TensorFlow

Differentiable Graphics with TensorFlow 2.0

Paige Bailey, Sofien Bouaziz,
Shan Carter, Josh Gordon,
Christian Häne, Alexander Mordvintsev,
Julien Valentin, Martin Wicke

Speakers

Sofien
Bouaziz

Josh
Gordon

Shan
Carter

Julien
Valentin

Christian
Häne

Speakers

Sofien
Bouaziz

Josh
Gordon

Shan
Carter

Julien
Valentin

Christian
Häne

Speakers

Sofien
Bouaziz

Josh
Gordon

Shan
Carter

Julien
Valentin

Christian
Häne

Speakers


Sofien Bouaziz


Josh Gordon


Shan Carter


Julien Valentin

Christian Häne

Speakers

Sofien
Bouaziz

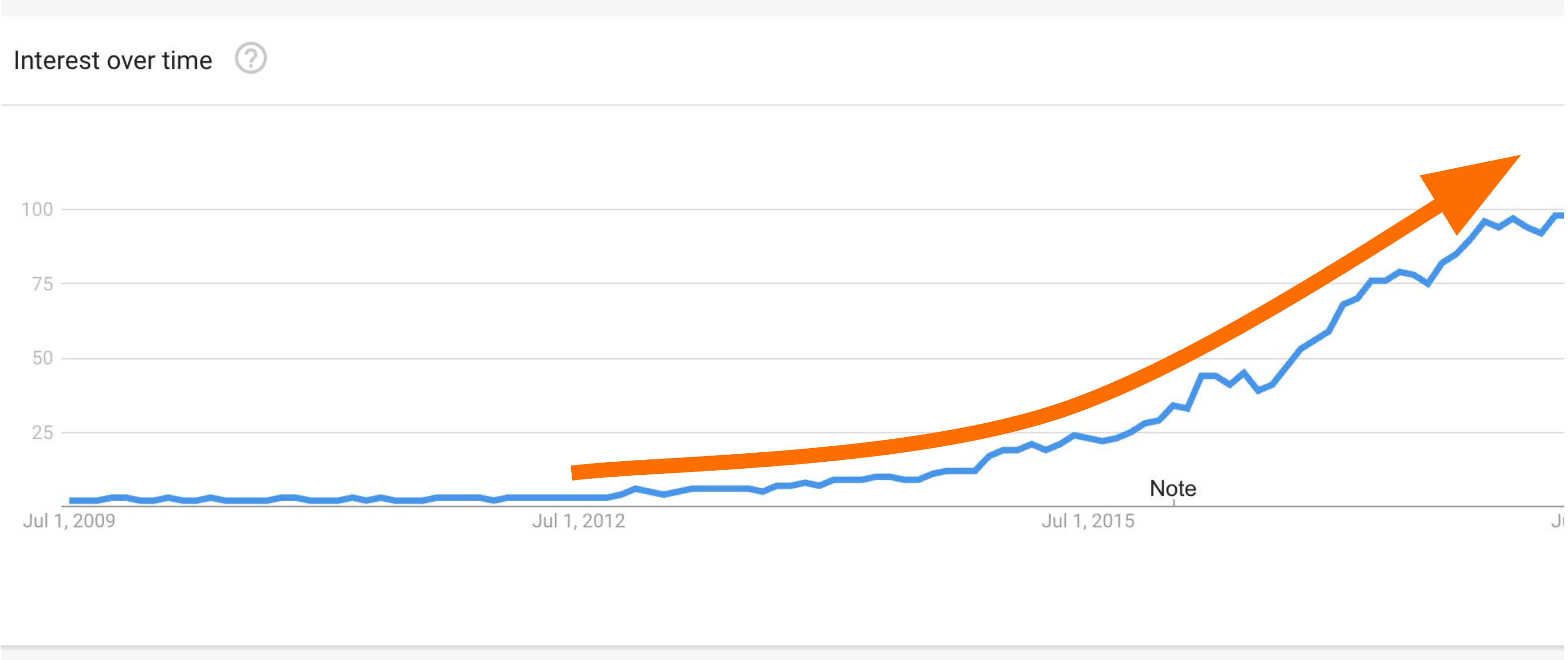
Josh
Gordon

Shan
Carter

Julien
Valentin

Christian
Häne

Deep Learning



TensorFlow

SIGGRAPH 2019

~35%

Deep Learning for Graphics

~~Deep Learning for Graphics~~

~~Deep Learning for Graphics~~

Graphics for Deep Learning

Agenda

1. Introduction to Deep Learning with TensorFlow 2.0 - *1h45mins*

Agenda

1. Introduction to Deep Learning with TensorFlow 2.0 - *1h45mins*
2. *Break - 10mins*

Agenda

1. Introduction to Deep Learning with TensorFlow 2.0 - *1h45mins*
2. *Break - 10mins*
3. Graphics Inspired Differentiable Layers - *45mins*

Agenda

1. Introduction to Deep Learning with TensorFlow 2.0 - *1h45mins*
2. *Break - 10mins*
3. Graphics Inspired Differentiable Layers - *45mins*
4. Visualization and Interpretation of Neural Networks - *25mins*

