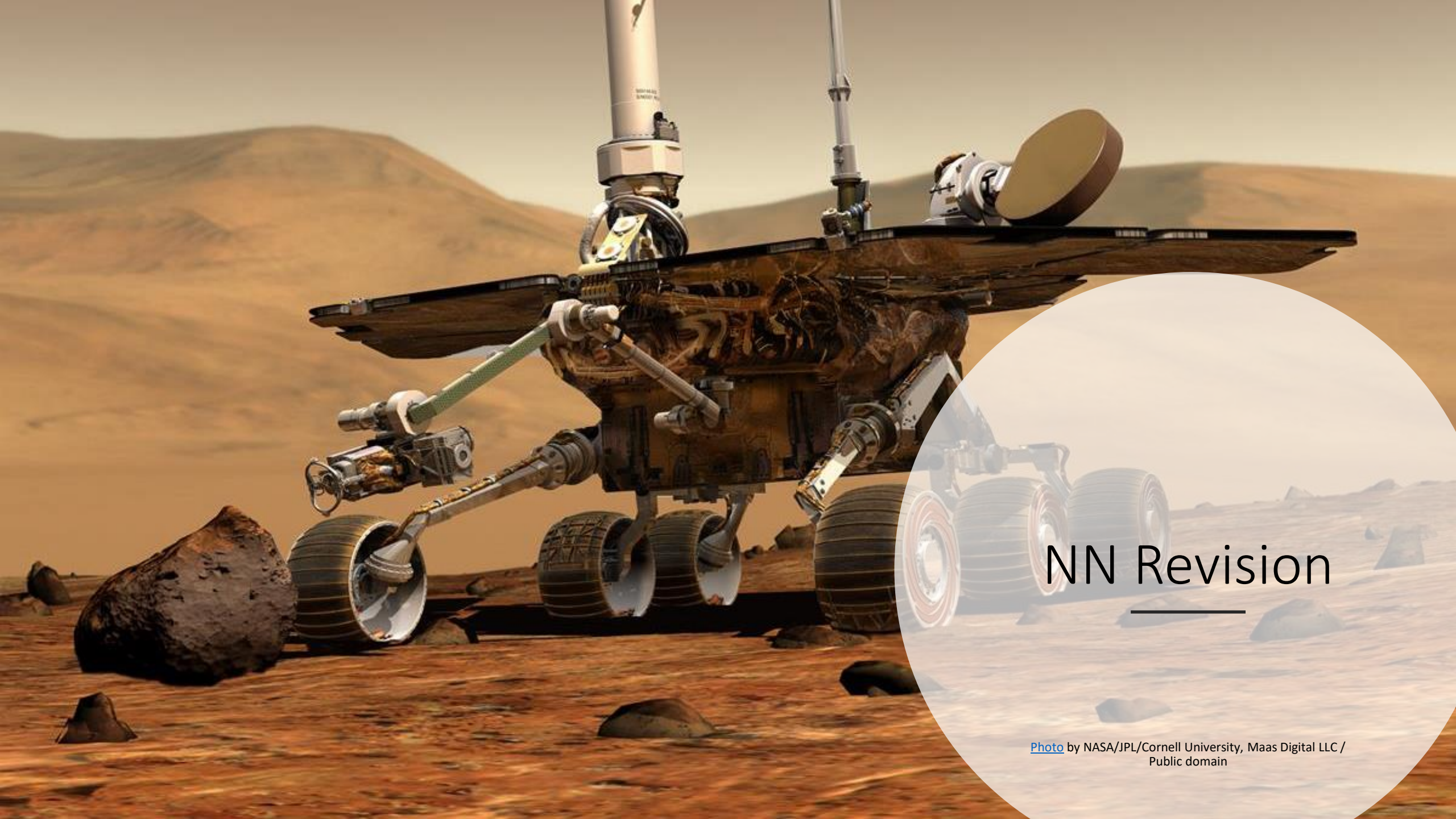
Introduction to Convolutional Neural Networks

Photo by NASA/JPL/Cornell University, Maas Digital LLC / Public domain

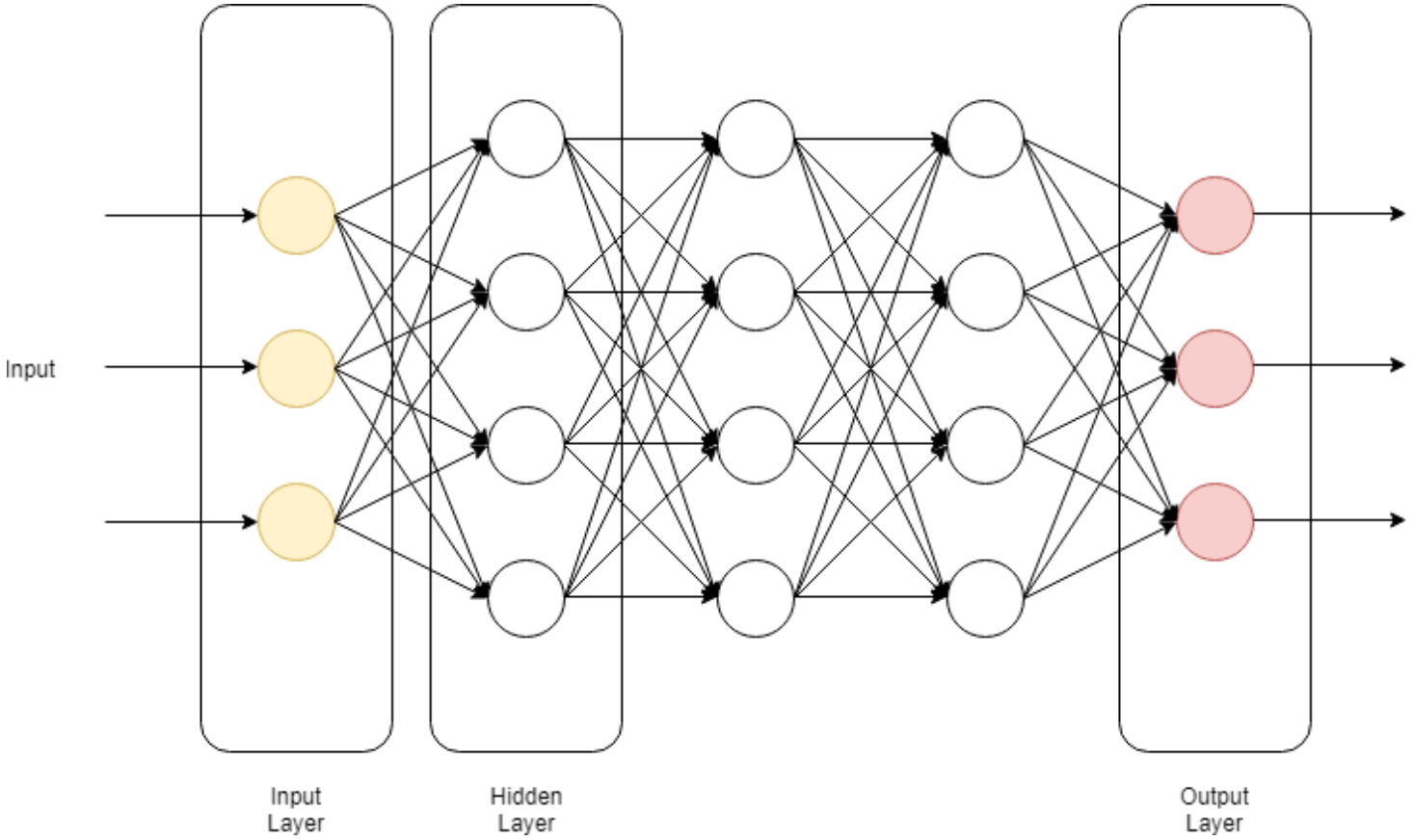
Outline

- 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

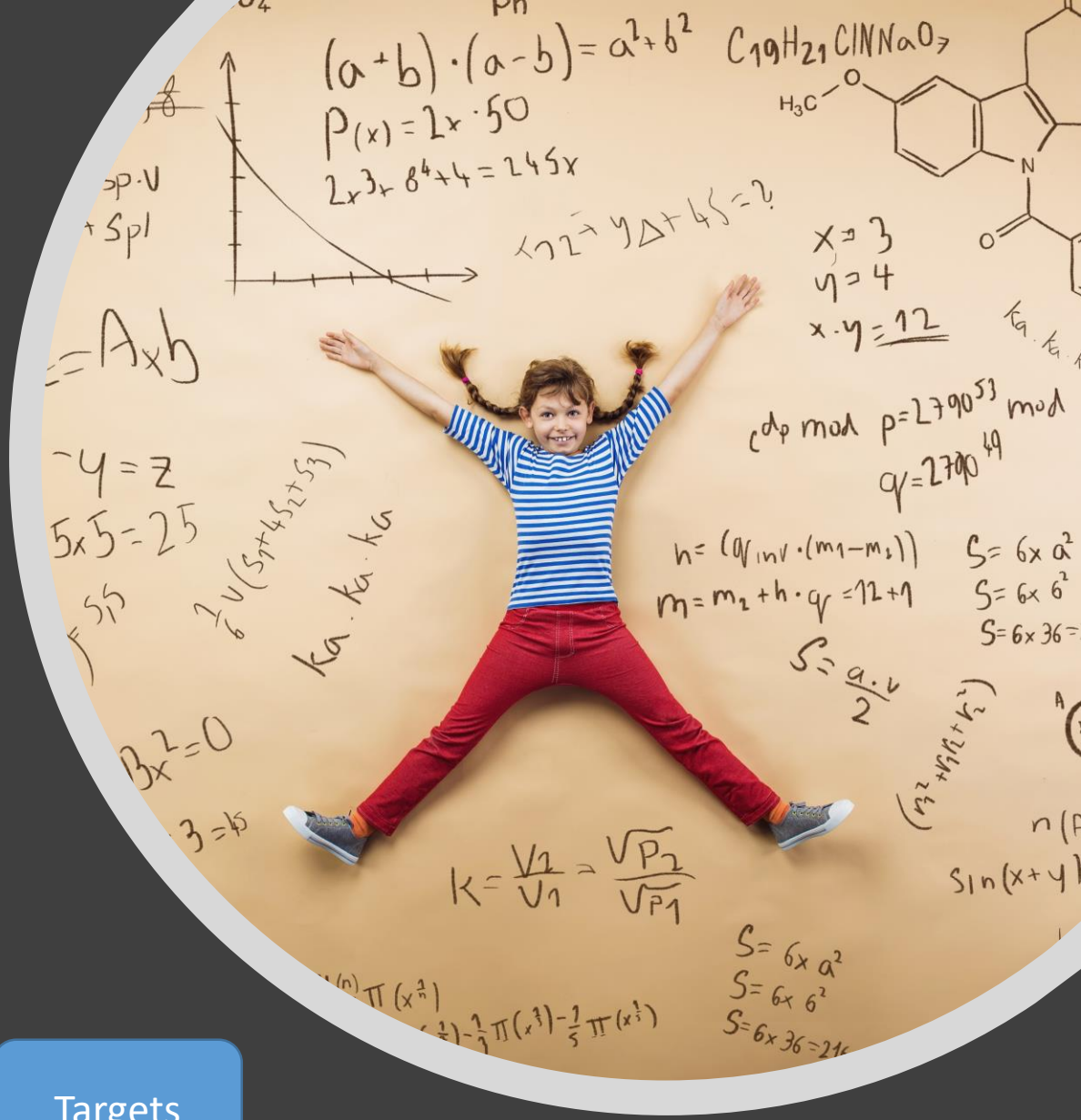
Photo by NASA/JPL/Cornell University, Maas Digital LLC / Public domain



