Fully Connected (FC) Layer. (Dense Layer)

- Note that neurons are all fully connected to all activations in the previous layer
- Like a normal ANN
- Normally observed at the end



ConvNetJS Demo

ConvNetJS CIFAR-10 demo

Description

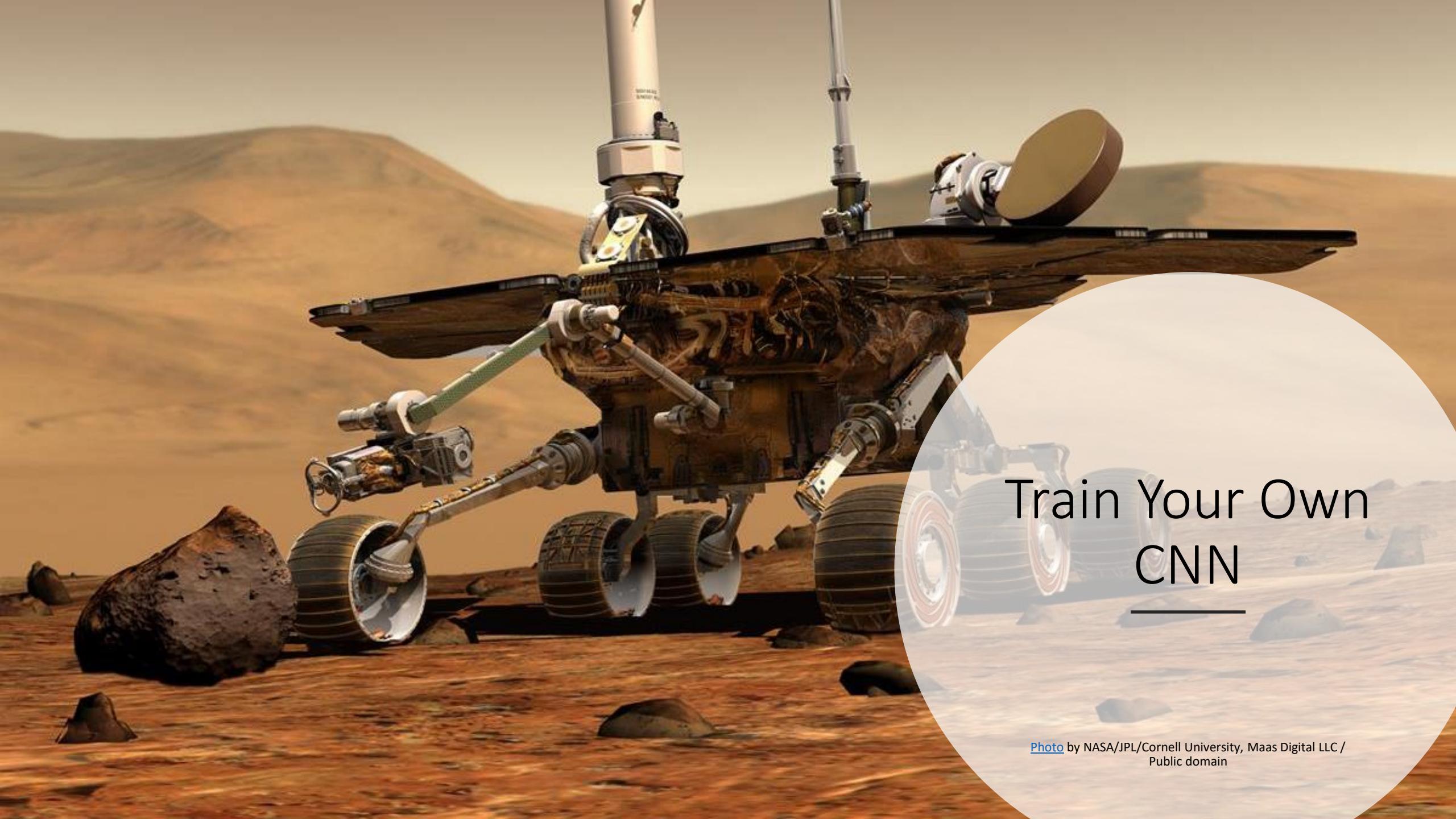
This demo trains a Convolutional Neural Network on the [CIFAR-10 dataset](#) in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used [this python script](#) to parse the [original files](#) (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and vertically.