Agenda

1. Introduction to Deep Learning with TensorFlow 2.0 - *1h45mins*
2. *Break - 10mins*
3. Graphics Inspired Differentiable Layers - *45mins*
4. Visualization and Interpretation of Neural Networks - *25mins*
5. *Conclusion - 5mins*

TensorFlow

Intro to TensorFlow 2.0

Josh Gordon (@random_forests)

Topics 1 of 2

Background on TensorFlow 2.0

- For beginners and experts
- Defining models (sequential vs subclassing)
- Training models (built-in vs custom loops)

Getting started walkthroughs

- Linear regression from scratch
- MNIST Sequential
- MNIST Subclassing

Topics 2 of 2

Recommended advanced tutorials

- Deep Dream and Style Transfer
- DCGAN, Pix2Pix, CycleGan

Learning more

- Book recommendations

TensorFlow

Released by Google in 2015

- >1800 contributors worldwide

Version 2.0

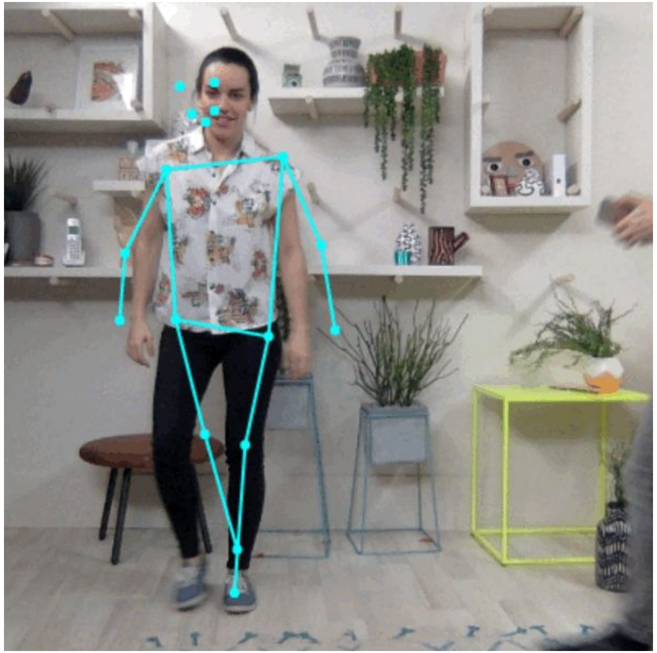
- Easier to use
- Currently in beta

tensorflow.org/beta

Beyond Python

tensorflow.org/js

tensorflow.org/swift



bit.ly/pose-net

Quick demo

PoseNet

http://bit.ly/pose-net

You can use TF 2.0 like NumPy

```
import tensorflow as tf # Assuming TF 2.0 is installed

a = tf.constant([[1, 2],[3, 4]])
b = tf.matmul(a, a)

print(b)
# tf.Tensor( [[ 7 10] [15 22]], shape=(2, 2), dtype=int32)

print(type(b.numpy()))
# <class 'numpy.ndarray'>
```

Sequential models

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),

  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

TF 1.x

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

TF 2.0

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Subclassed models

```
class MyModel(tf.keras.Model):
  def __init__(self, num_classes=10):
    super(MyModel, self).__init__(name='my_model')
    self.dense_1 = layers.Dense(32, activation='relu')
    self.dense_2 = layers.Dense(num_classes,activation='sigmoid')
  def call(self, inputs):
    # Define your forward pass here,
    x = self.dense_1(inputs)
    return self.dense_2(x)
```

Use a built-in training loop...

```
model.fit(x_train, y_train, epochs=5)
```

Or define your own

```
model = MyModel()

with tf.GradientTape() as tape:
  logits = model(images)
  loss_value = loss(logits, labels)

grads = tape.gradient(loss_value, model.trainable_variables)
optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

What’s the difference?

Sequential, Functional

- Your model is a stack of layers (Sequential) or DAG (Functional)
- Any graph you compile will run
- **Catch errors at compile time**

Subclassing

- Your model is Python bytecode
- Hackable and flexible
- **Run-time errors / harder to maintain**

Walkthrough 1

Linear regression in TensorFlow 2.0

tensorflow.org/beta/tutorials/eager/custom_training

Walkthrough 2

MNIST with a sequential model and built-in training loop

tensorflow.org/beta/tutorials/quickstart/beginner

Walkthrough 3

MNIST with a subclassed model and custom training loop

tensorflow.org/beta/tutorials/quickstart/advanced

Convolution

Not a Deep Learning concept

```python
import scipy
from skimage import color, data
import matplotlib.pyplot as plt
img = data.astronaut()
img = color.rgb2gray(img)
plt.axis('off')
plt.imshow(img, cmap=plt.cm.gray)
```

Convolution example

Does anyone know who this is?

-1	-1	-1
-1	8	-1
-1	-1	-1

Notes

Edge detection intuition: dot product of the filter with a region of the image will be zero if all the pixels around the border have the same value as the center.

Convolution example

Eileen Collins

-1	-1	-1
-1	8	-1
-1	-1	-1

Notes
Edge detection intuition: dot product of the filter with a region of the image will be zero if all the pixels around the border have the same value as the center.

A simple edge detector