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



Input Data → ? → Targets

VGG16

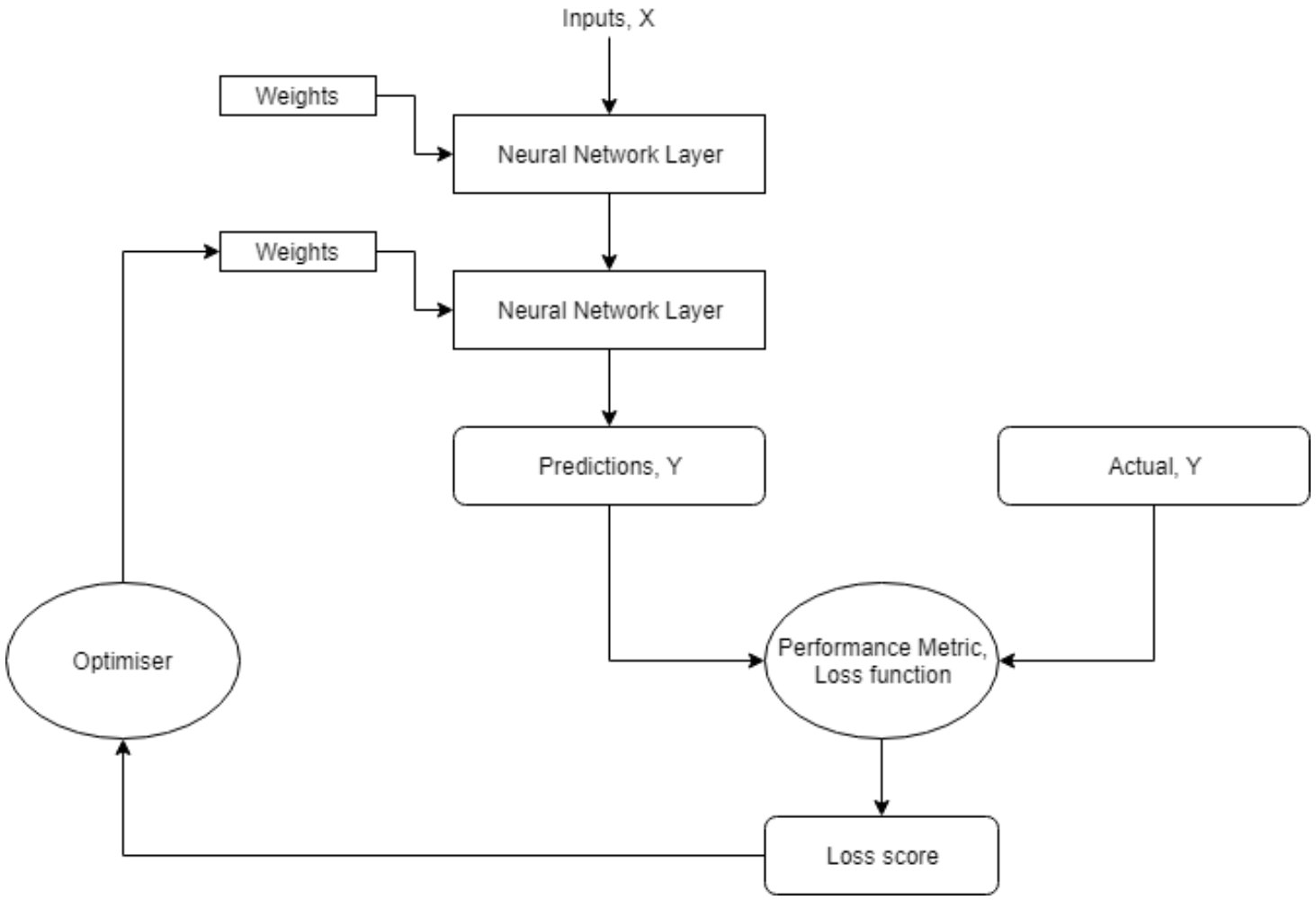
- Example of a NN architecture
- VGG16

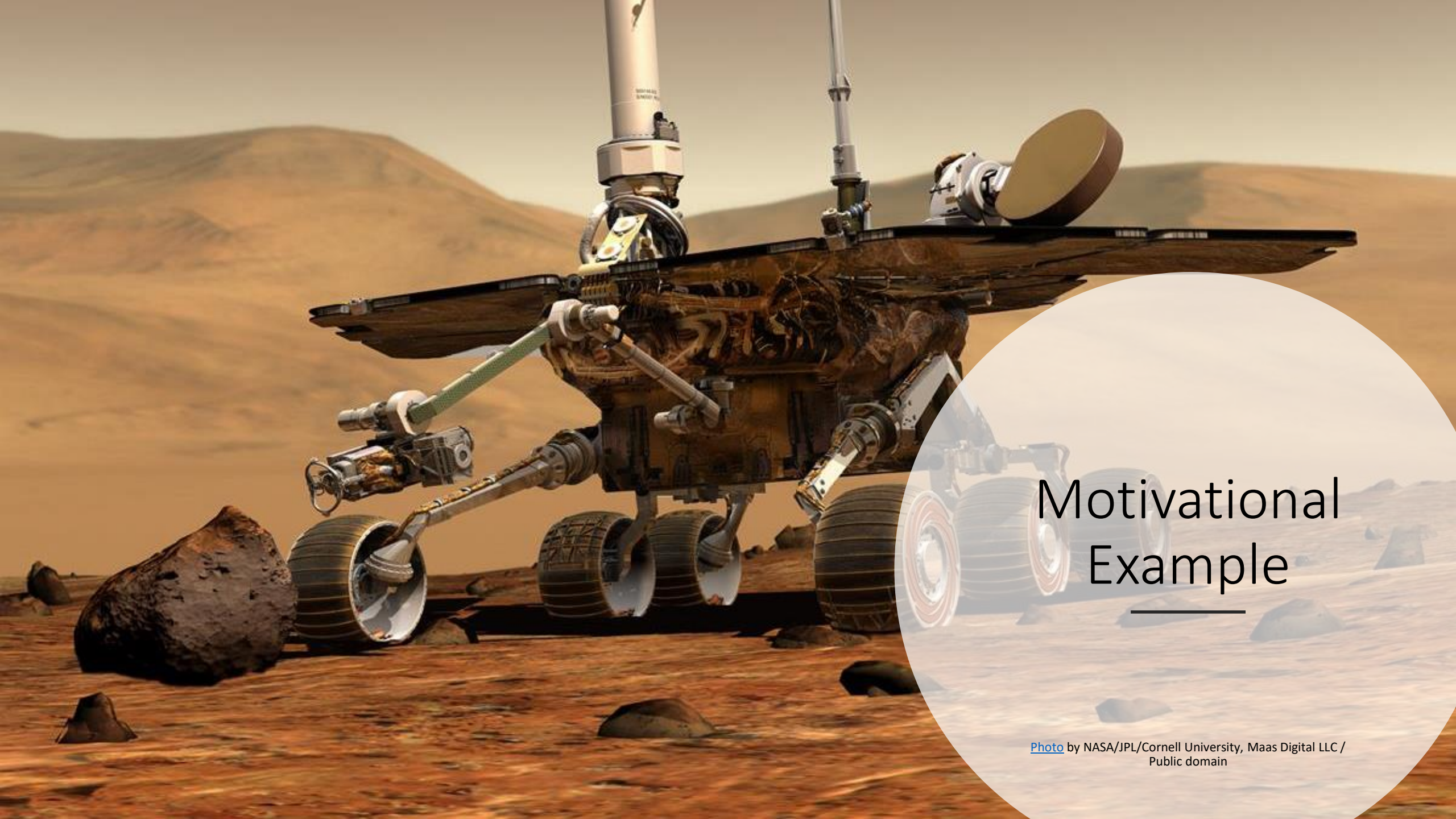
- NB:
  - Fully connected layer not shown

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

Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

Neural Network





Motivational Example

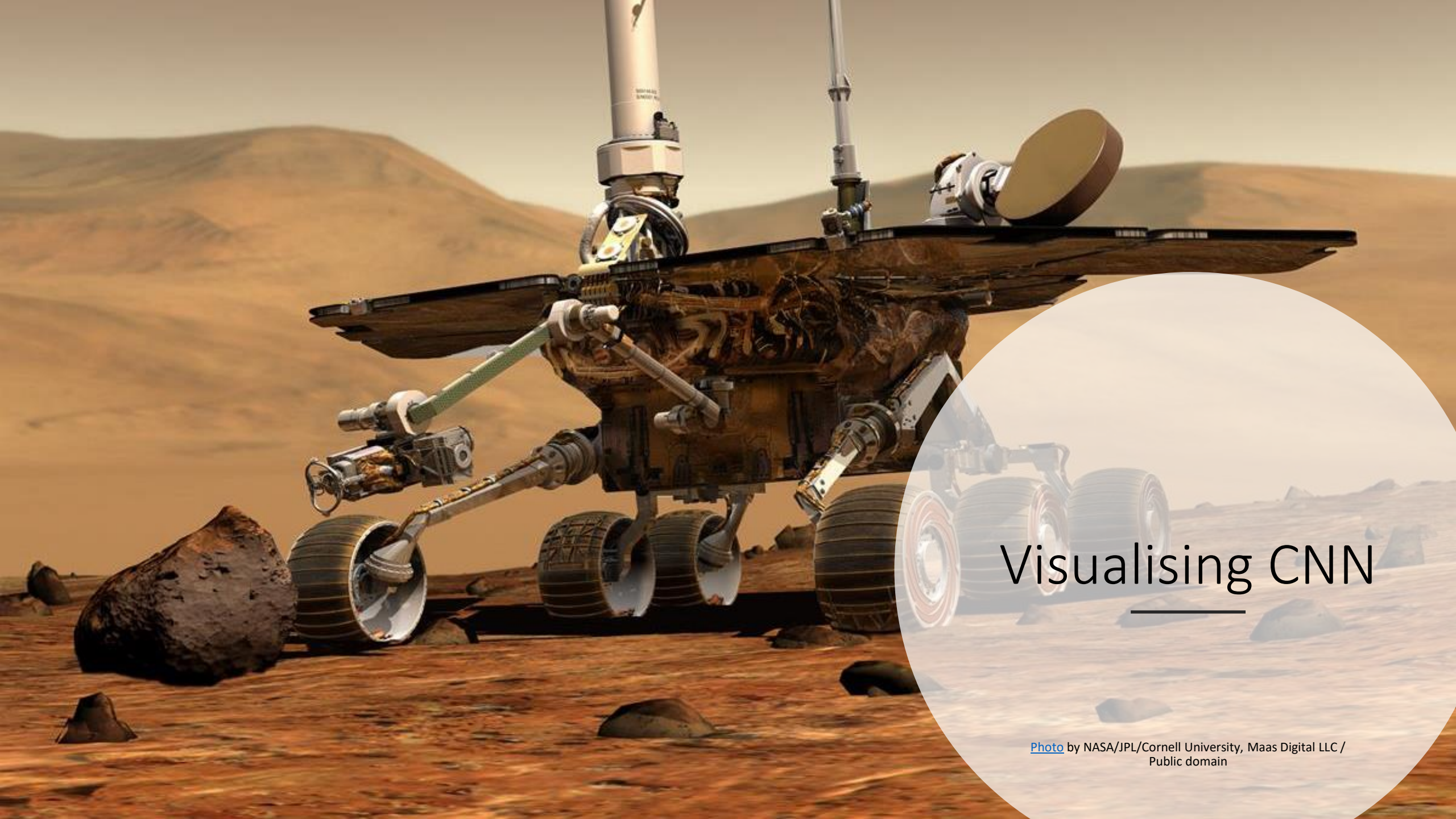
Photo by NASA/JPL/Cornell University, Maas Digital LLC / Public domain

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

Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

Image Kernels

Image Kernels

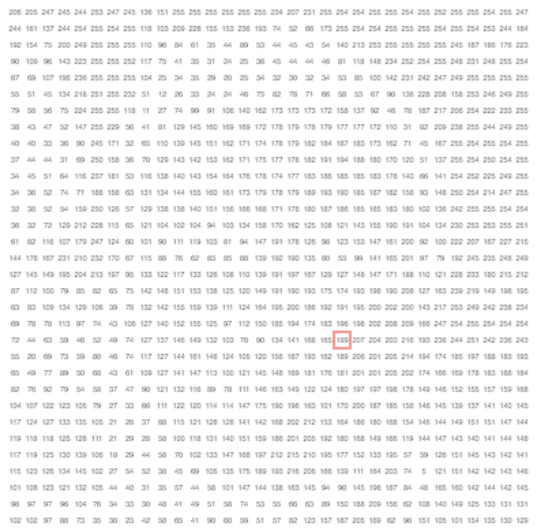
Explained Visually

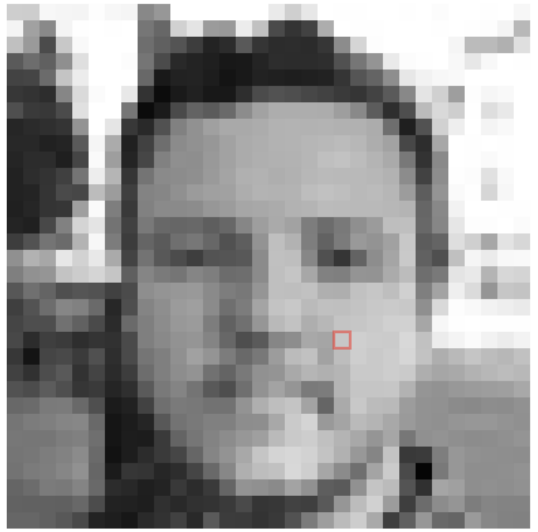
Tweet Like 903 Share

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.

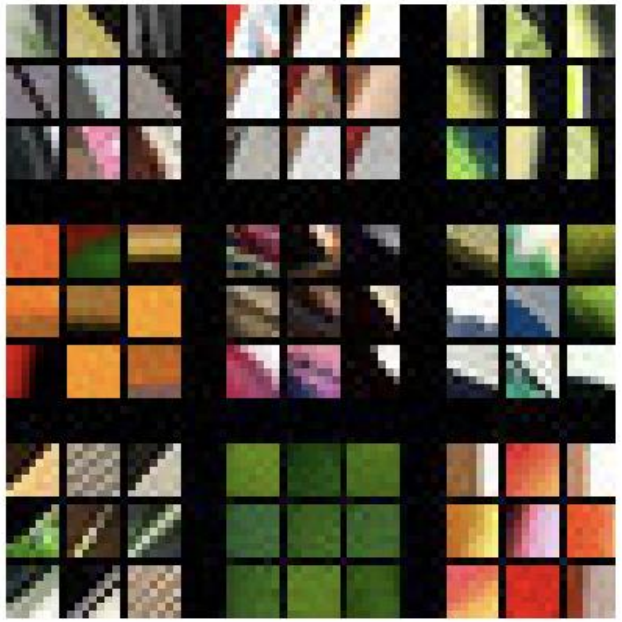




Source: http://setosa.io/ev/image-kernels/

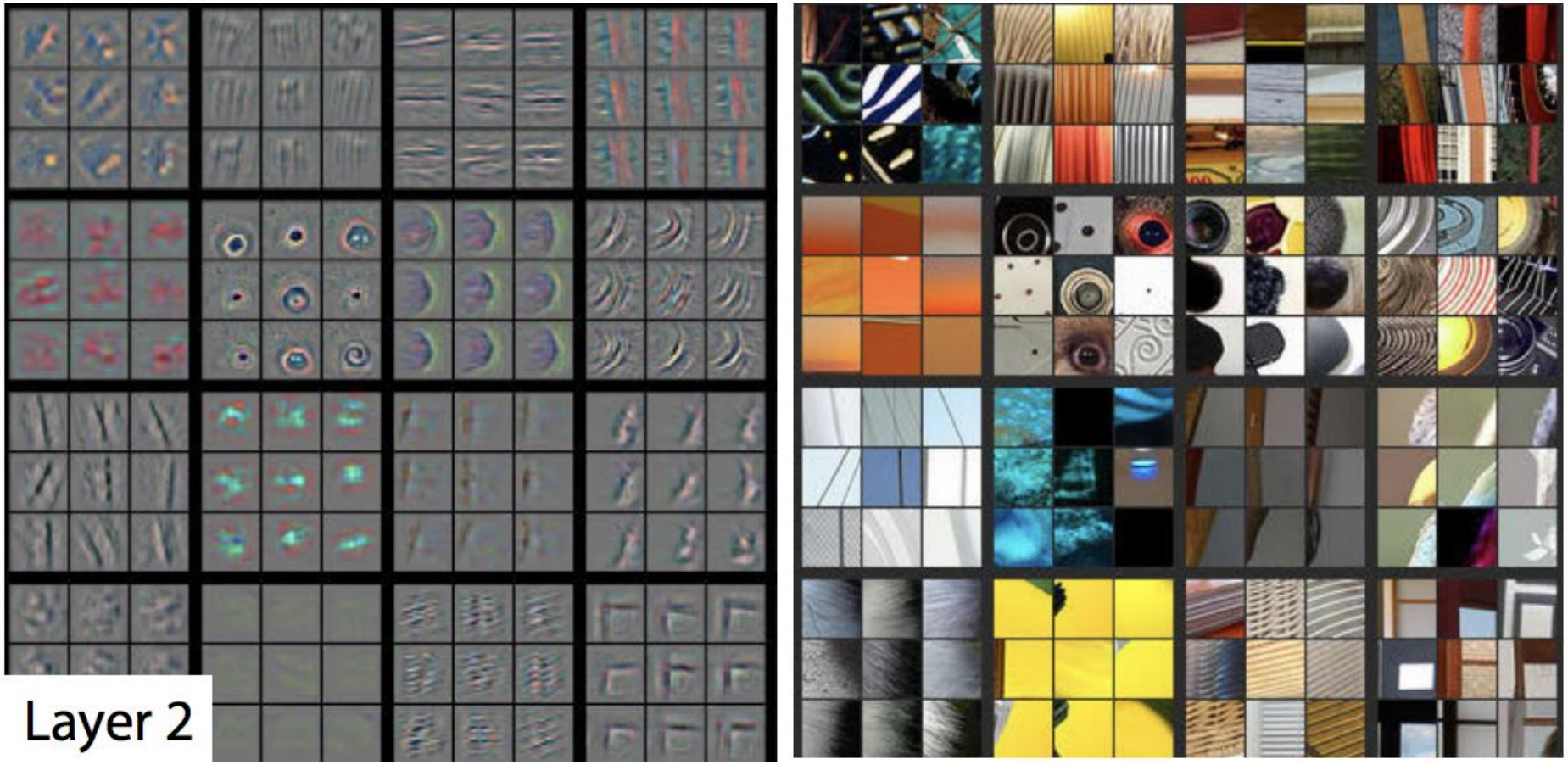
Visualizing and Understanding Convolutional Networks by Matthew D. Zeiler and Rob Fergus (2013)