By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to [@karpathy](#).

Source: <https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>



Train Your Own CNN

[Photo](#) by NASA/JPL/Cornell University, Maas Digital LLC /
Public domain



Getting the Data | [Dogs and Cats Redux](#)



Train Your Own CNN

`cats_and_dogs.ipynb`

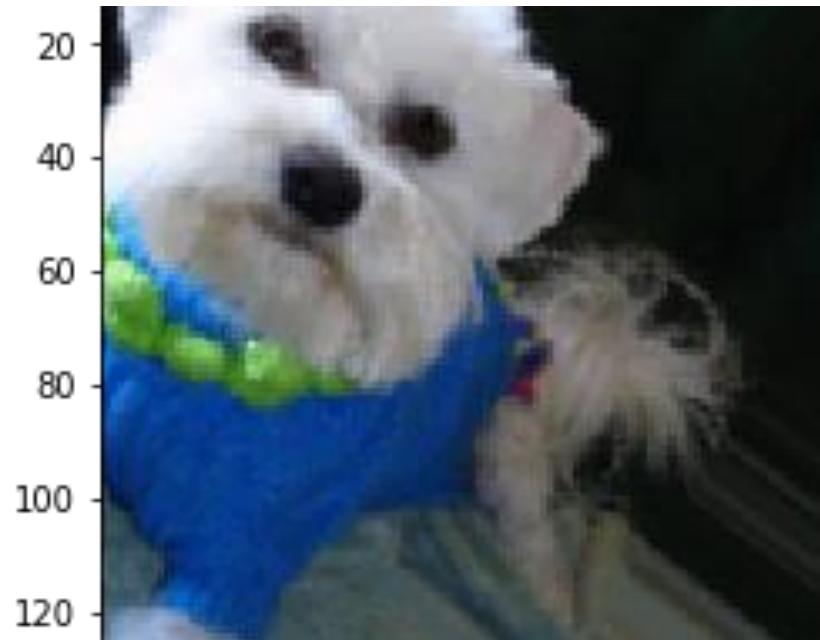
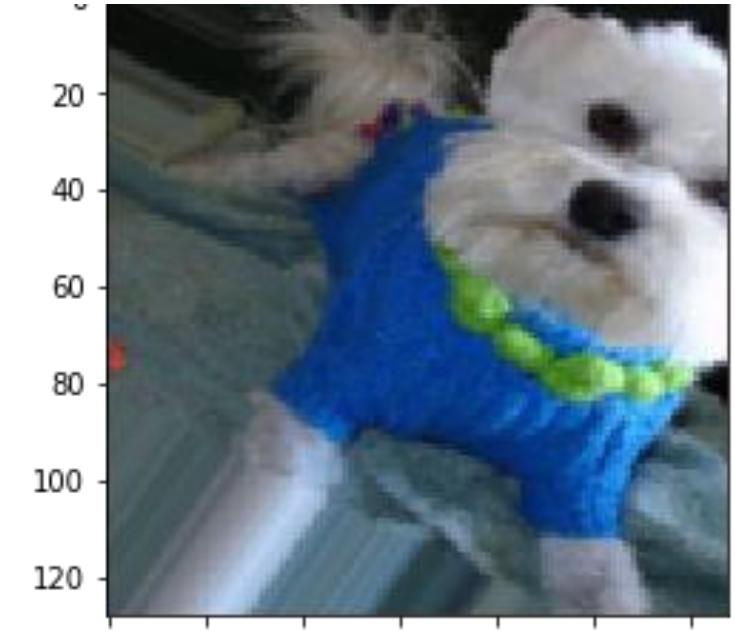
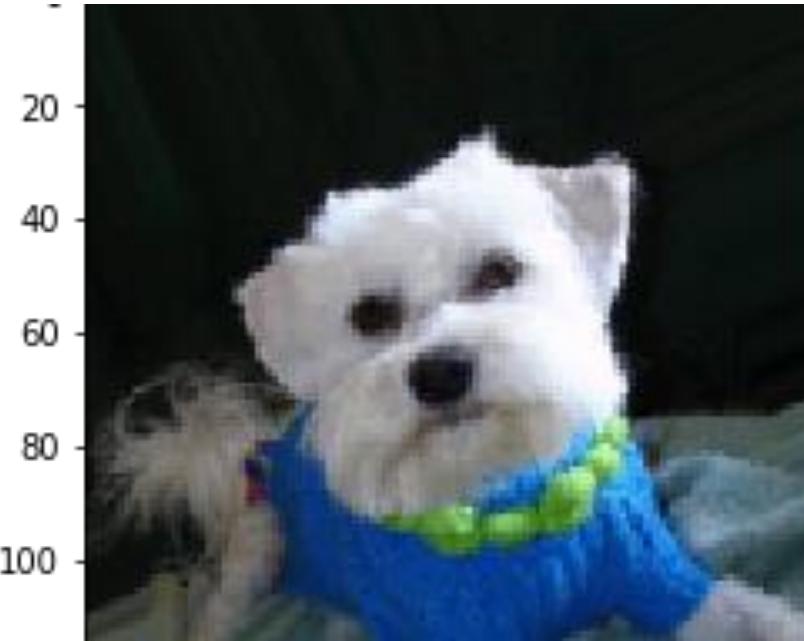
Save and Load Model