```
kernel = np.array([[-1,-1,-1],
                   [-1,8,-1],
                   [-1,-1,-1]])
result = scipy.signal.convolve2d(img, kernel, 'same')
plt.axis('off')
plt.imshow(result, cmap=plt.cm.gray)
```

Easier to see with seismic

Eileen Collins

-1	-1	-1
-1	8	-1
-1	-1	-1

Notes
Edge detection intuition: dot product of the filter with a region of the image will be zero if all the pixels around the border have the same value as the center.

Example

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

An input image
(no padding)

1	0	1
0	0	0
0	1	0

A filter
(3x3)

Output image
(after convolving with stride 1)

Example

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

An input image
(no padding)

1	0	1
0	0	0
0	1	0

A filter
(3x3)

3	

Output image
(after convolving with stride 1)

2*1 + 0*0 + 1*1 + 0*0 + 1*0 + 0*0 + 0*0 + 0*1 + 1*0

Example

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

An input image
(no padding)

1	0	1
0	0	0
0	1	0

A filter
(3x3)

3	2

Output image
(after convolving with stride 1)

Example

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

An input image
(no padding)

1	0	1
0	0	0
0	1	0

A filter
(3x3)

3	2
3	

Output image
(after convolving with stride 1)

Example

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

An input image
(no padding)

1	0	1
0	0	0
0	1	0

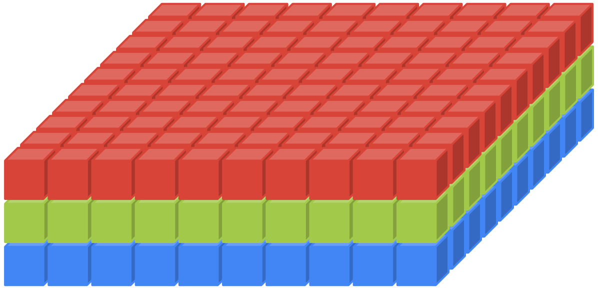
A filter
(3x3)

3	2
3	1

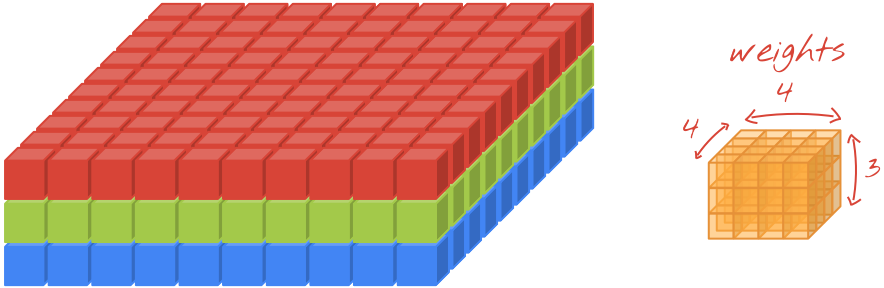
Output image
(after convolving with stride 1)

In 3d

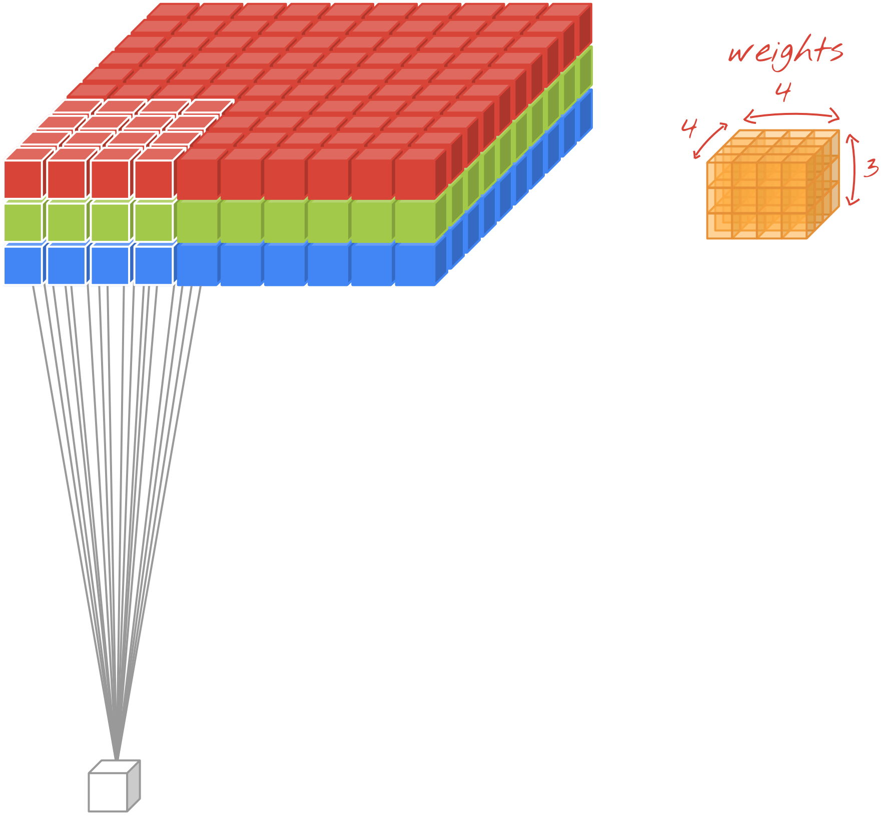
```
model = Sequential()