Layer 1



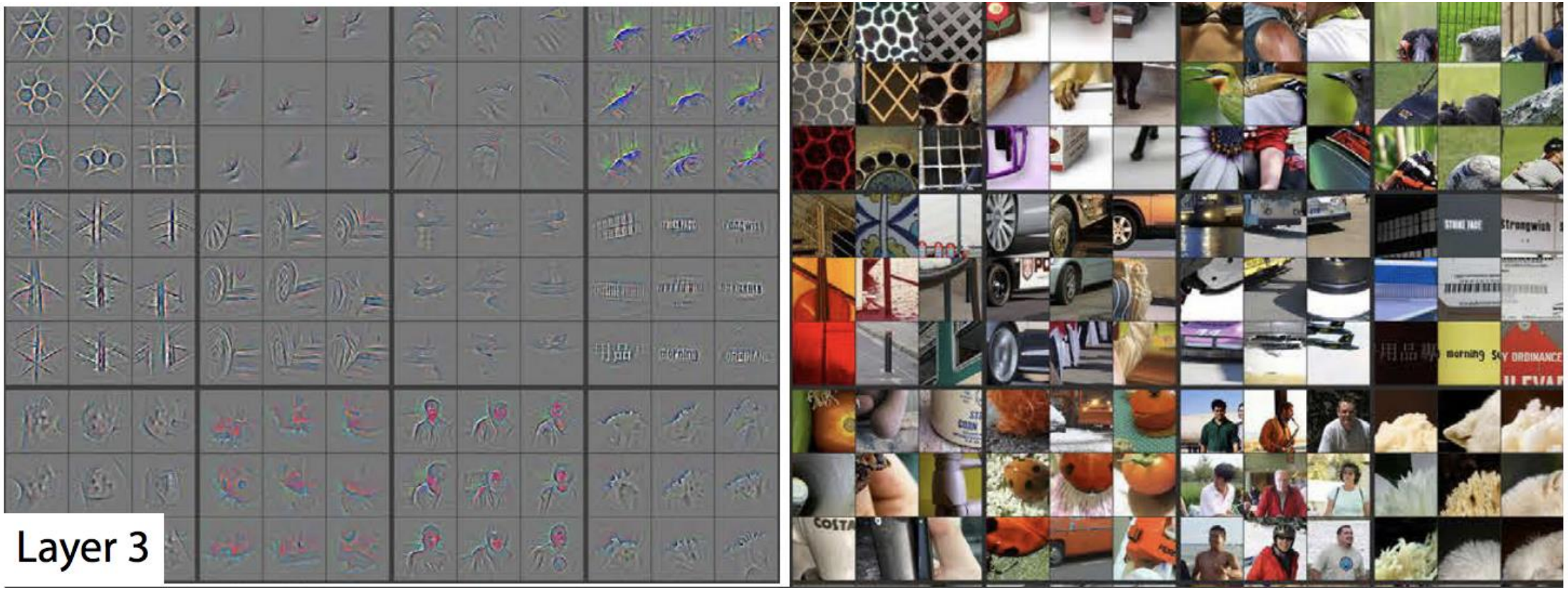
Source: https://arxiv.org/abs/1311.2901

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



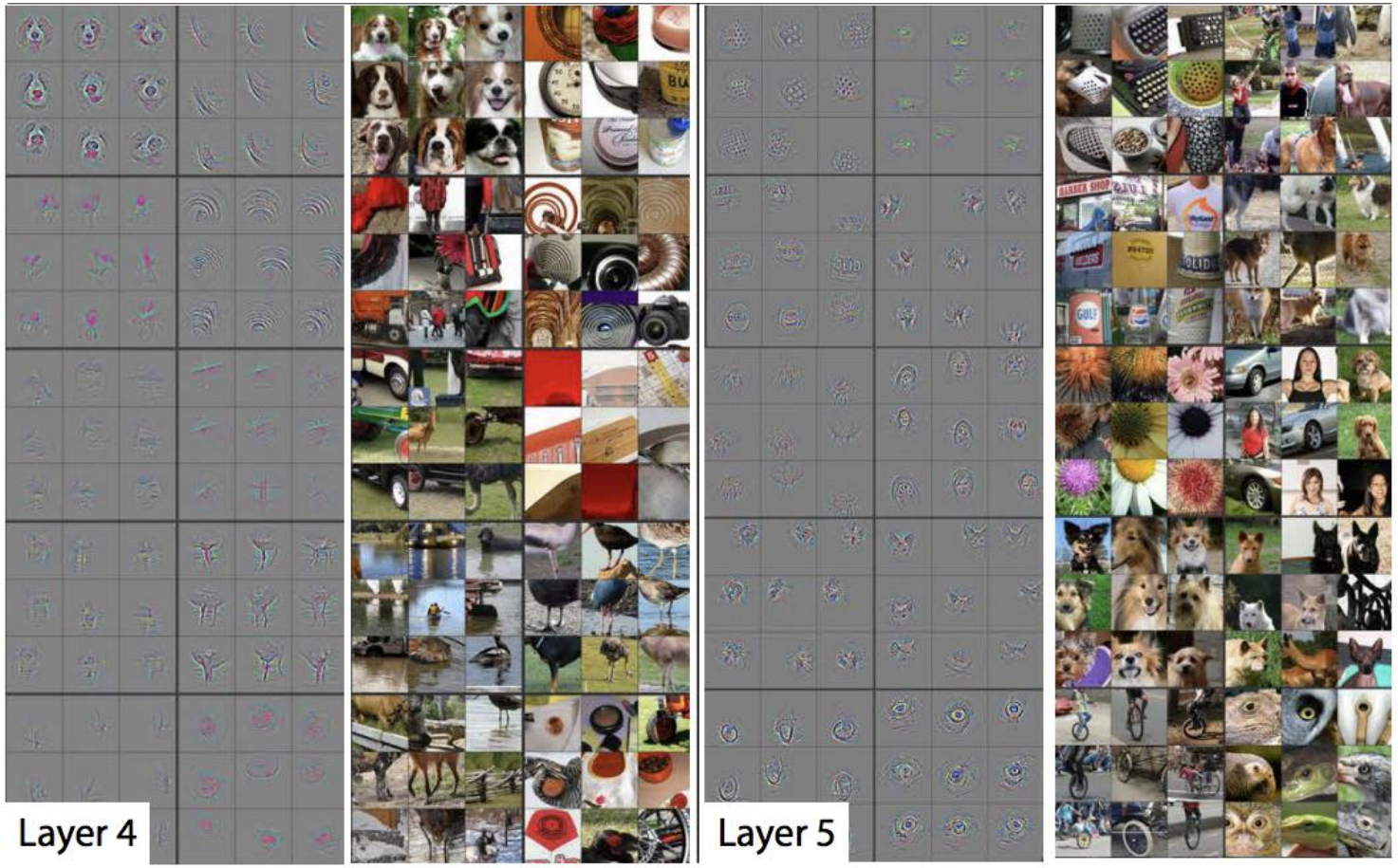
Source: https://arxiv.org/abs/1311.2901

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



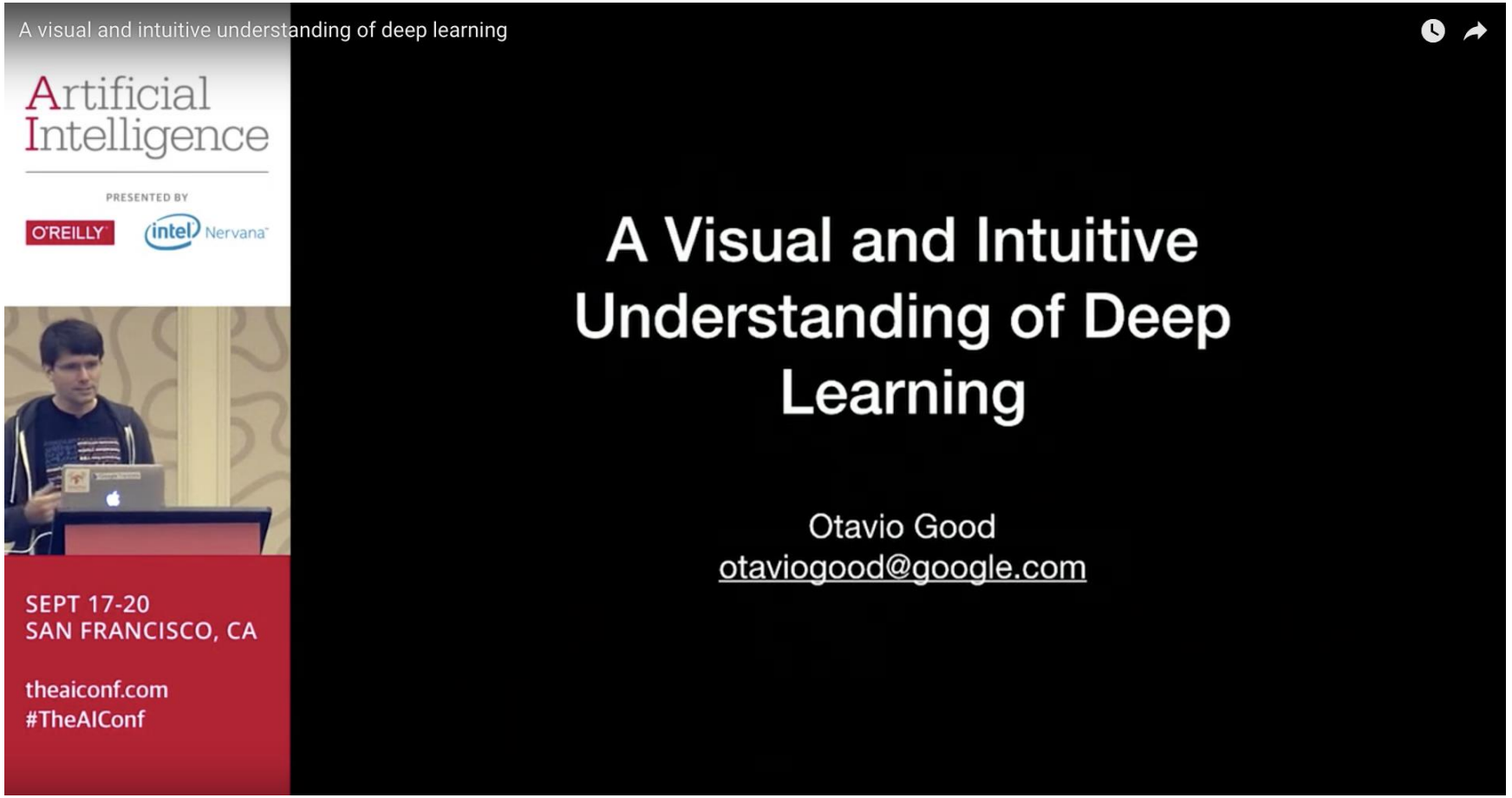
Source: https://arxiv.org/abs/1311.2901

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



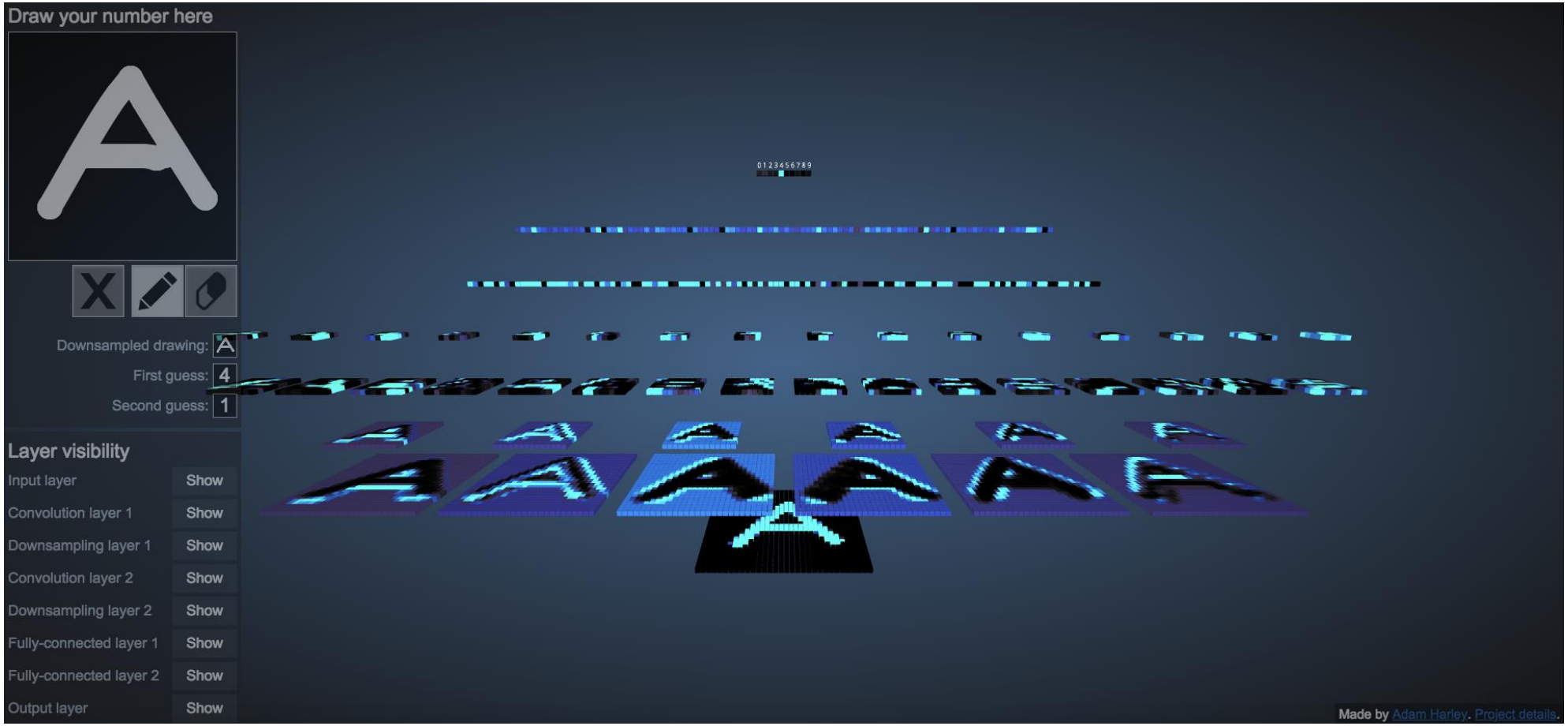
Source: https://arxiv.org/abs/1311.2901

Otavio Good



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

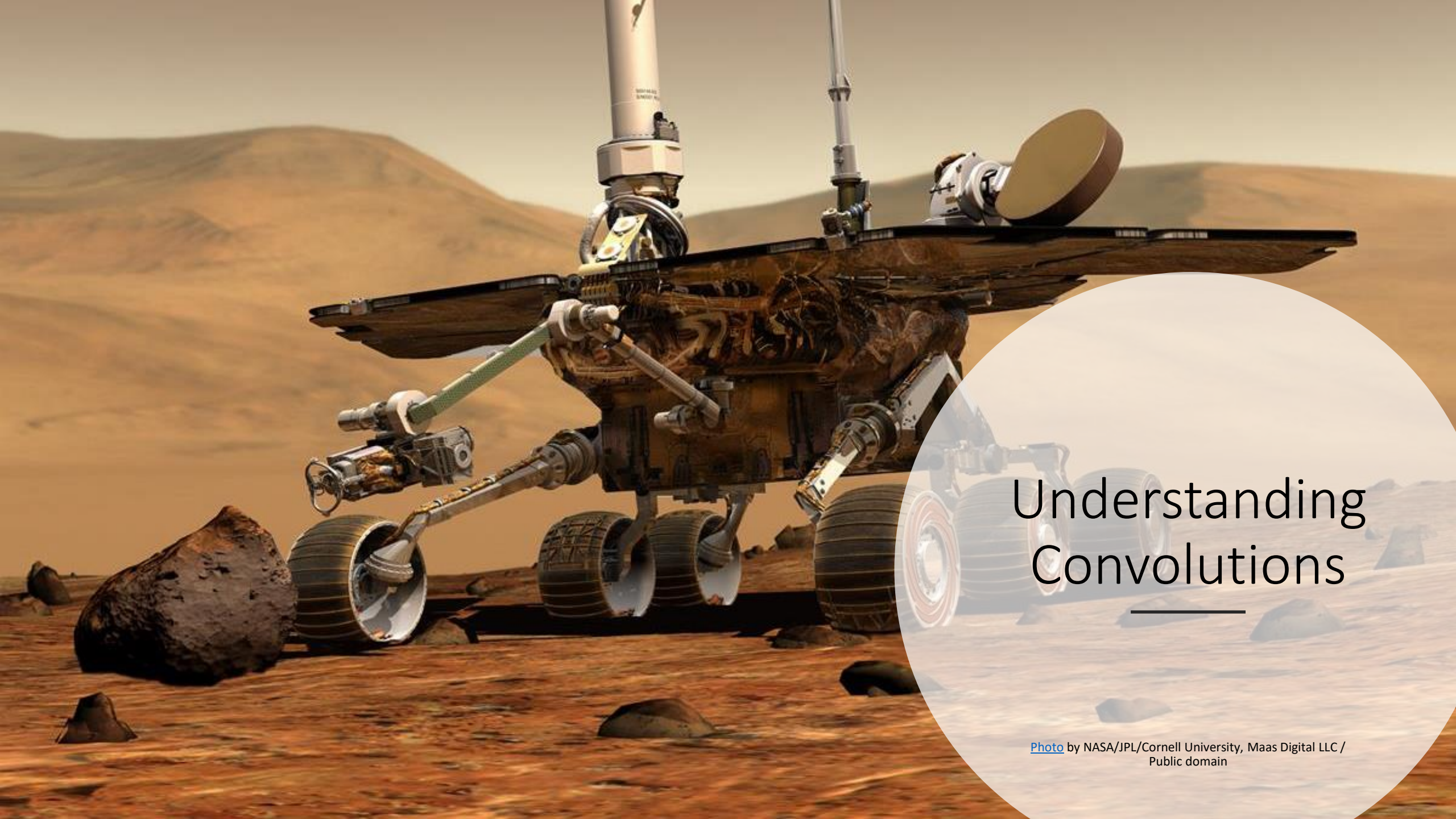
Adam Harley



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

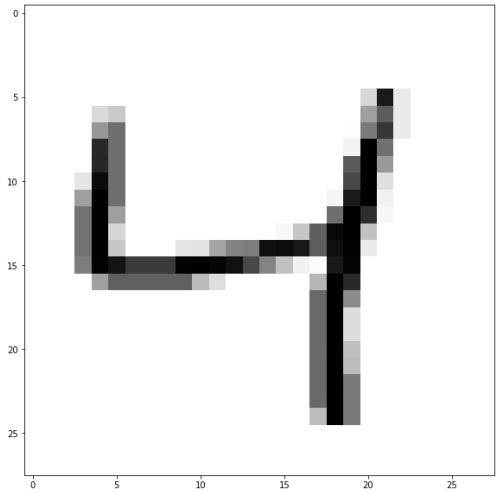
Convolutional Networks (CNN) Properties

- 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 x 28 pixels in 1 channel. Grayscale
  - 28x28 = 784 values
- Colour Images
  - Large
  - 640 x 480 x 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