- cats_and_dogs.ipynb
- cats_and_dogs_load.ipynb



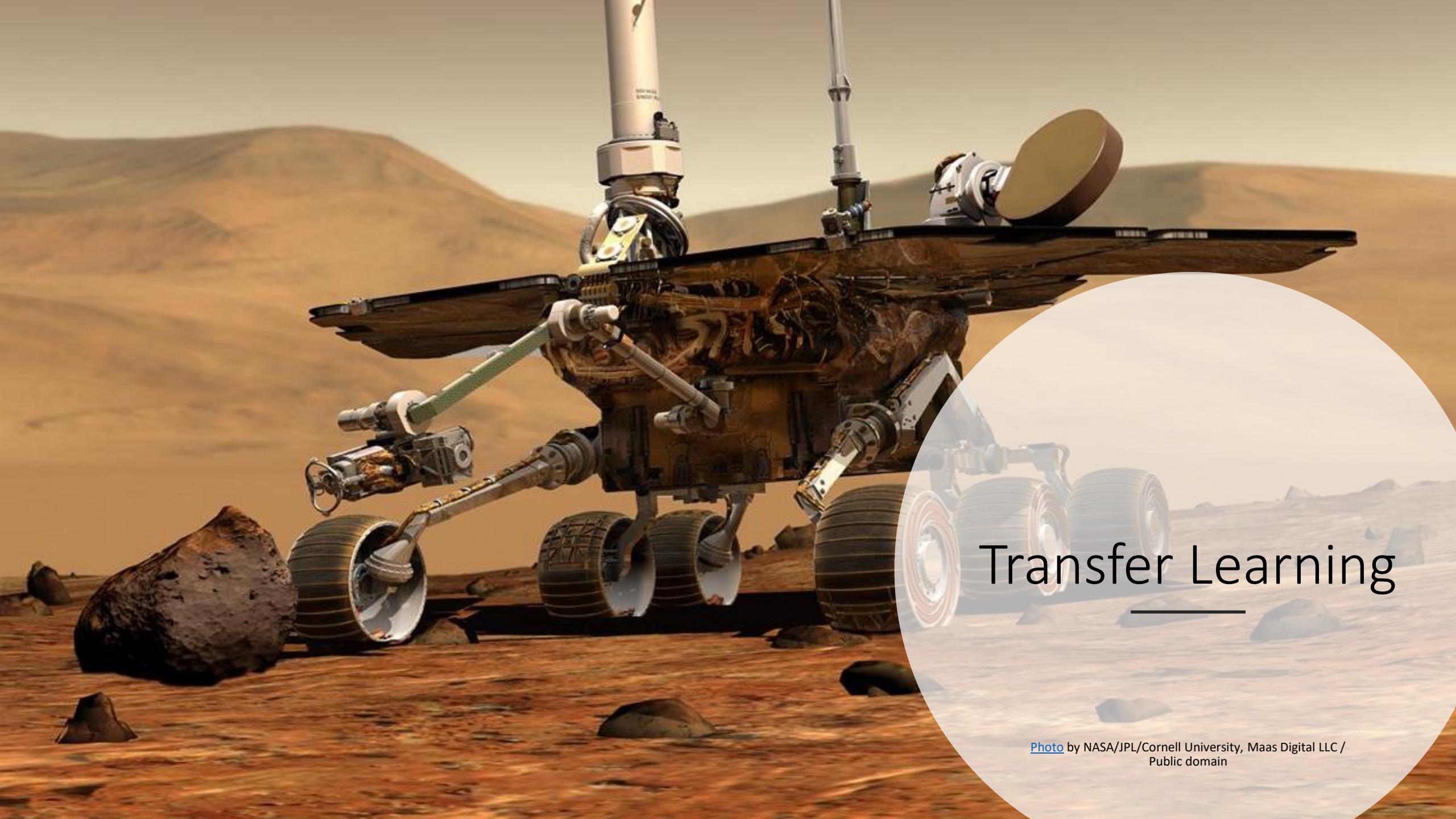
Data Augmentation

- Kera Image Preprocessing [Image Data Generator](#)
- 111_Data_Augmentation.ipynb
- rotation_range
- width_shift_range
- height_shift_range
- shear_range
- zoom_range
- horizontal_flip
- vertical_flip



Data Augmentation

Show randomly generated examples



Transfer Learning

[Photo](#) by NASA/JPL/Cornell University, Maas Digital LLC /
Public domain

Transfer Learning



- cats_and_dogs_transfer_learning.ipynb

The background of the image is a photograph of a forest path. The path is a dirt trail winding through a dense forest. The trees are tall and thin, with dark trunks and green leaves. The ground is covered in green grass and small plants. The lighting suggests it might be early morning or late afternoon, with dappled sunlight filtering through the canopy.