model.add(Conv2D(filters=4,
                 kernel_size=(4,4),
                 input_shape=(10,10,3)))
```

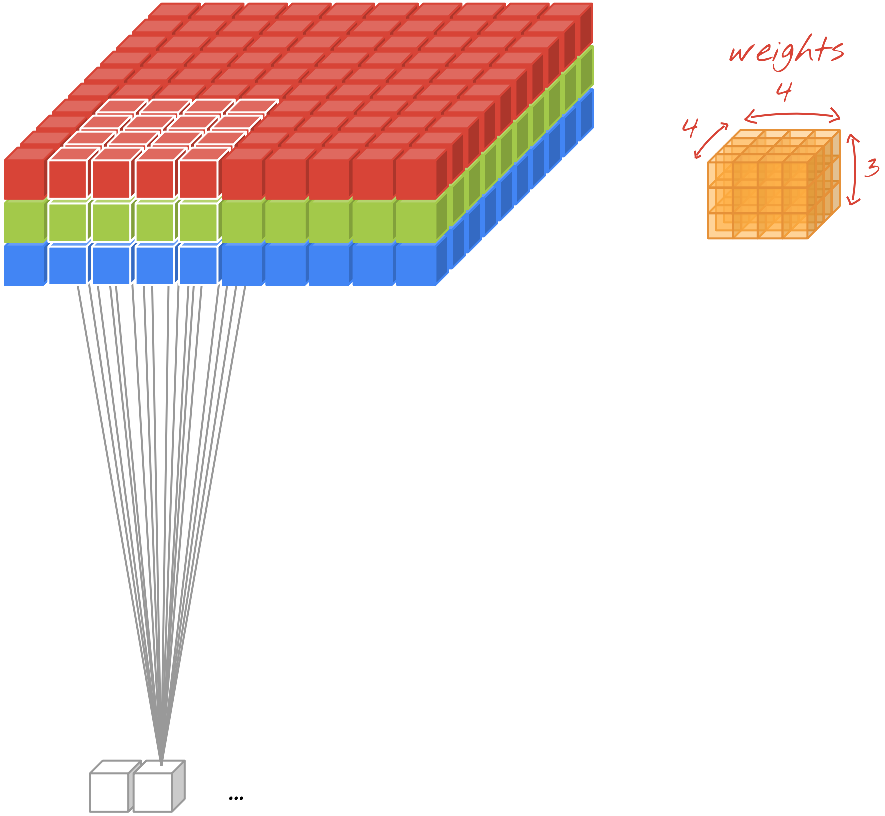


A RGB image as a 3d **volume**.