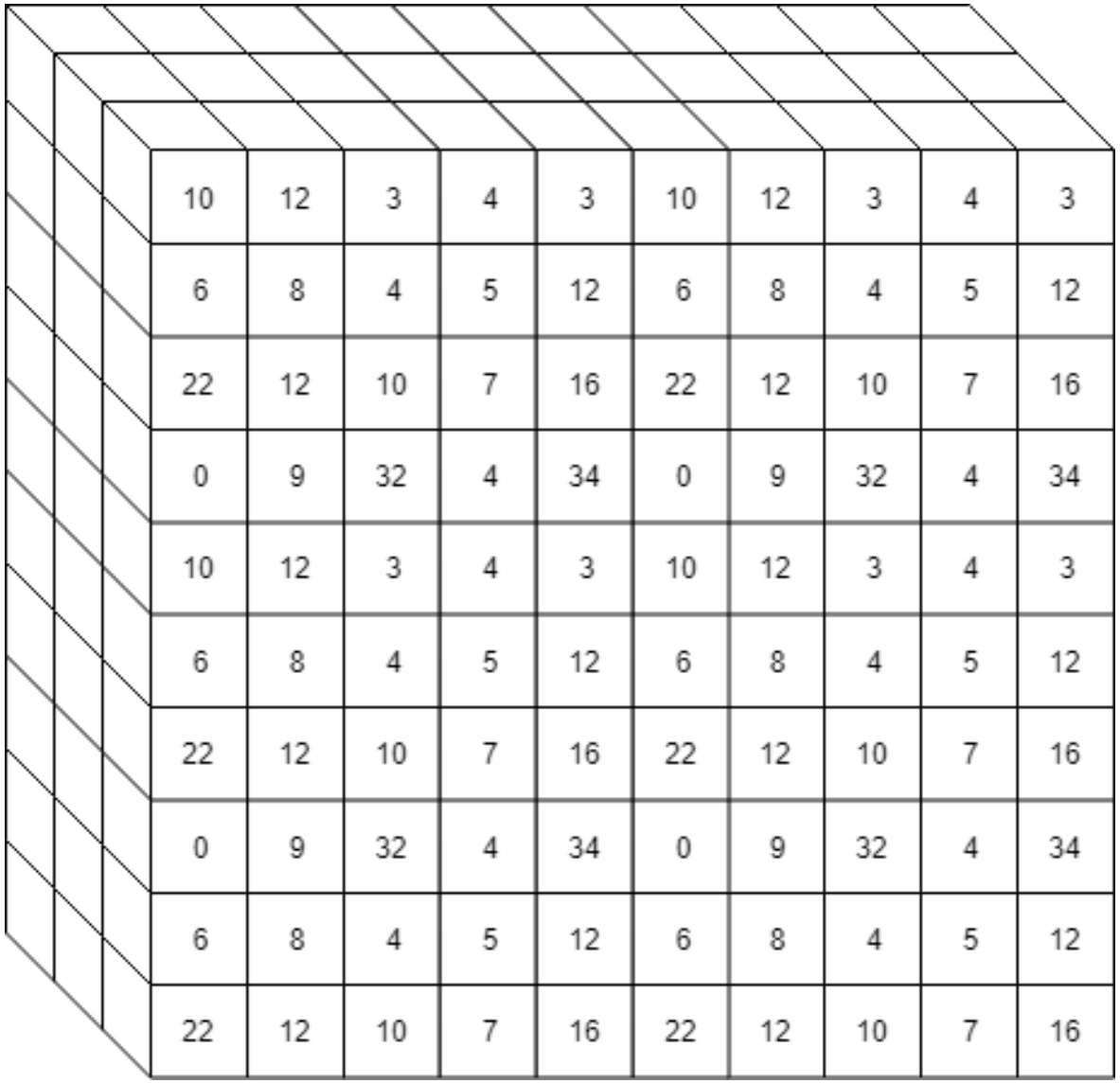
- This is an example of a black a white picture

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

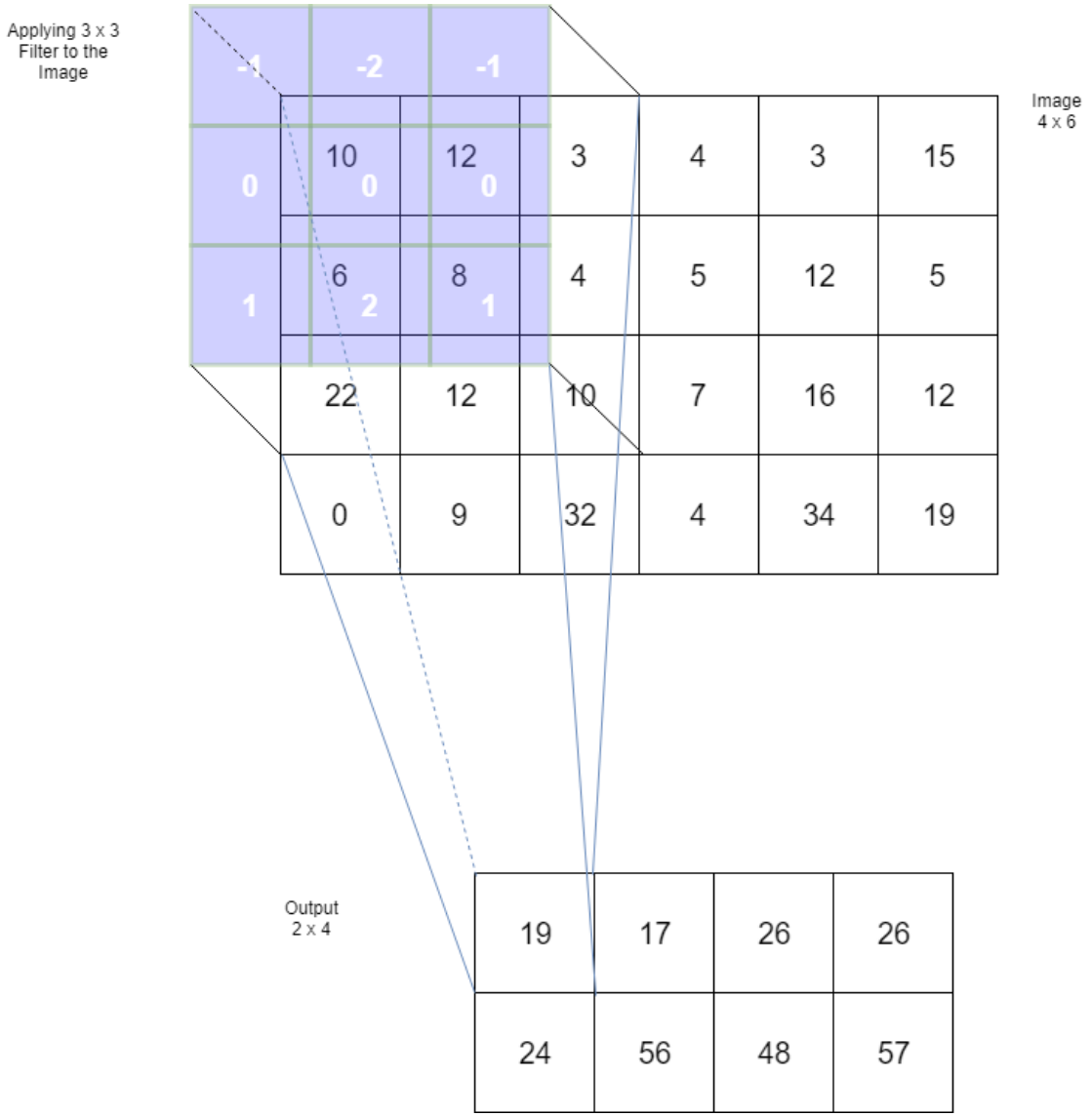
FC

OUTPUT

Convolution Operation

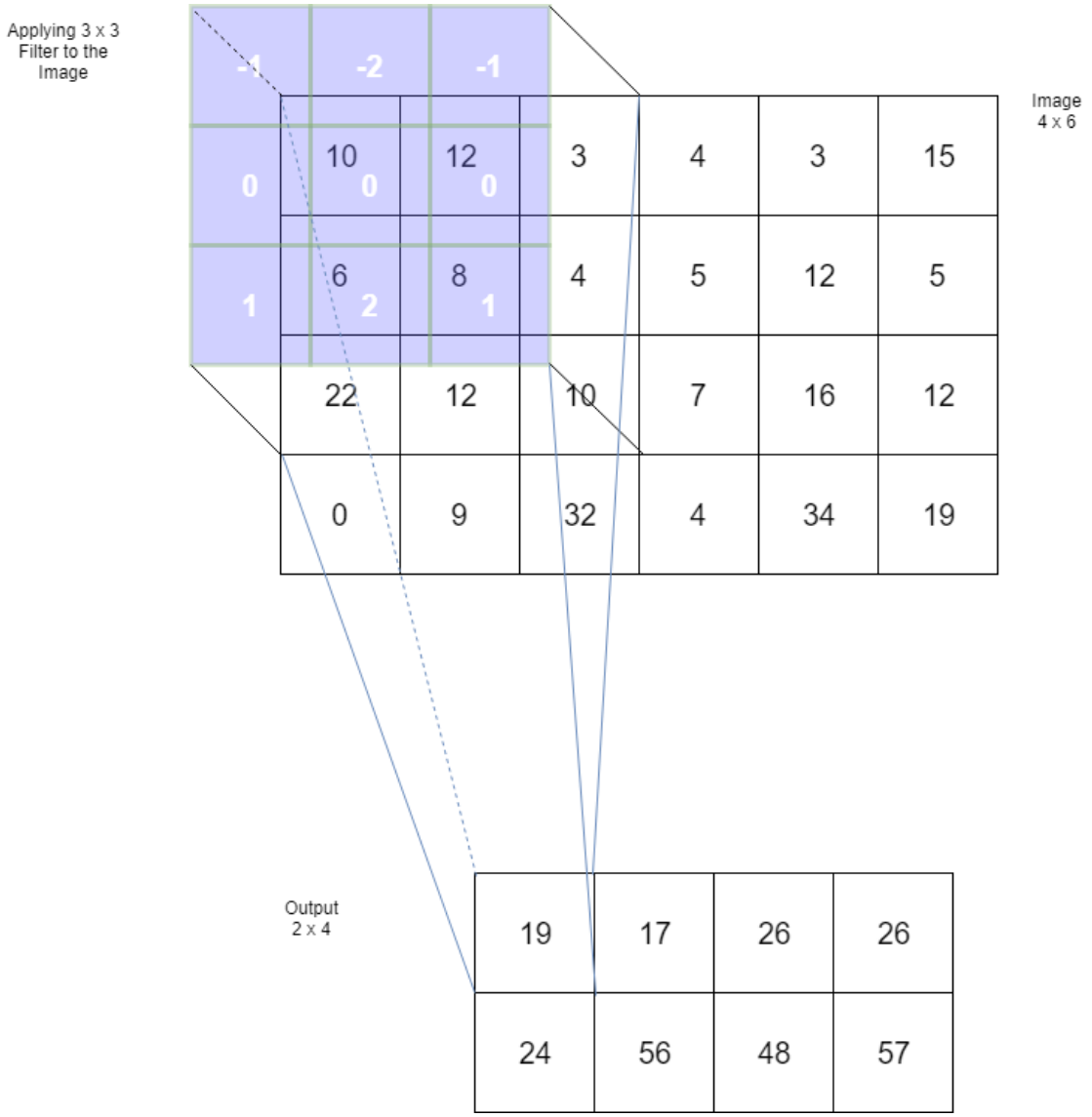
- Sum of Element-wise multiplication
- (-1 x 10) + (-2 x 12) + (-1 x 3) + (0 x 6) + (0 x 8) + (0 x 4) + (1 x 22) + (2 x 12) + (1 x 10) = 19