Feature Extraction

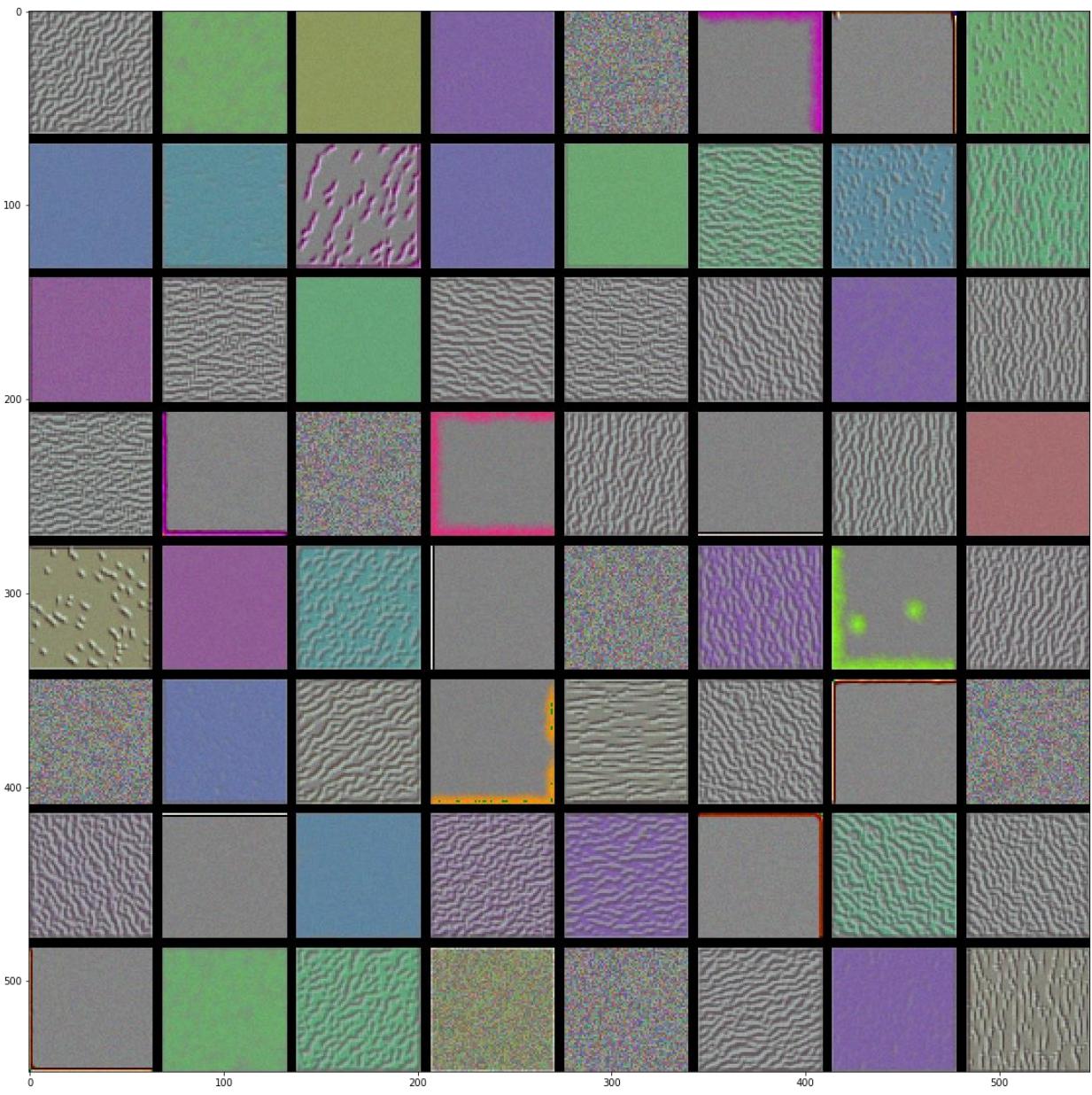
121 Feature Extraction.ipynb



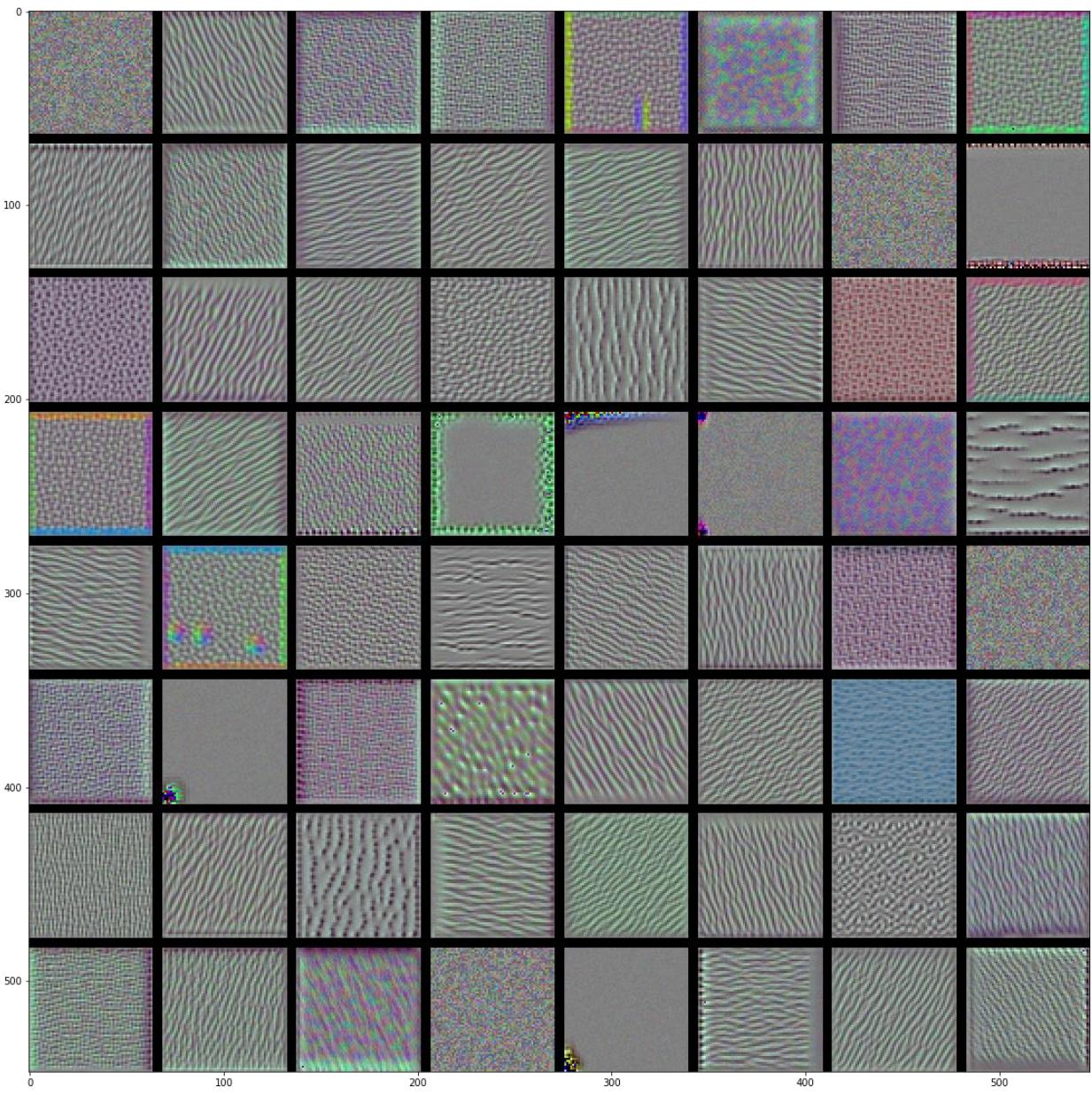
Visualising Network Layer

[Photo](#) by NASA/JPL/Cornell University, Maas Digital LLC /
Public domain