Each color (or channel) is a layer.

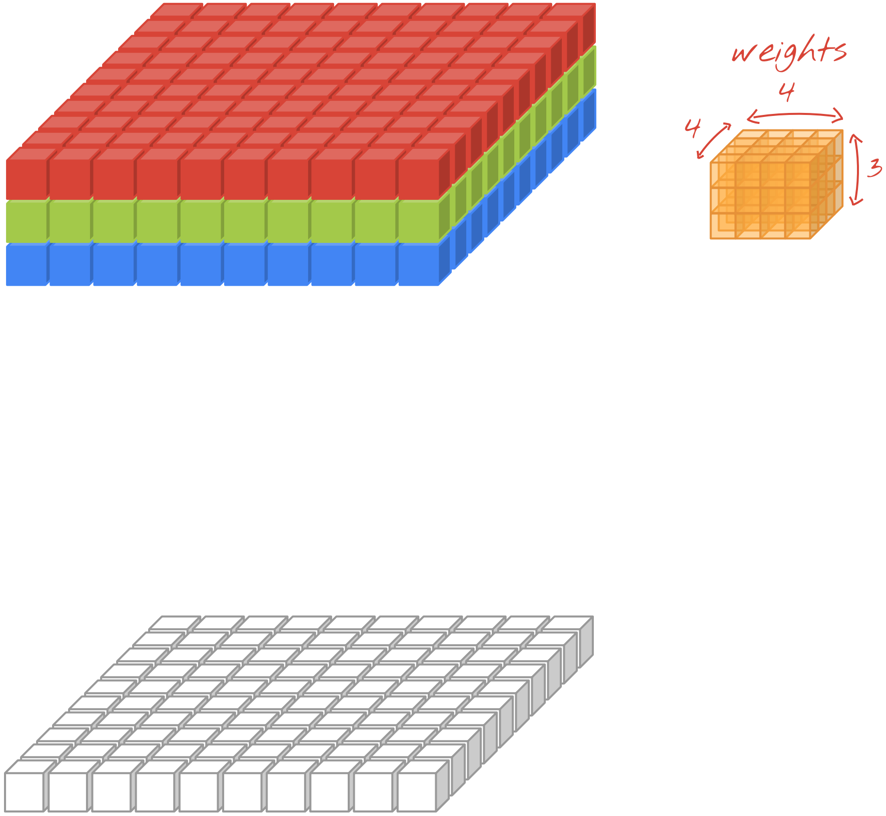


In 3d, our filters have width, height, and depth.

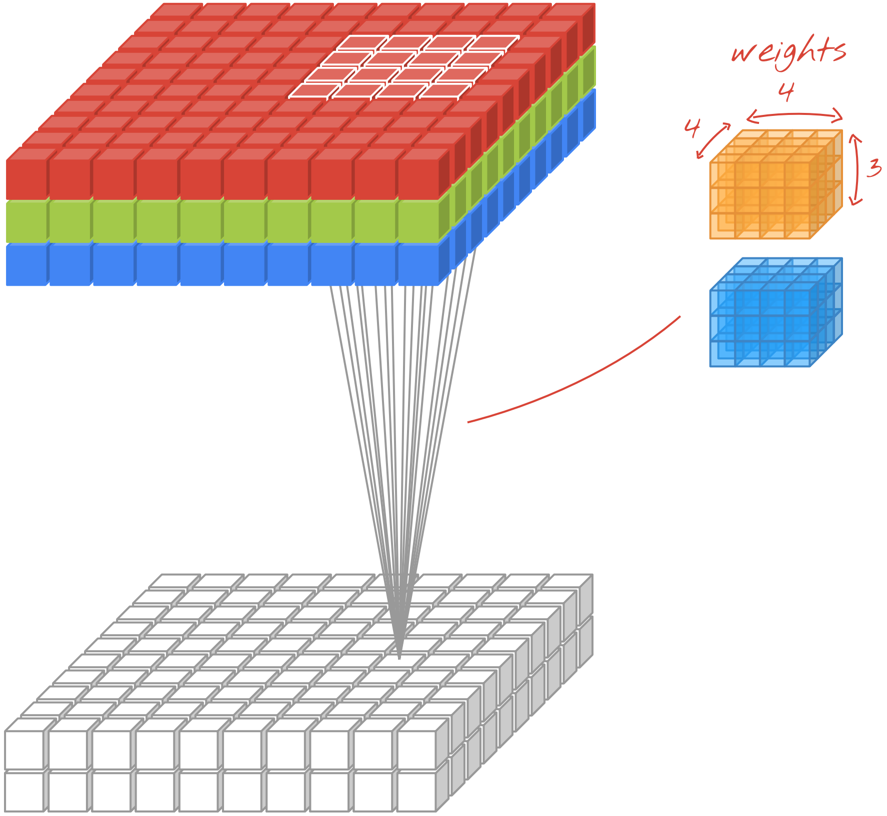

weights
4
4
3