- convolution operation.xlsx
- “stride=1” worksheet



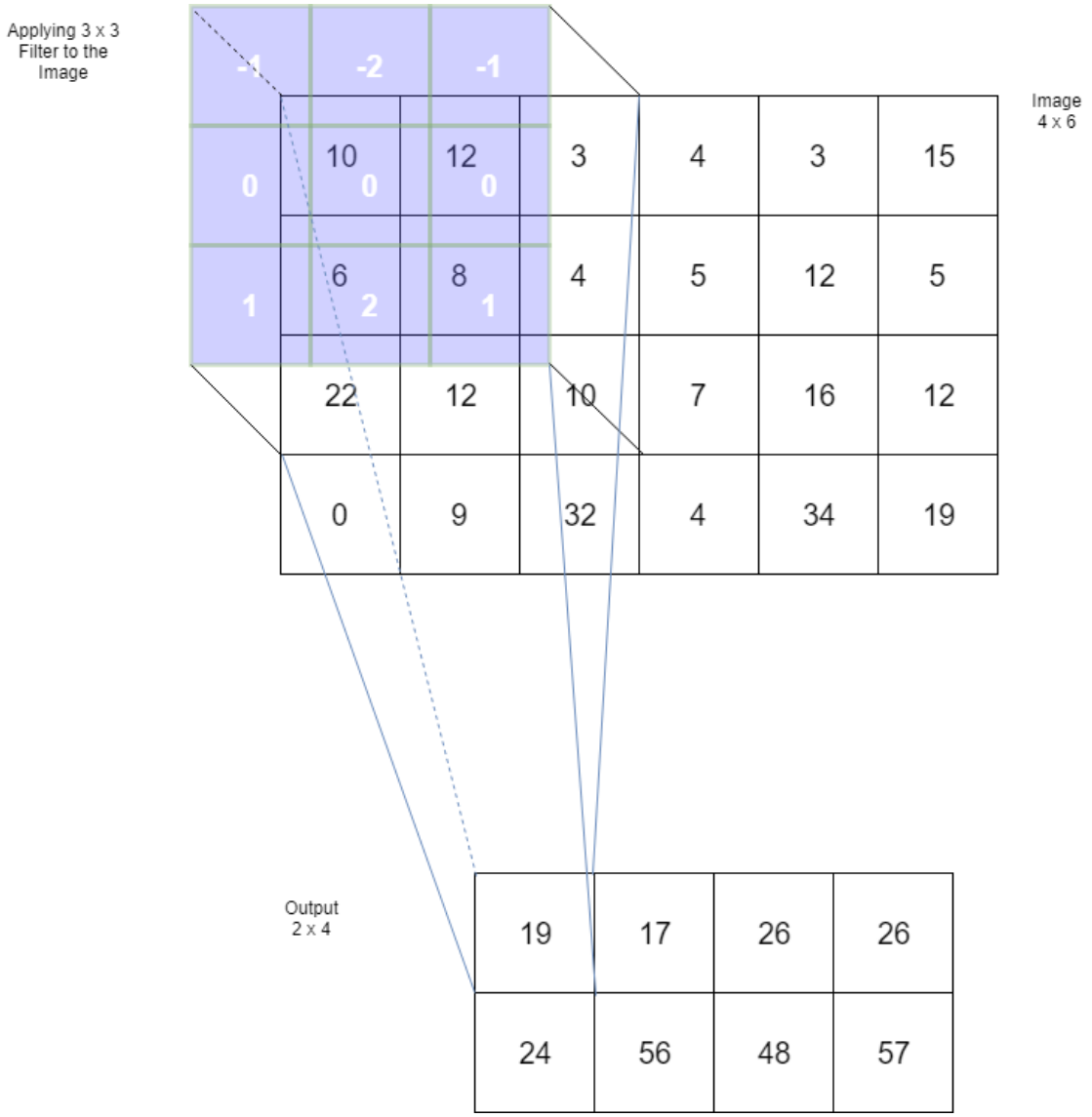
Convolution Example

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



Convolution Example

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



Kernel /Filter

```
from keras import models
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
cnn = models.Sequential()
cnn.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
cnn.add(MaxPooling2D(2, 2))
cnn.add(Conv2D(64, (3, 3), activation='relu'))
cnn.add(MaxPooling2D(2, 2))
cnn.add(Conv2D(64, (3, 3), activation='relu'))
cnn.add(Flatten())
cnn.add(Dense(64, activation='relu'))
cnn.add(Dense(10, activation='softmax'))
cnn.compile(optimizer='rmsprop',
            loss='categorical_crossentropy',
            metrics=['accuracy'])
cnn.fit(x_train_dat, y_train_dat, epochs=10,
        batch_size=64,
        validation_data=(x_val, y_val));
print(cnn.summary())
test_loss, test_acc = cnn.evaluate(x_test, y_test)
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

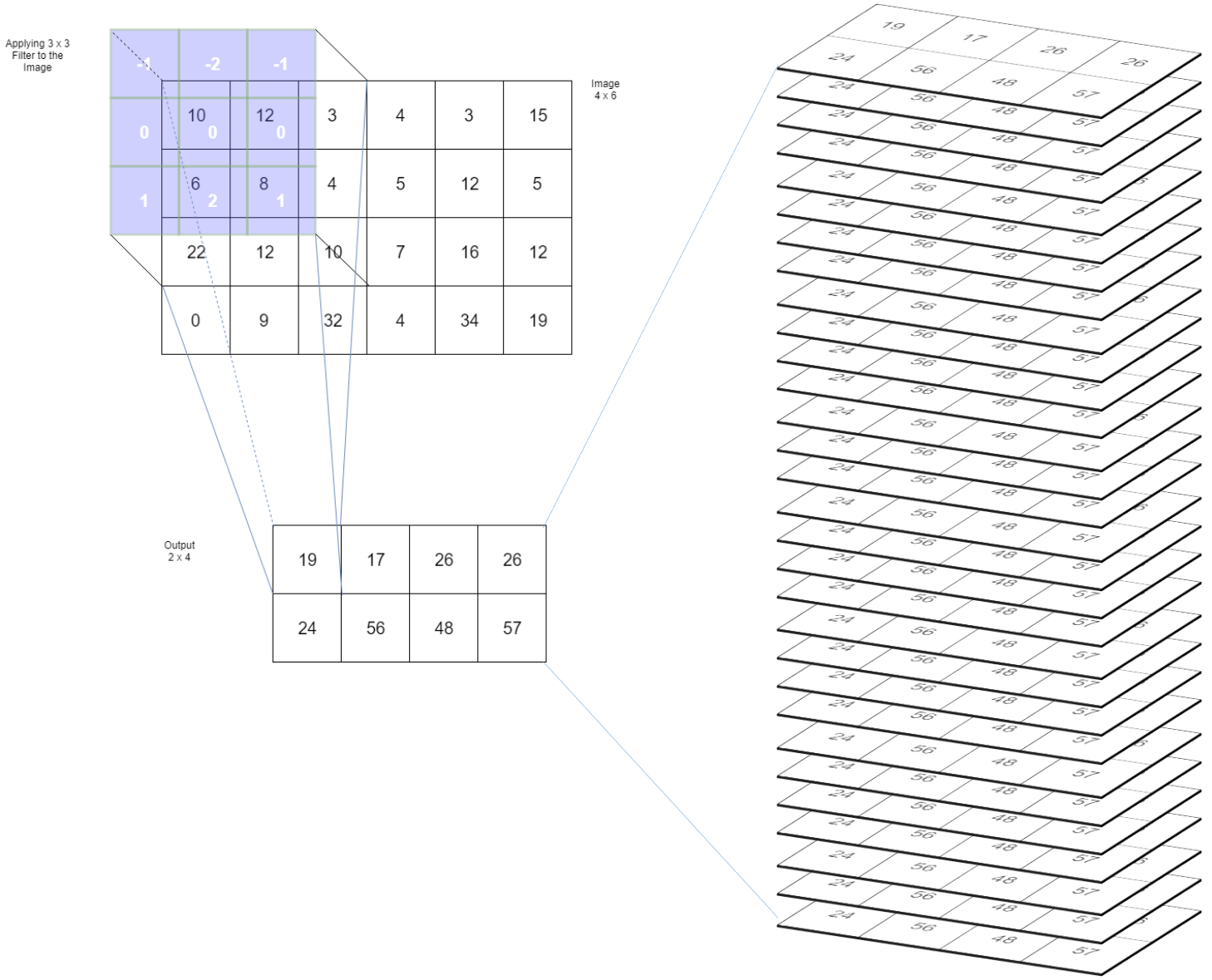
Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_3 (MaxPooling2	(None, 13, 13, 32)	0
conv2d_5 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_4 (MaxPooling2	(None, 5, 5, 64)	0