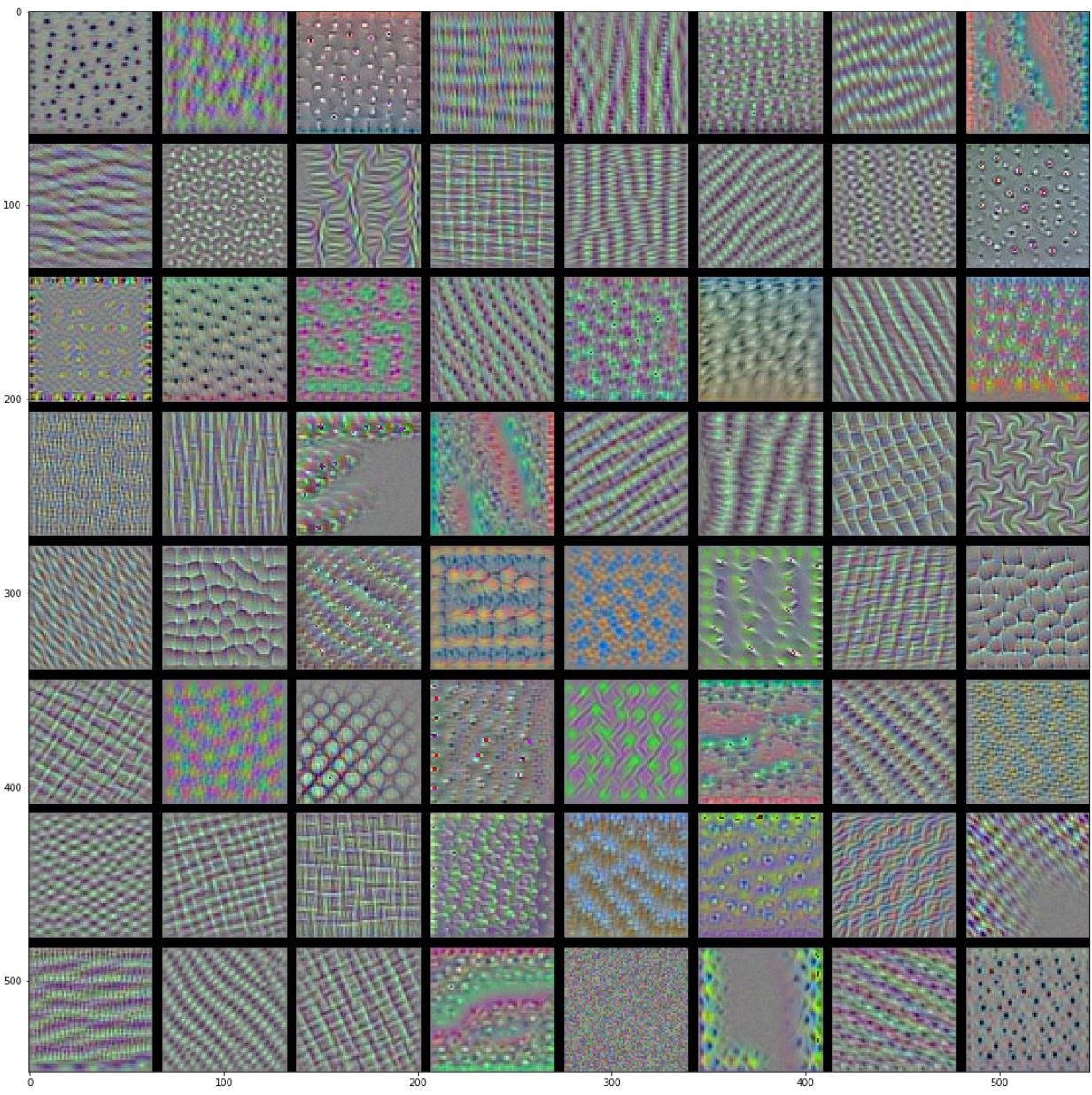
Visualising Network Layer



Visualising Network Layer



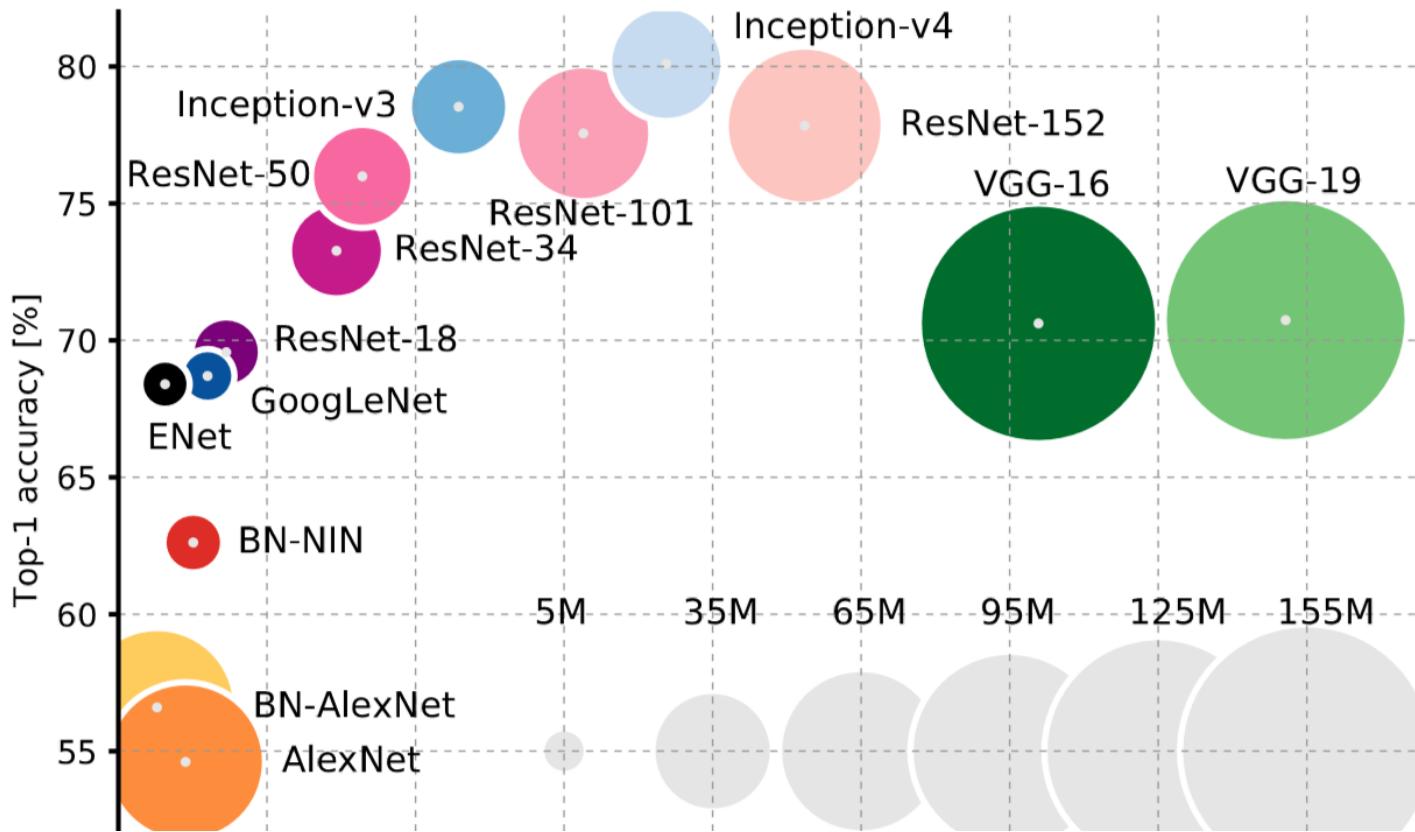
Visualising Network Layer





Where to From
Here?

[Photo](#) by NASA/JPL/Cornell University, Maas Digital LLC /
Public domain



Where to From Here?

- Fine-Tuning Networks
- Adaptive Learning Rate Methods
- Optimisation Method
- Advanced / State of the Art Architecture
 - VGGNet
 - AlexNet
 - GoogLeNet (Inception)
 - ResNet
- Picture source: An Analysis of Deep Neural Network Models for Practical Applications

State of the Art Tools

- [H2O.ai](#)
- [Deep Cognition](#)
- [Rapidminer](#)
- [KNIME](#)
- [ALTERYX](#)
- [DataRobot](#)

- Google Co-Lab
- Kaggle
- Notebooks.azure.com