Applied in the same way as 2d (sum of weight * pixel value as they slide across the image).



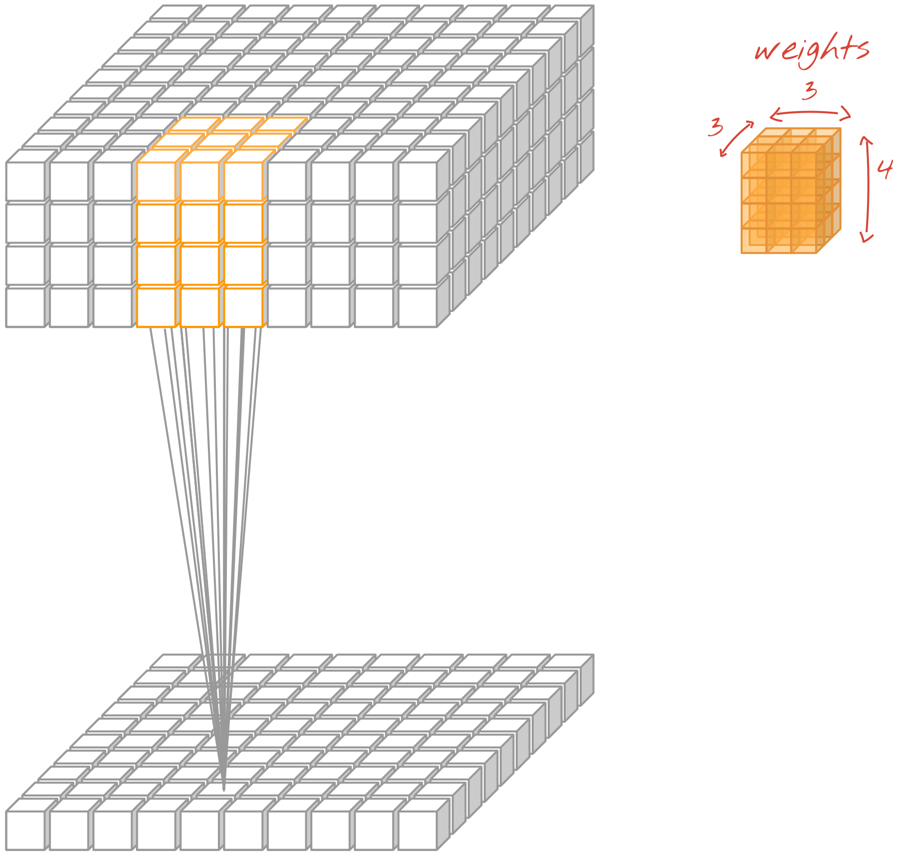
Applying the convolution over the rest of the input image.