conv2d_6 (Conv2D)	(None, 3, 3, 64)	36928
flatten_2 (Flatten)	(None, 576)	0
dense_5 (Dense)	(None, 64)	36928
dense_6 (Dense)	(None, 10)	650

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 x 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/

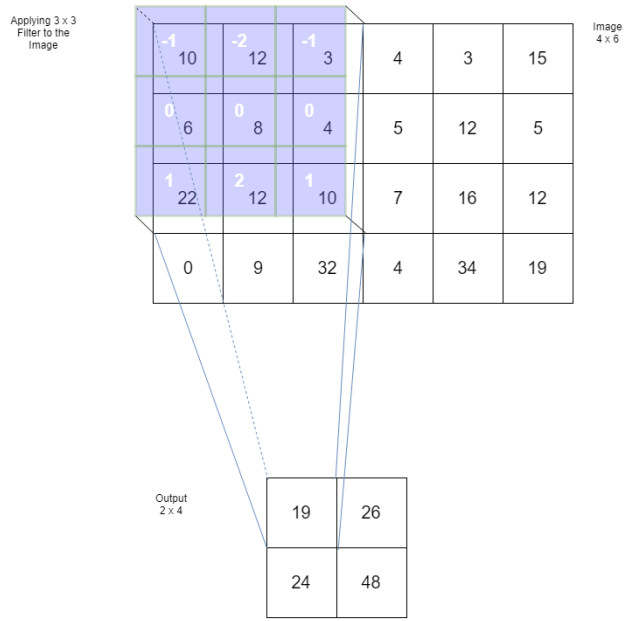
Filters

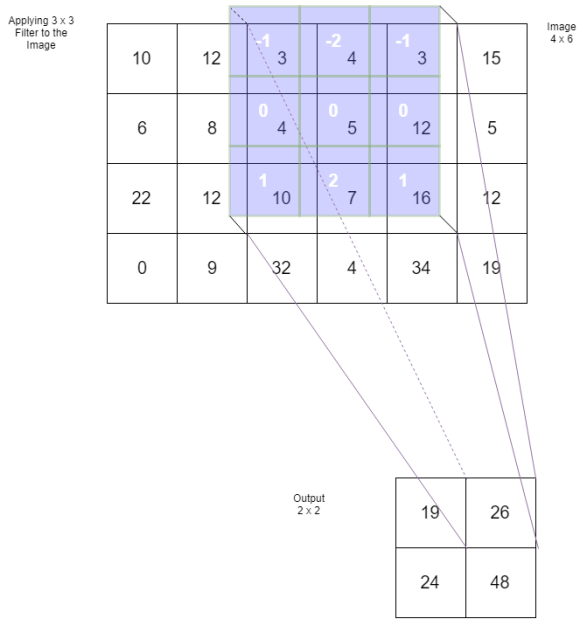
- A few more things to note that affect the output dimension:
  - The depth of the output relates to the **number of filters** (`filters in Keras Conv2D`) applied
  - **Strides** (`strides=(1,1) in Keras`) specifying the strides of the convolution along the height and width. The default is 1 for both 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





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

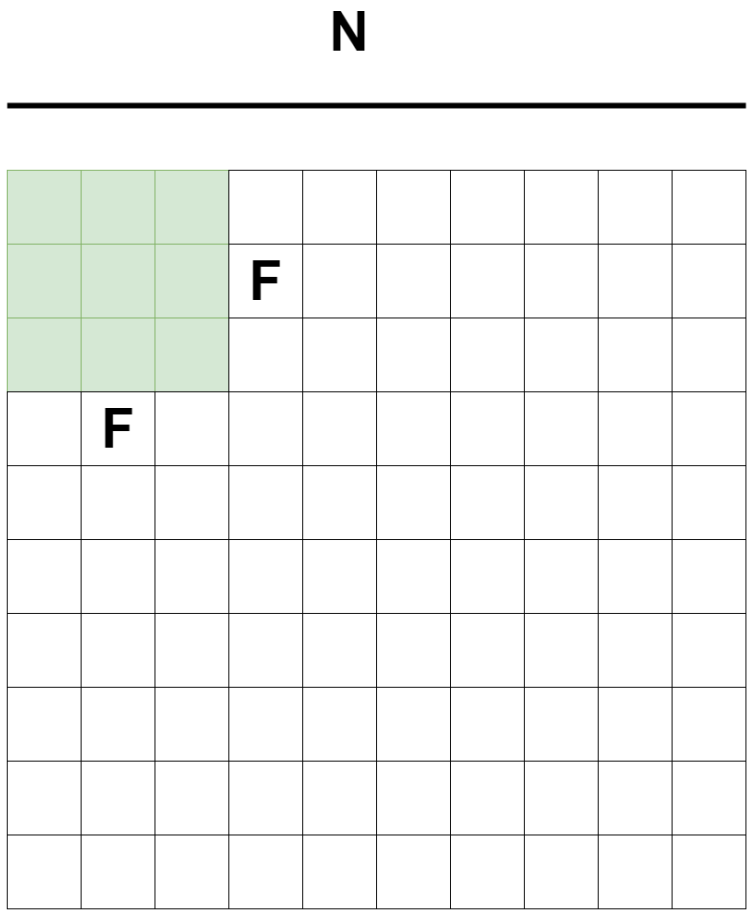
“stride=2” worksheet

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) / Stride + 1

N=10, F = 3

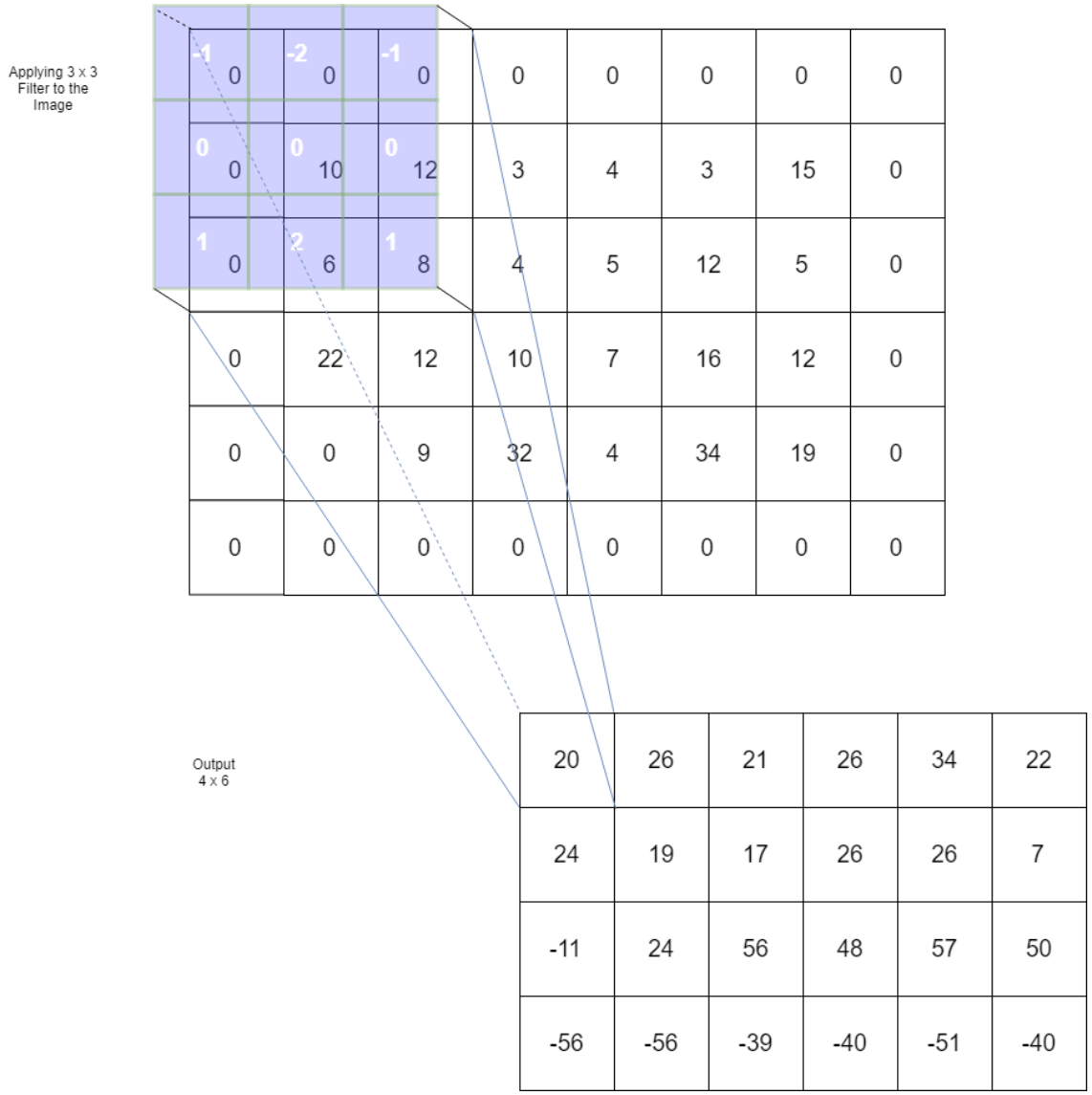
Stride = 1, (10-3)/1 + 1 = 8

Stride = 2, (10-3)/2 + 1 = 4.5

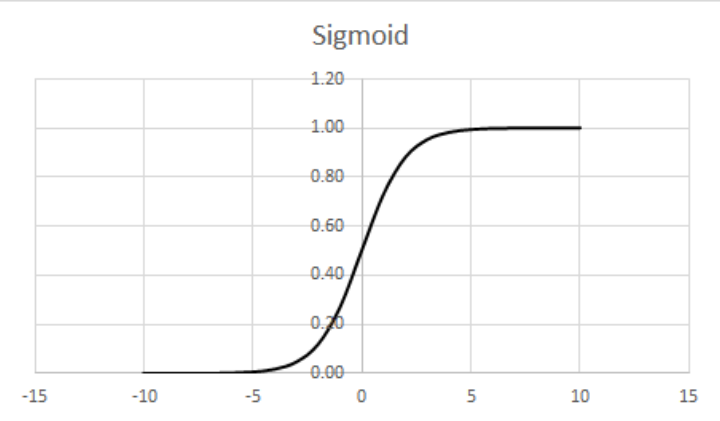
Zero Padding

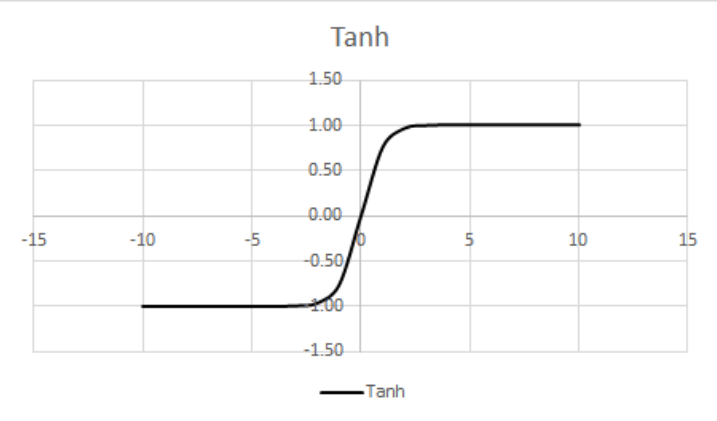
- With zero padding, you are able to recover the original dimension after applying the filter

- convolution operation.xlsx
- “zero padding” worksheet

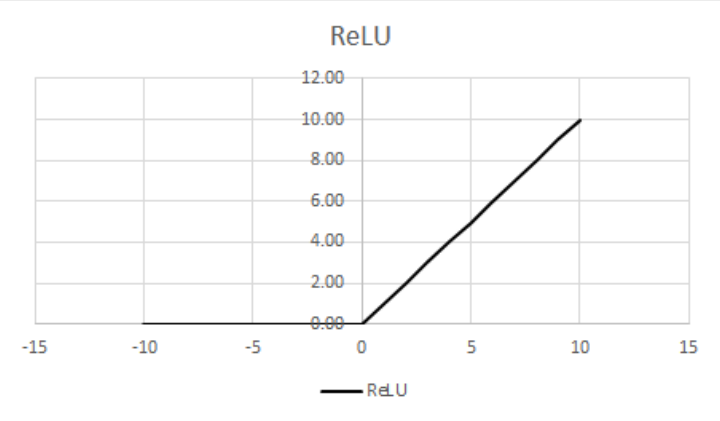


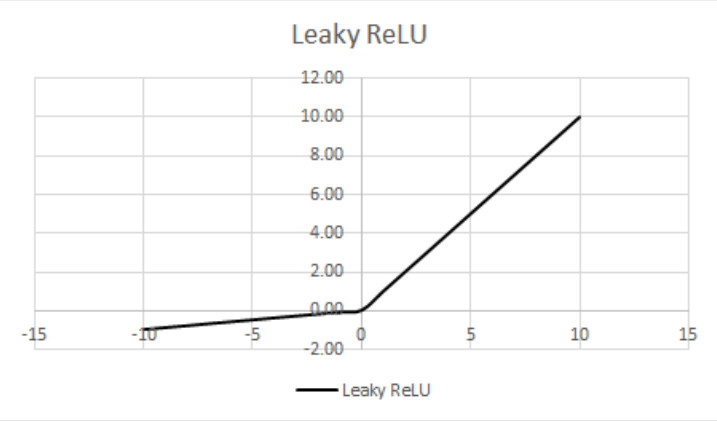
Common Activation Functions

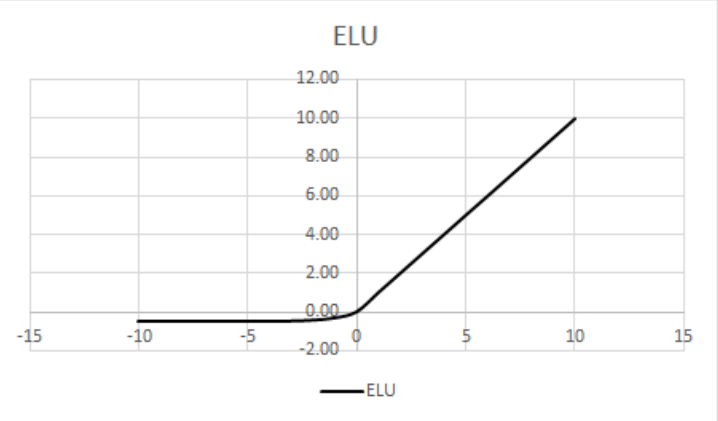




activation fn.xlsx

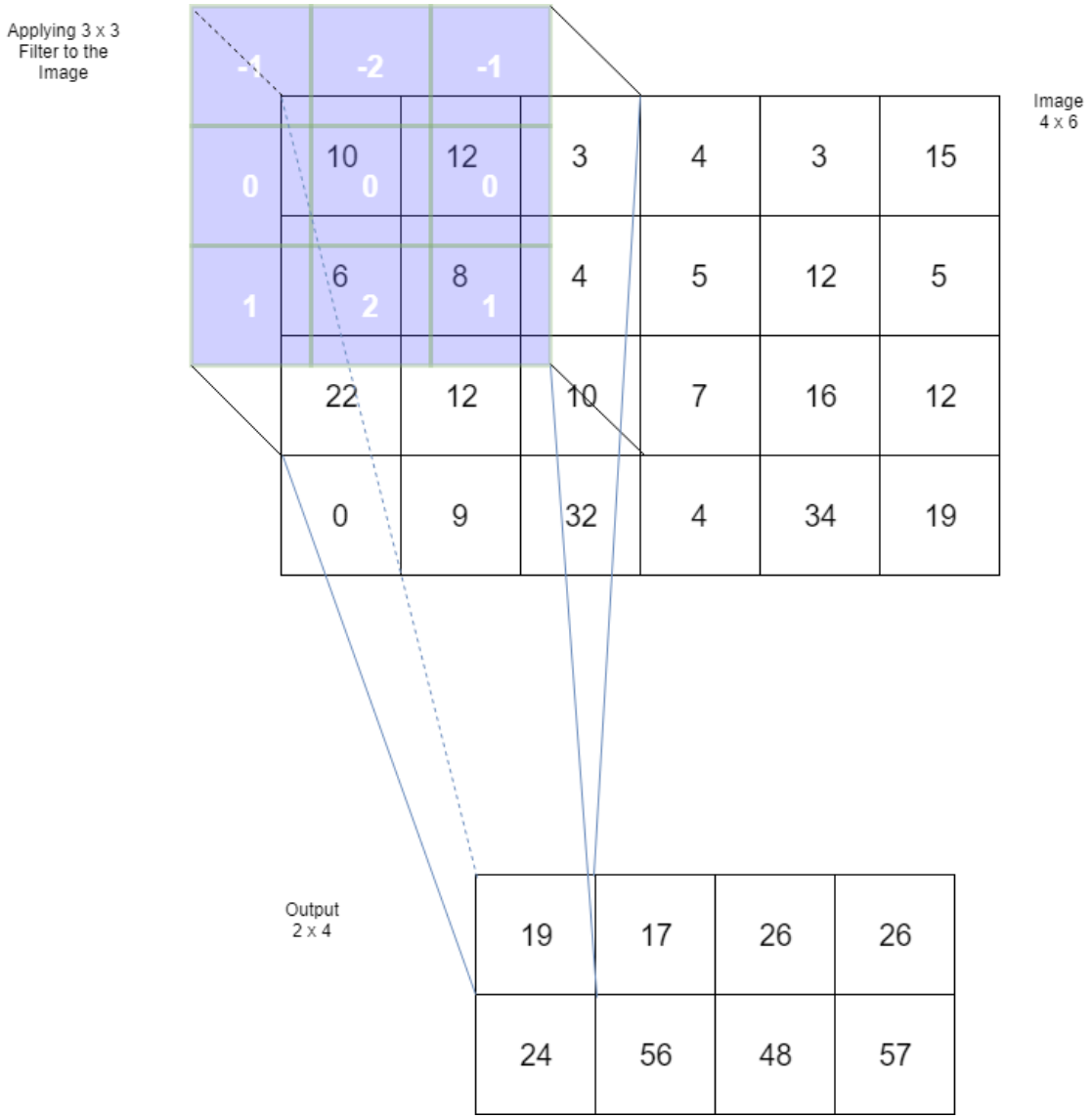






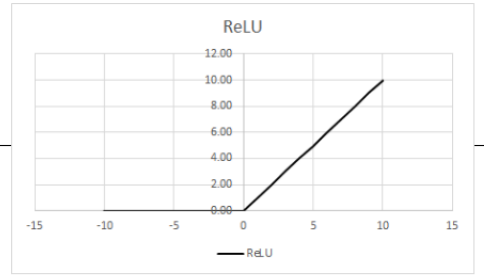
Activation

- 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



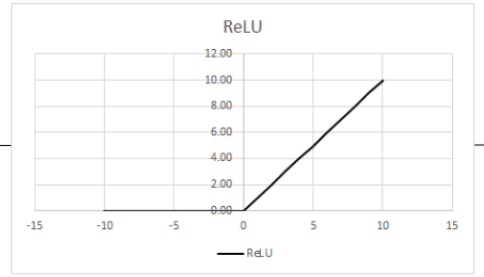
Output + Activation Functions

19	17	26	26
24	56	48	57



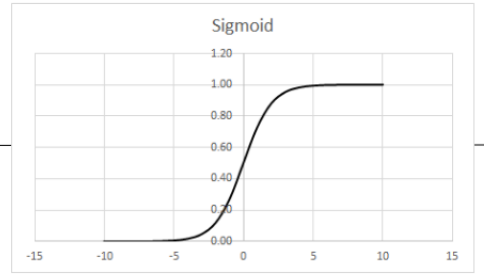
19	17	26	26
24	56	48	57

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



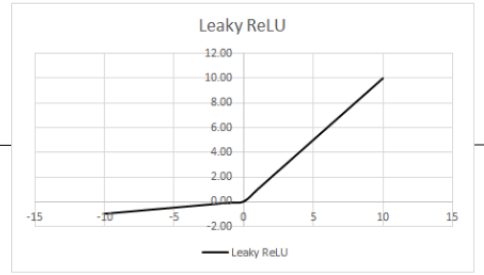
0.5	0.3	0.0	0.6
0.0	0.0	1.4	0.0

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

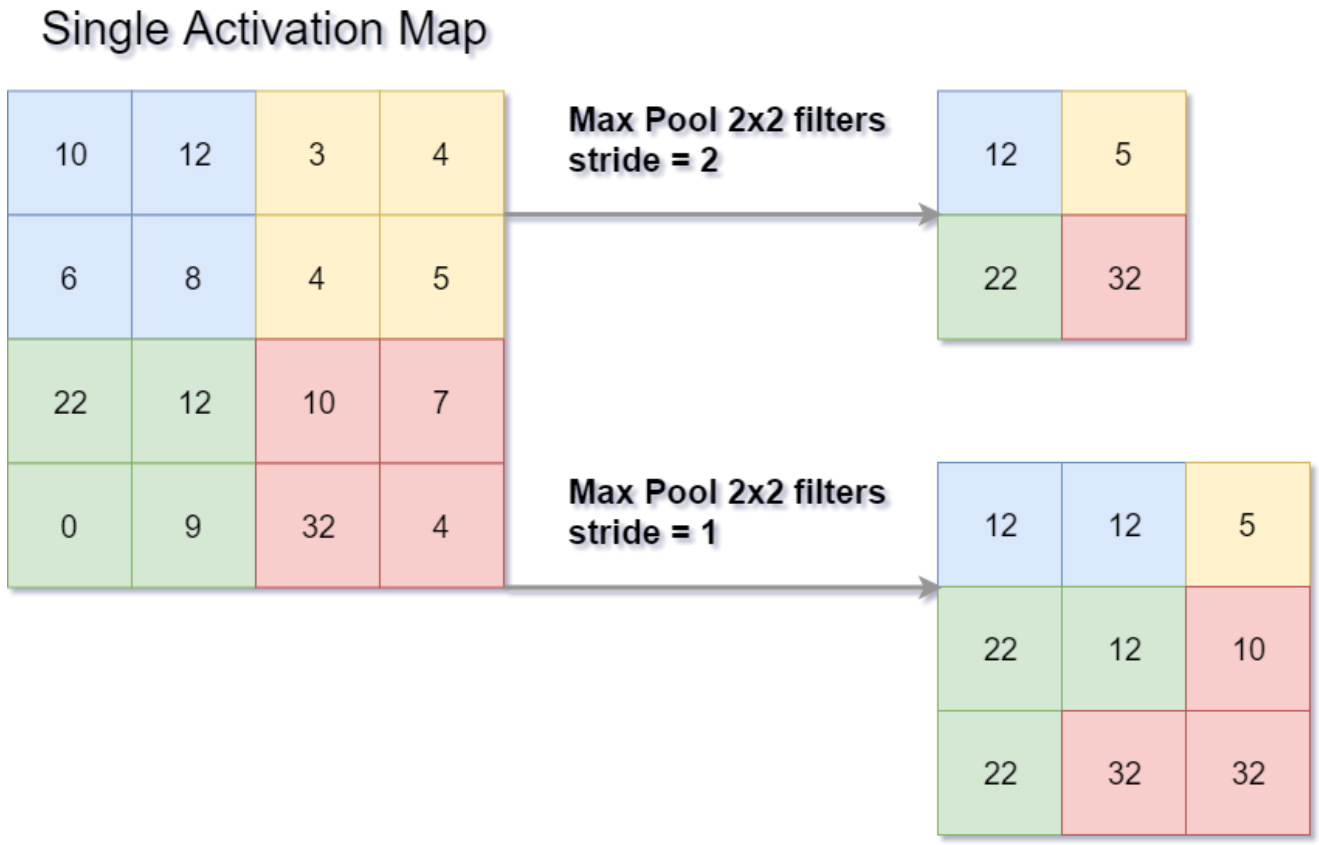
FC

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

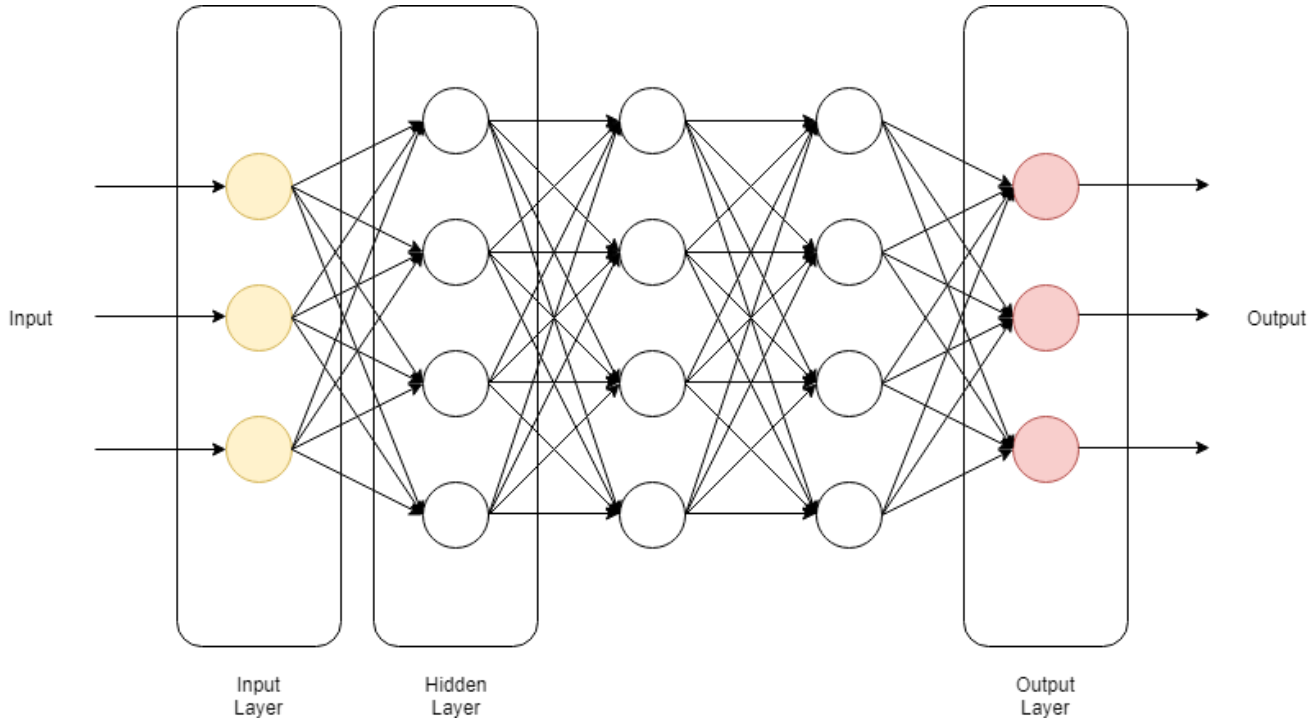


Flatten

- 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

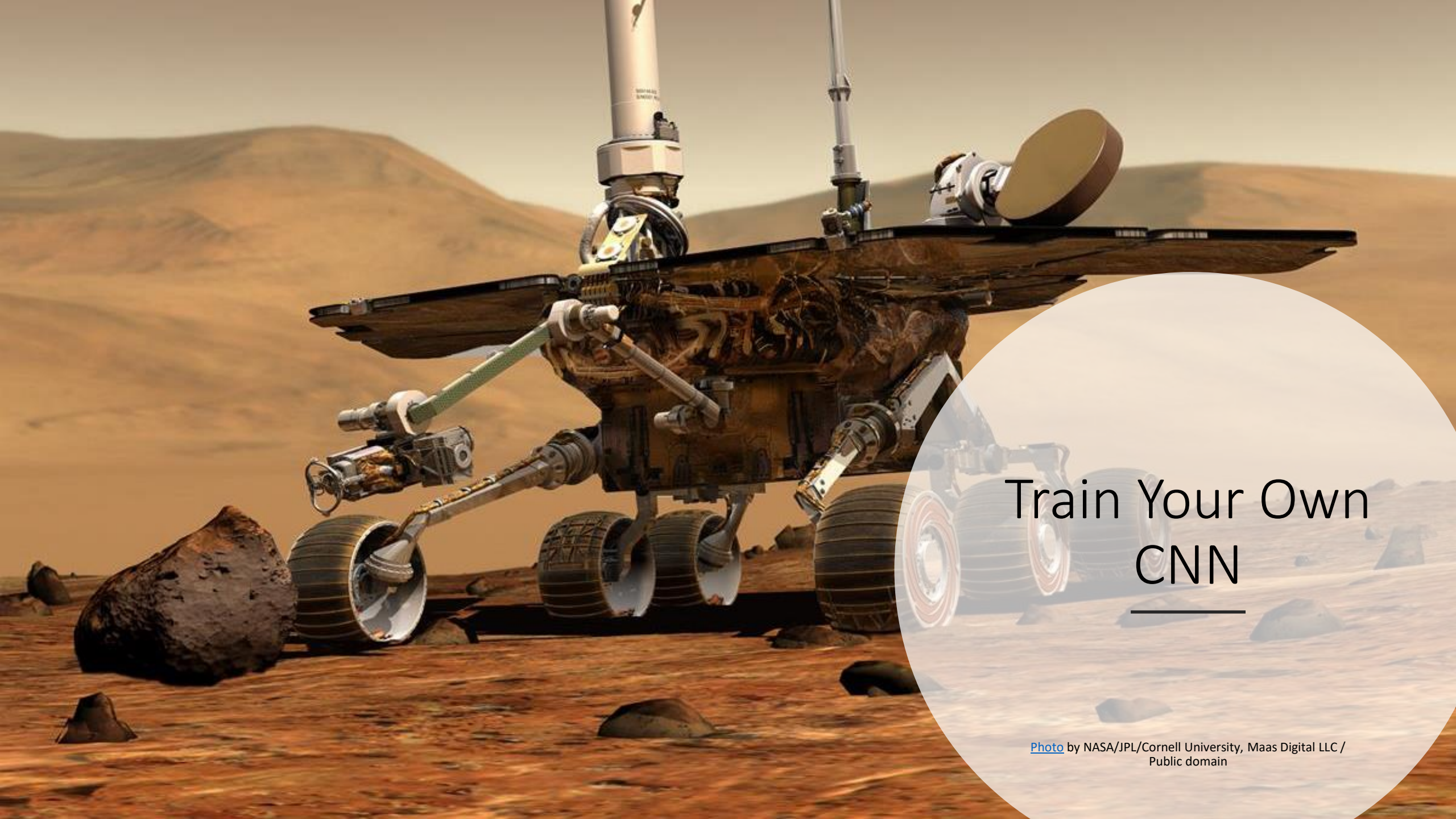
This demo trains a Convolutional Neural Network on the CIFAR-10 dataset in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used this python script to parse 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 verically.

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