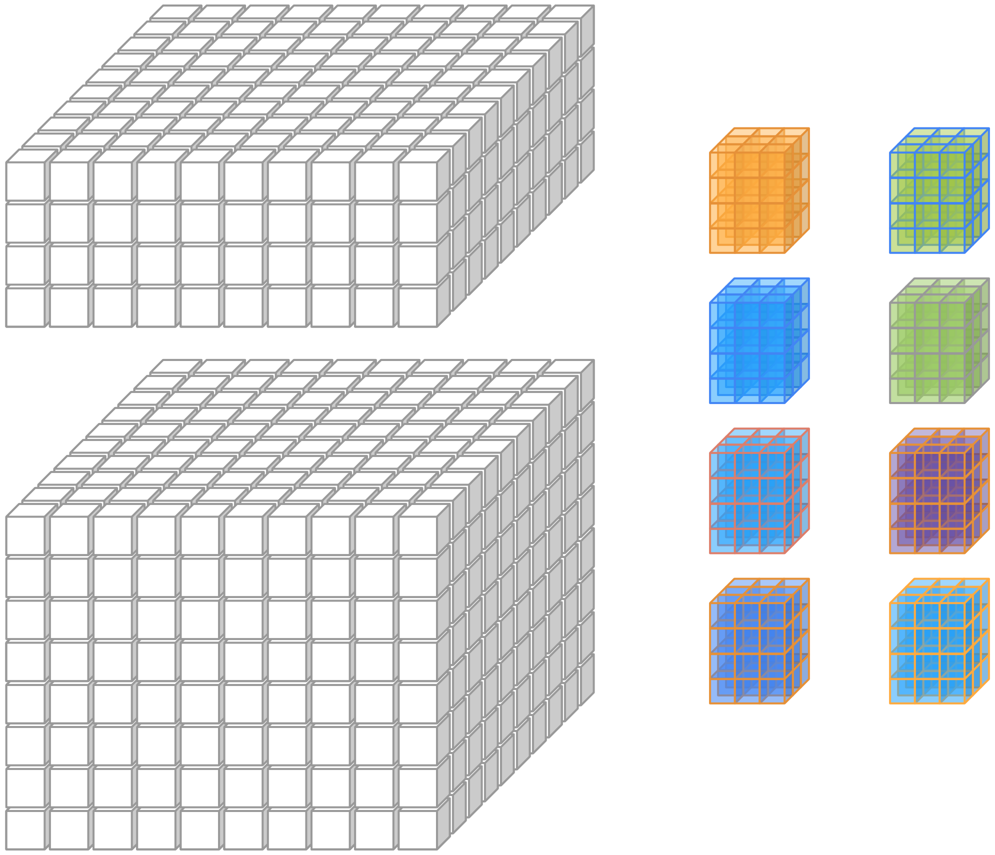


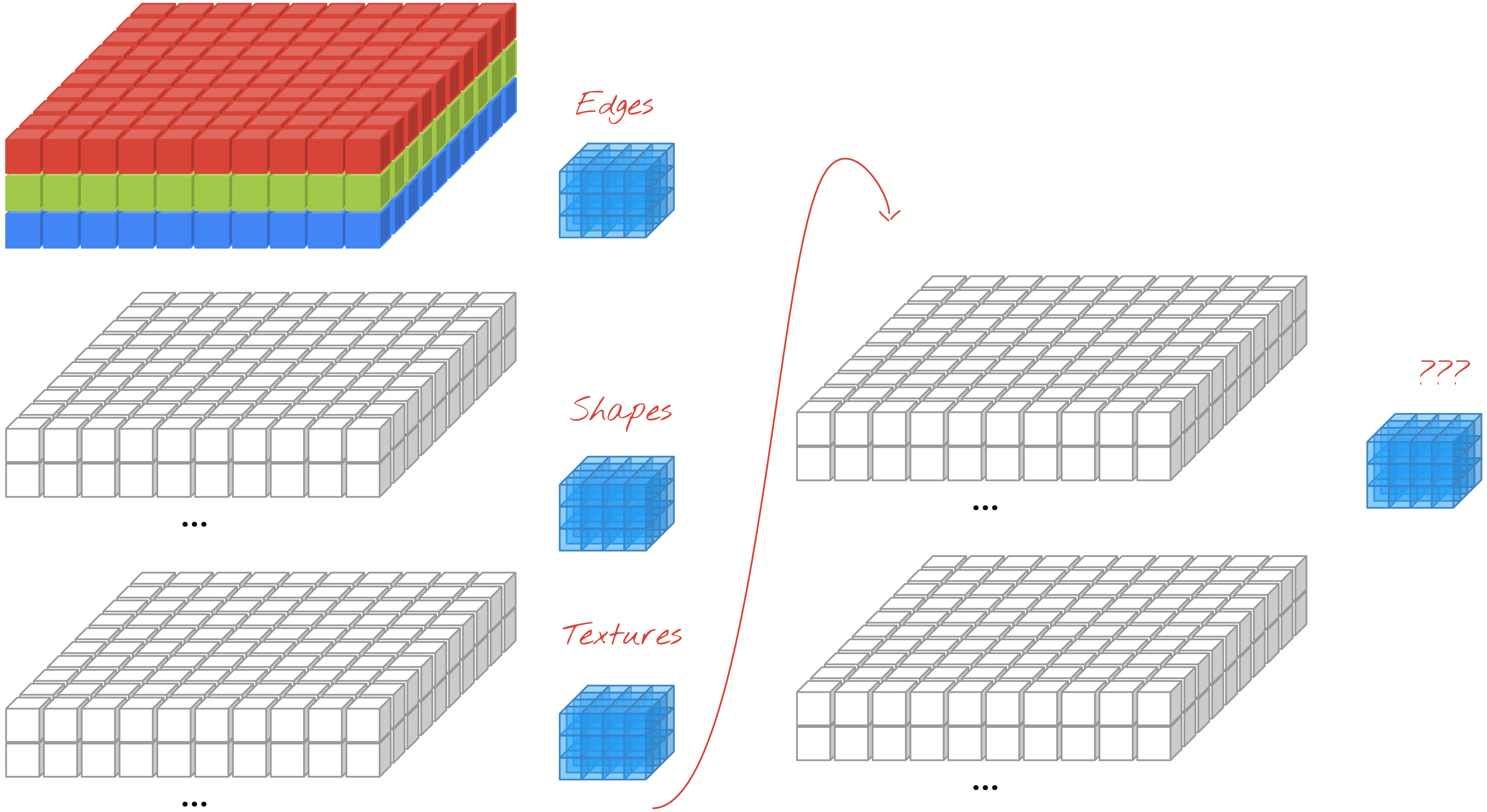
More filters, more output channels.

Going deeper

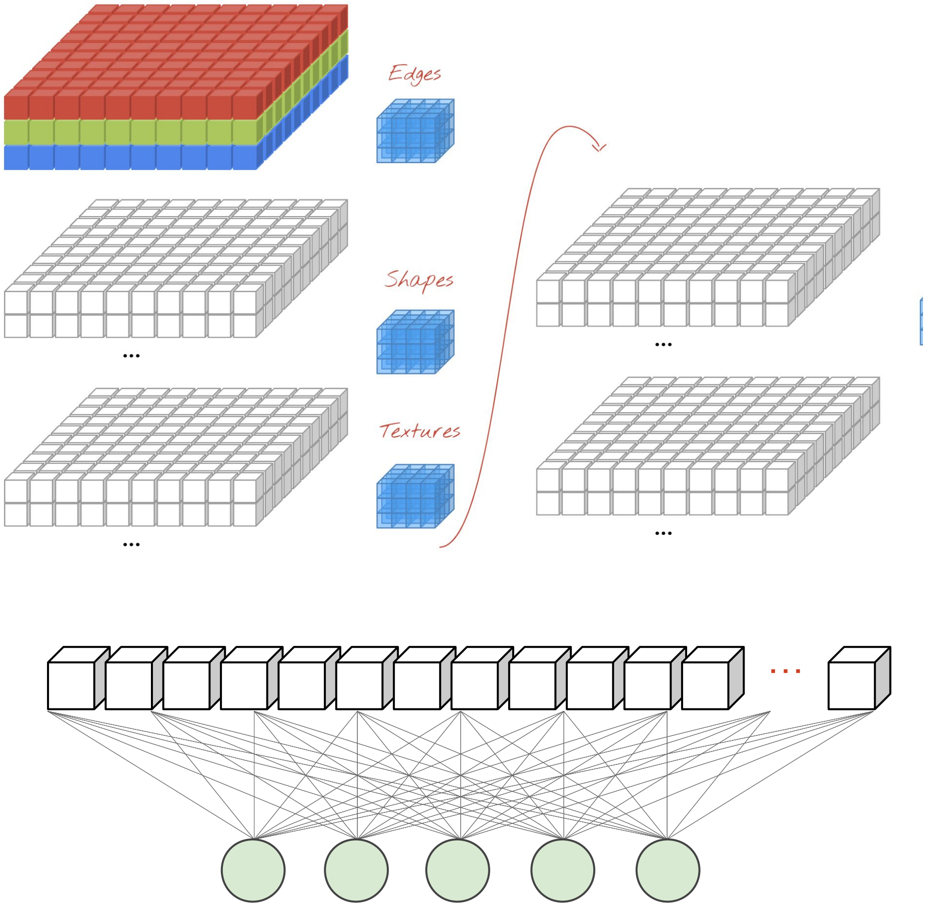
```
model = Sequential()