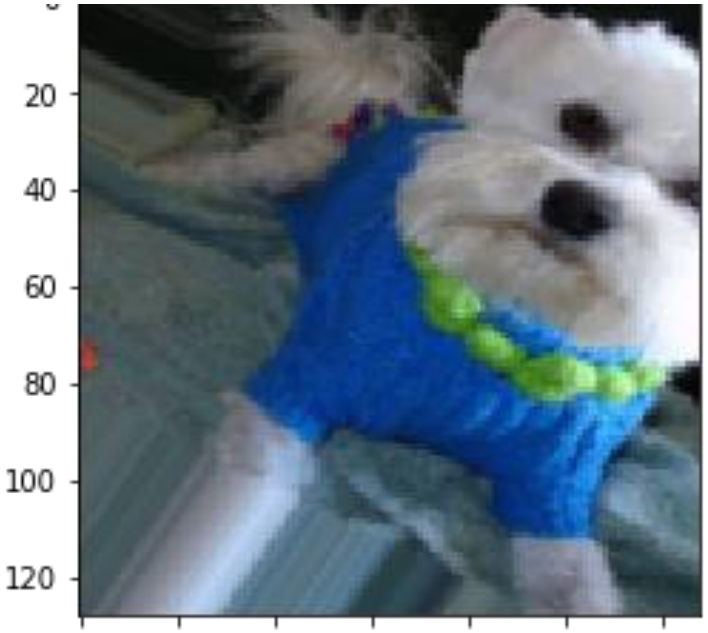
cats_and_dogs.ipynb

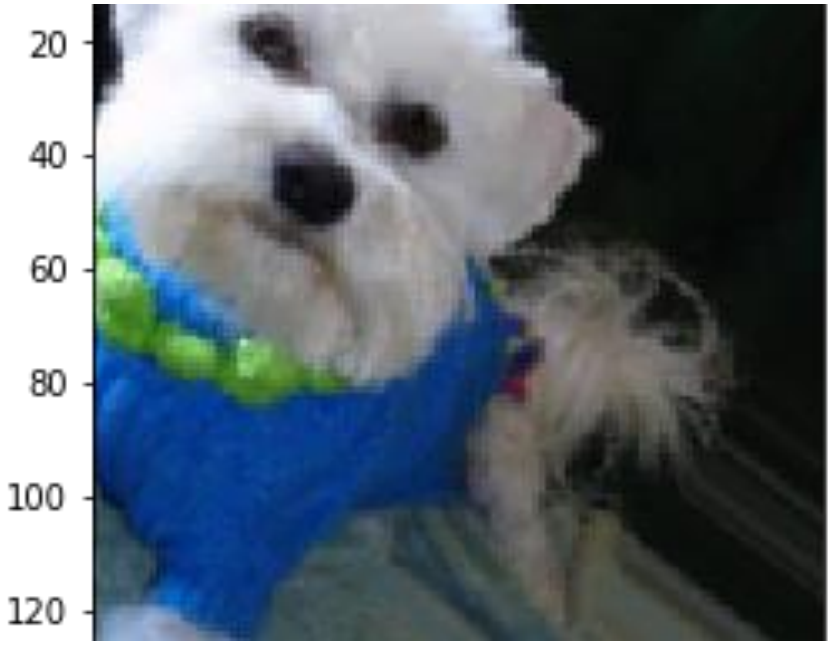
Save and Load Model

- cats_and_dogs.ipynb
- cats_and_dogs_load.ipynb

Data Augmentation

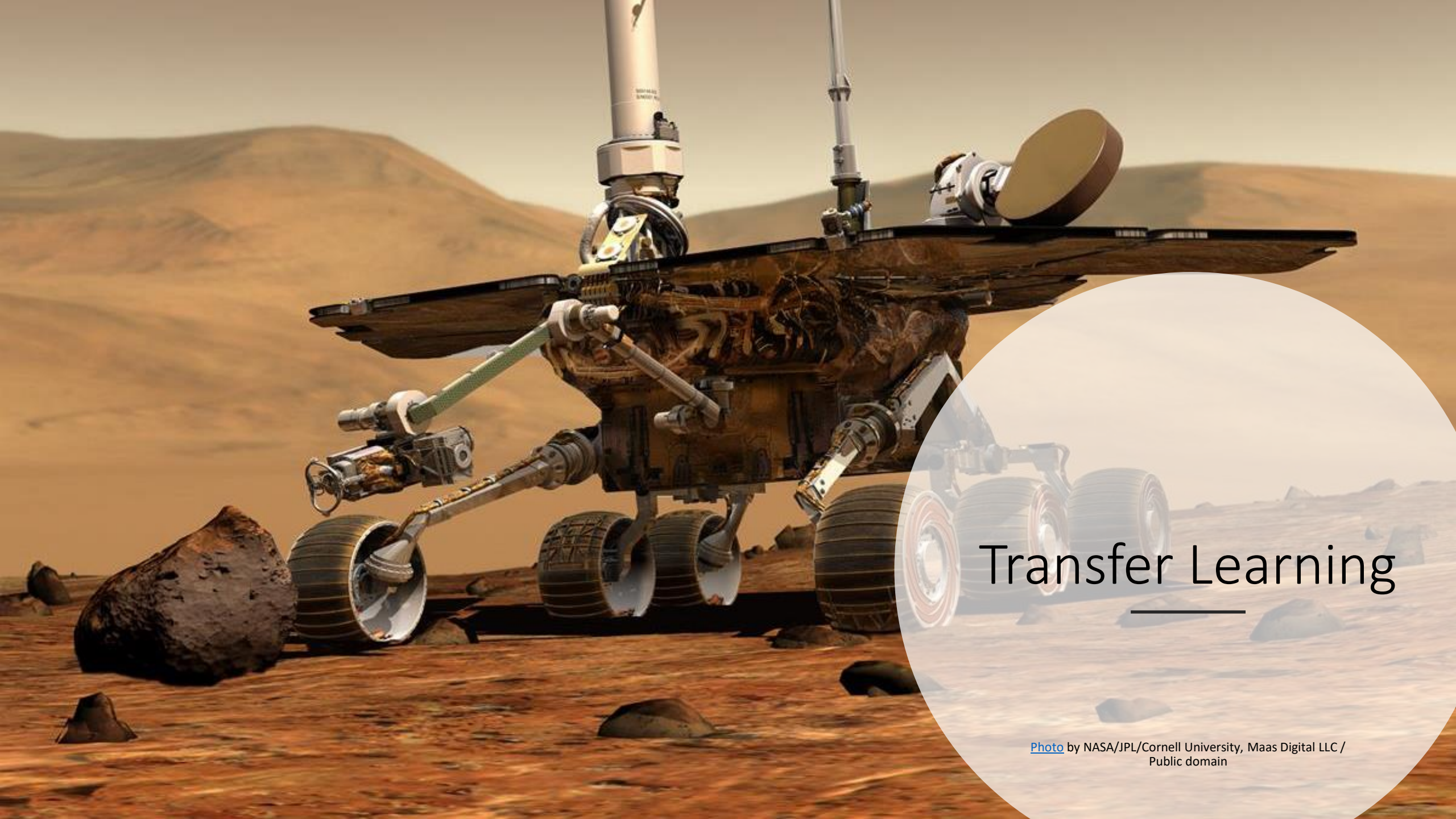
- Kera Image Preprocessing Image Data Generator
- 111_Data_Augmentation.ipynb
- rotation_range
- width_shirt_range
- height_shift_range
- shear_range
- zoom_range
- horizontal_flip
- vertical_flip





Data Augmentation

Show randomly generated examples



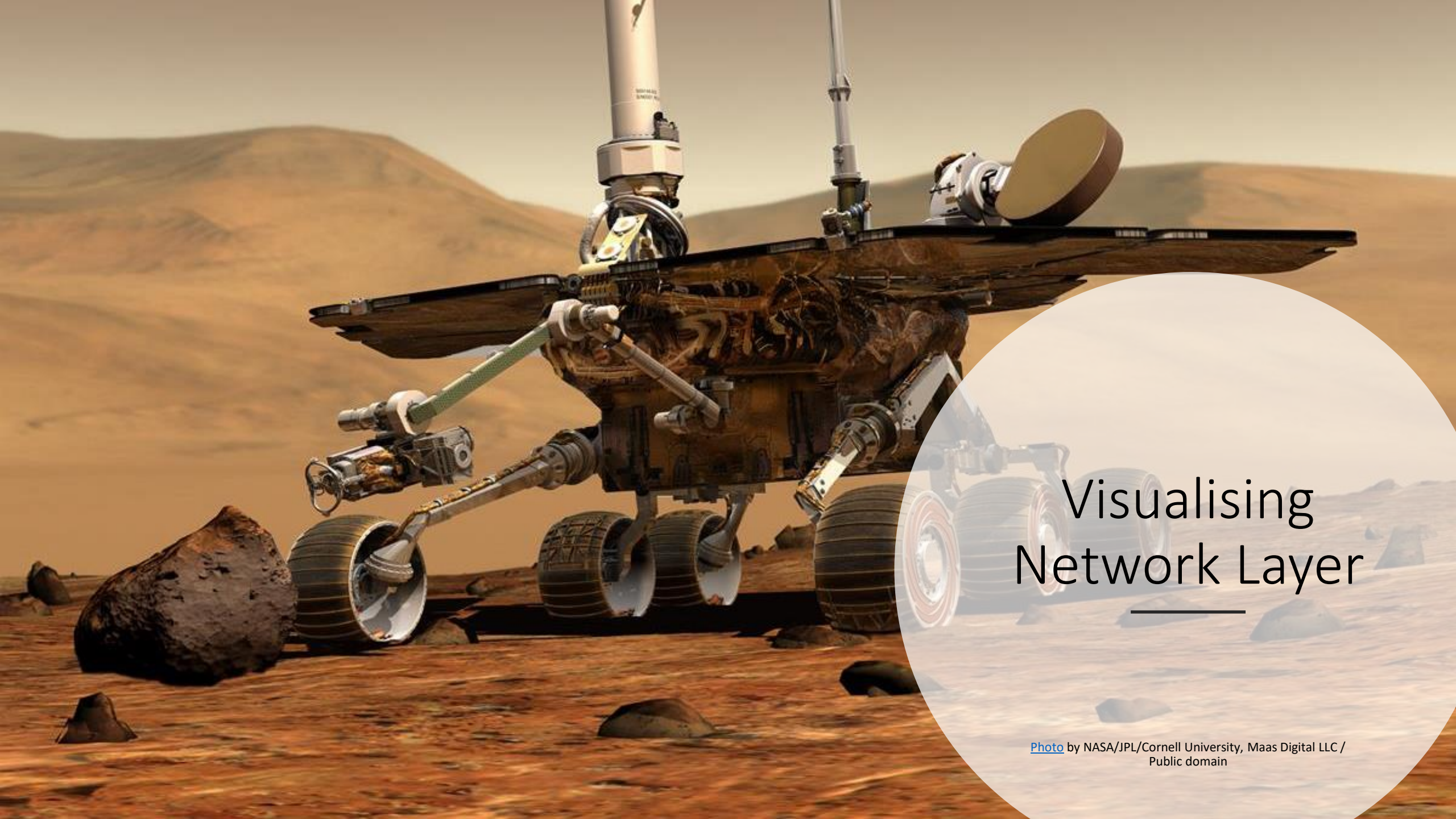
Transfer Learning

Photo by NASA/JPL/Cornell University, Maas Digital LLC / Public domain

Transfer Learning

- cats_and_dogs_transfer_learning.ipynb

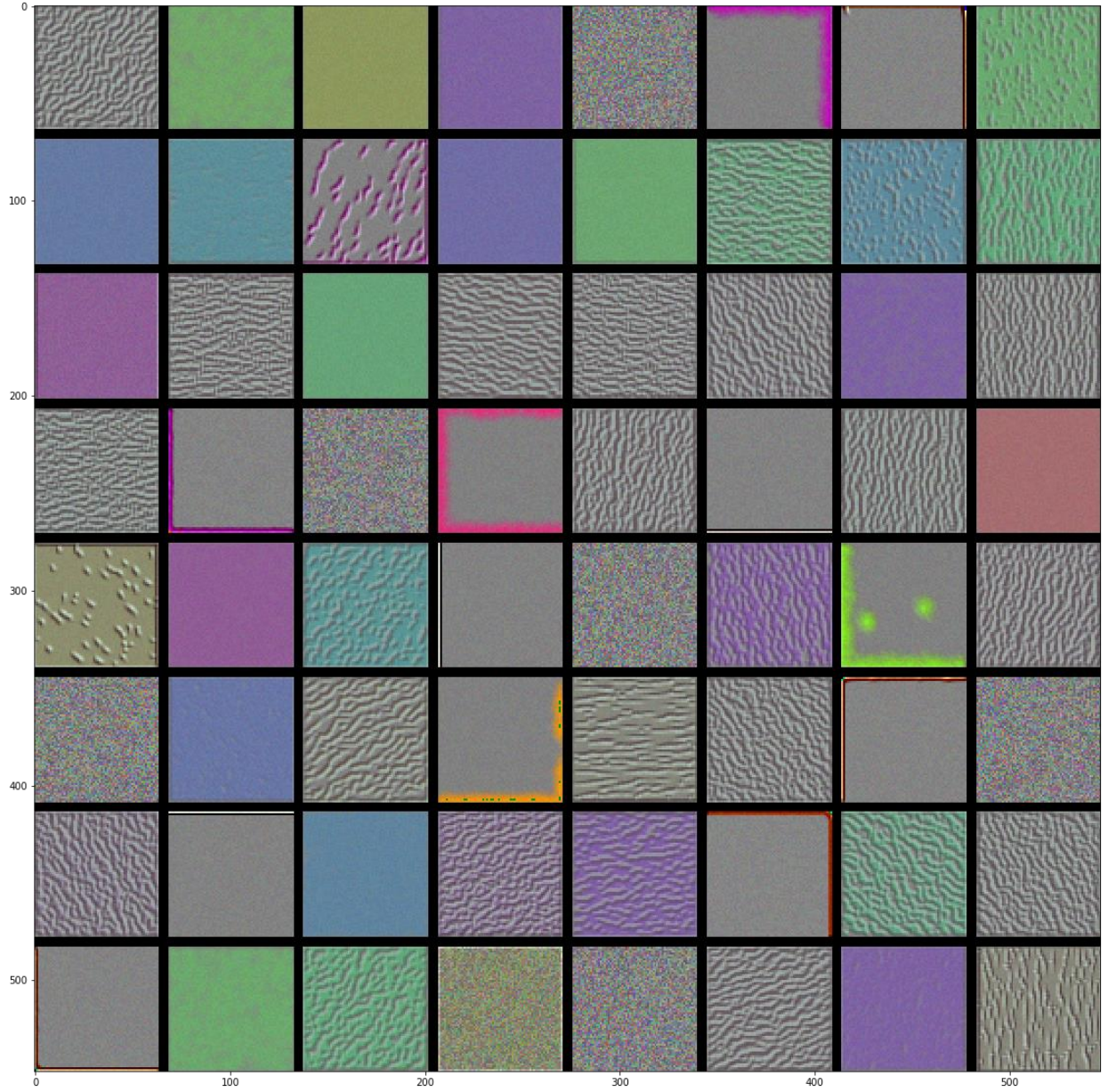

Feature Extraction
121 Feature Extraction.ipynb



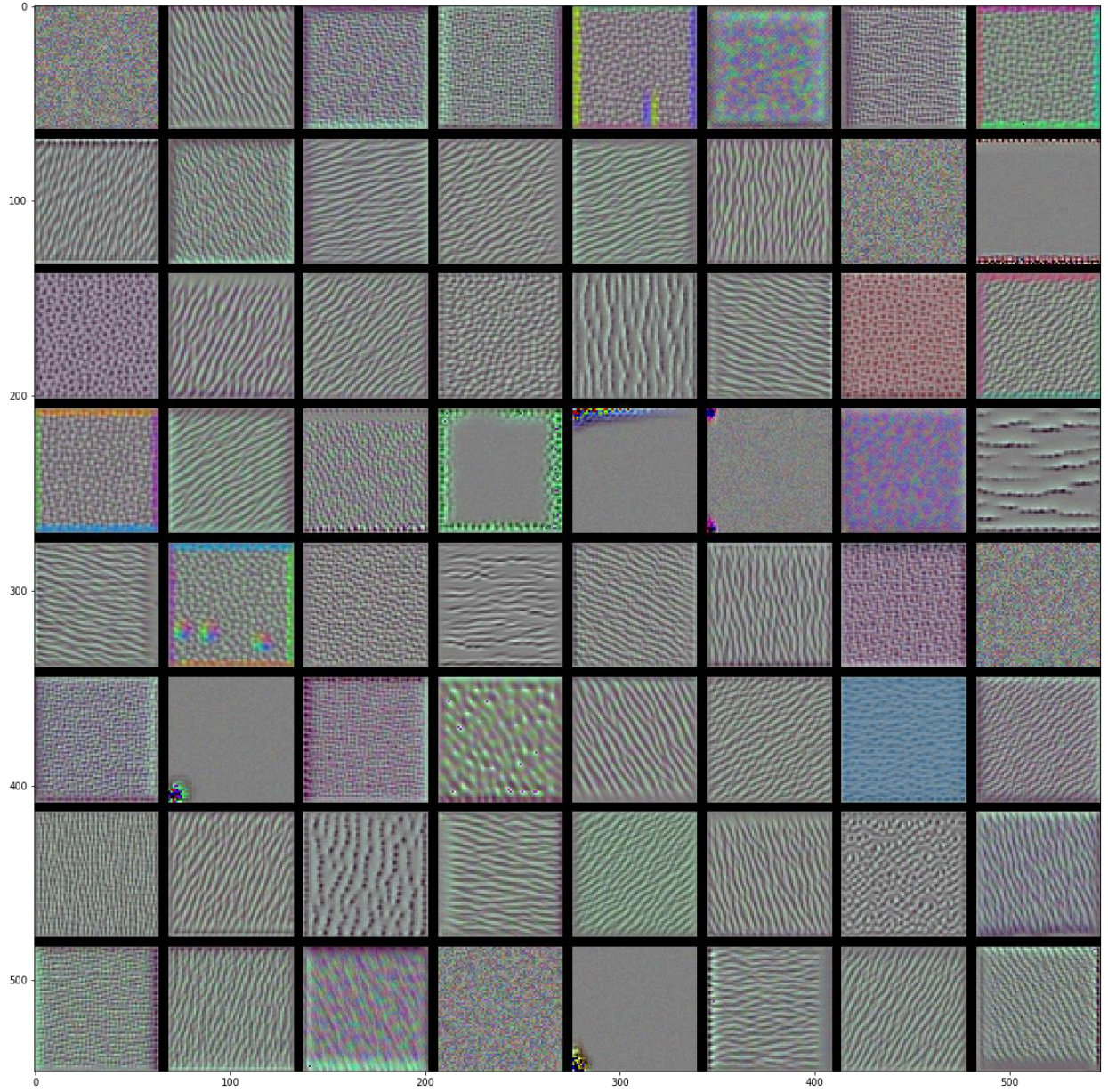
Visualising Network Layer

Photo by NASA/JPL/Cornell University, Maas Digital LLC / Public domain

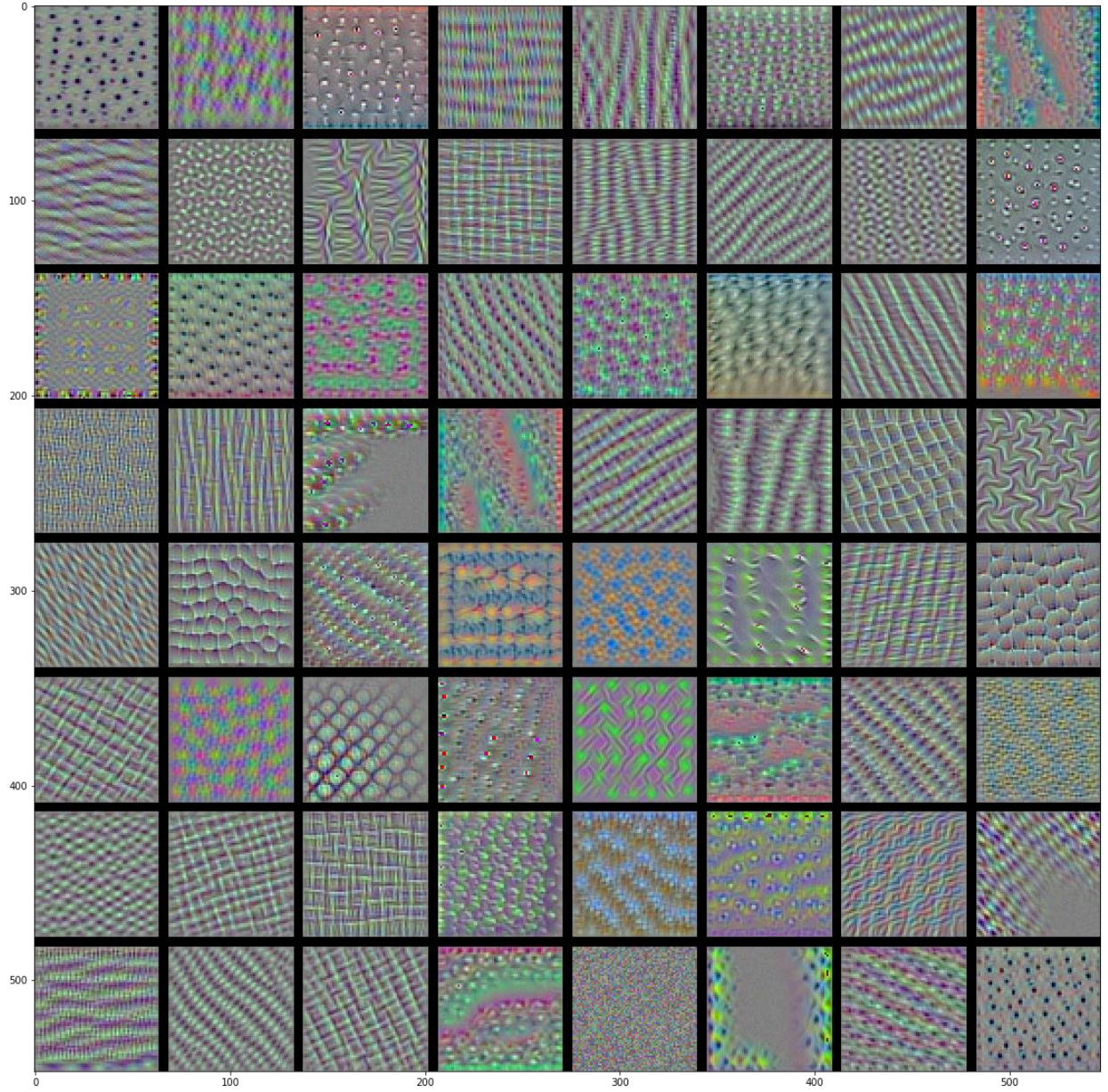
Visualising Network Layer



Visualising Network Layer

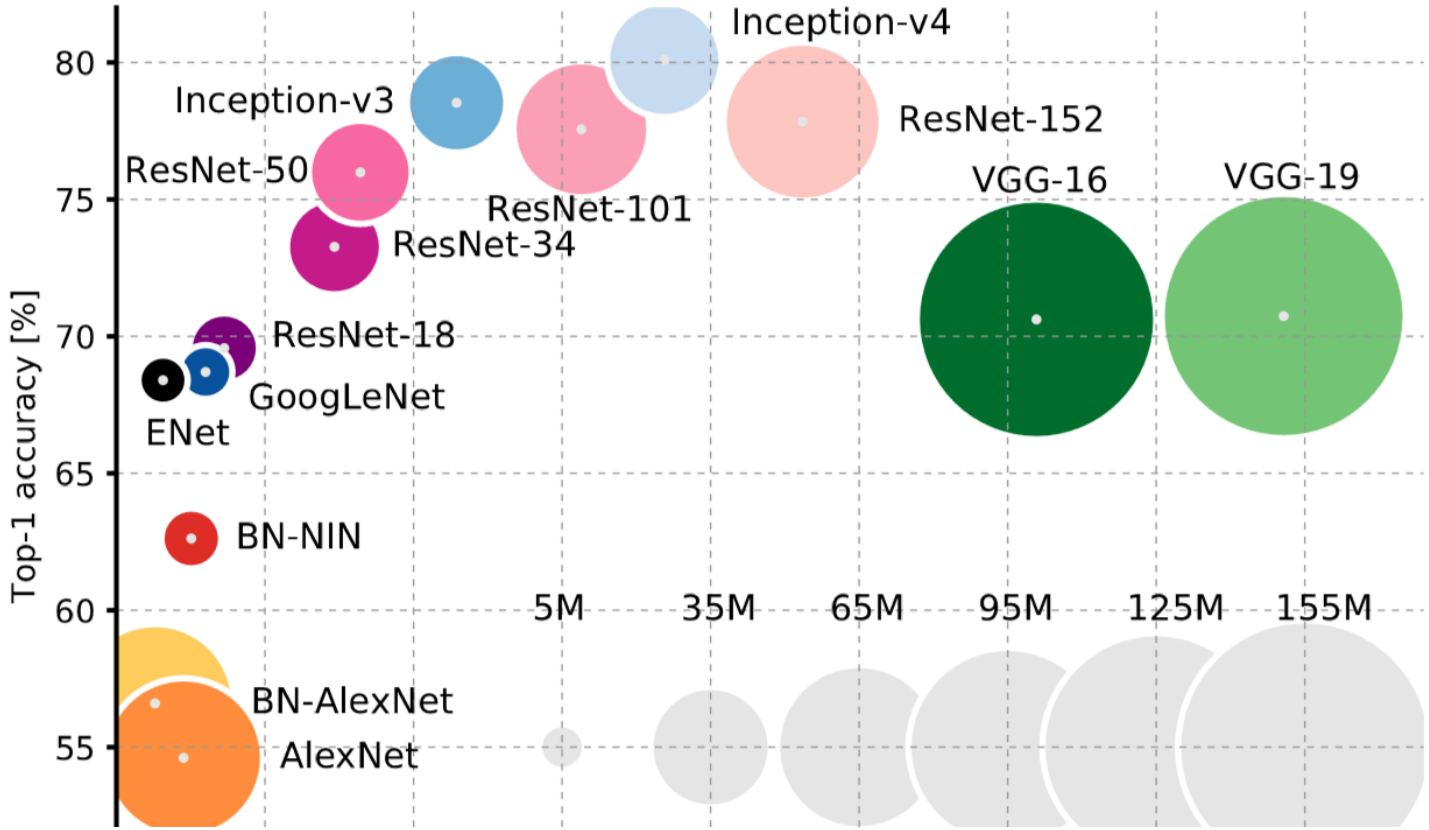


Visualising Network Layer



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

- H20.ai
- Deep Cognition
- Rapidminer
- KNIME
- ALTERYX
- DataRobot

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