model.add(Conv2D(filters=4,
                 kernel_size=(4,4),
                 input_shape=(10,10,3)))

model.add(Conv2D(filters=8,
                 kernel_size=(3,3)))
```


weights
3
3
4





Edges
Shapes
Textures
???



Edges
Shapes
Textures

Walkthrough 4

A simple CNN

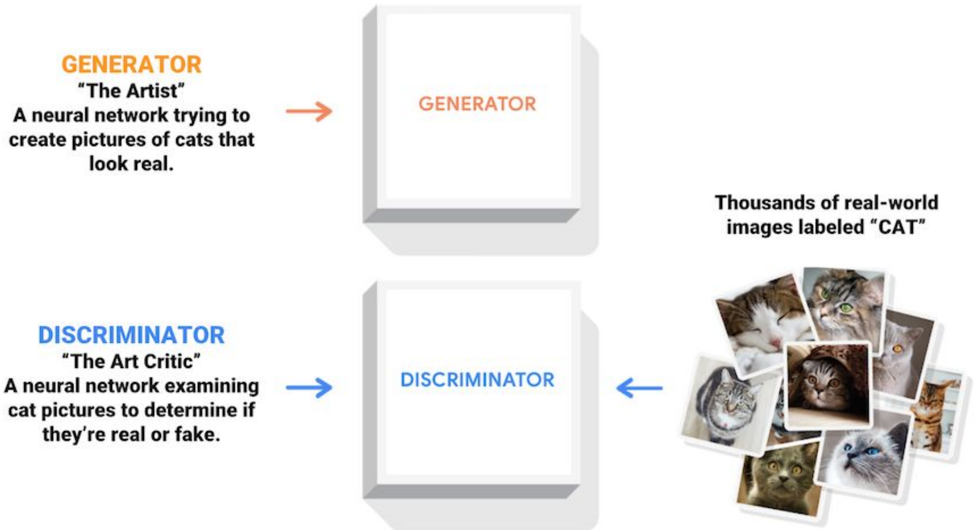
tensorflow.org/beta/tutorials/images/intro_
to_cnns

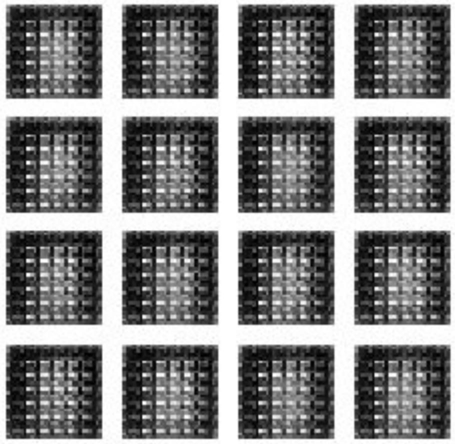
Walkthrough 5

Deep Dream

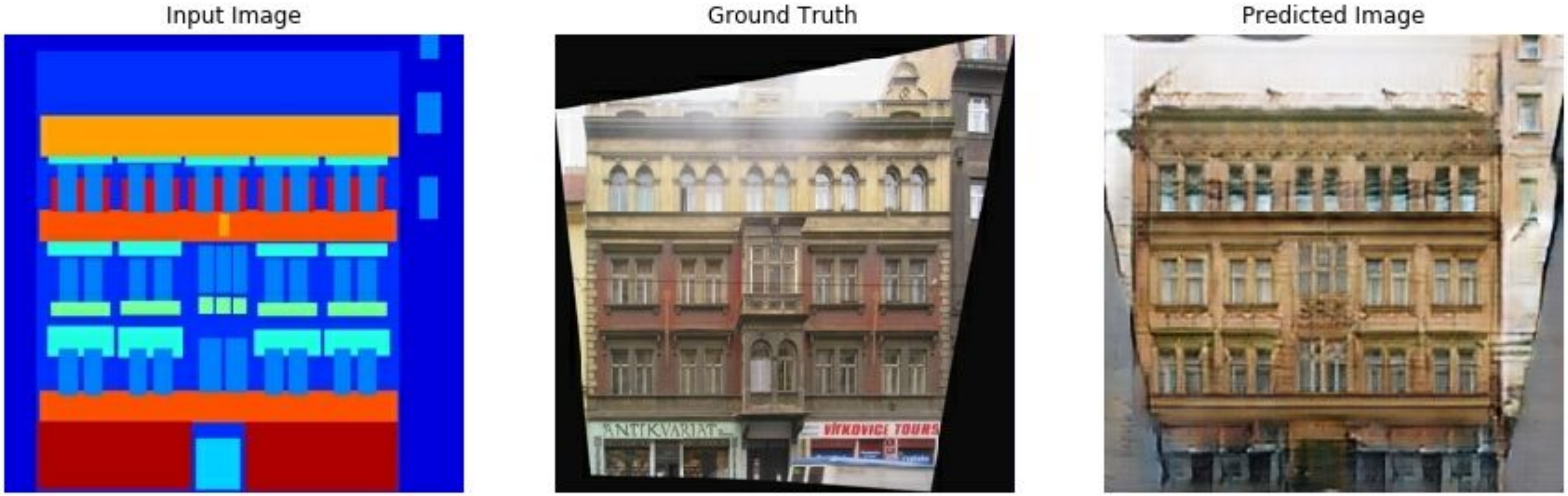
tensorflow.org/beta/tutorials/generative/deepdream

Recommended tutorials

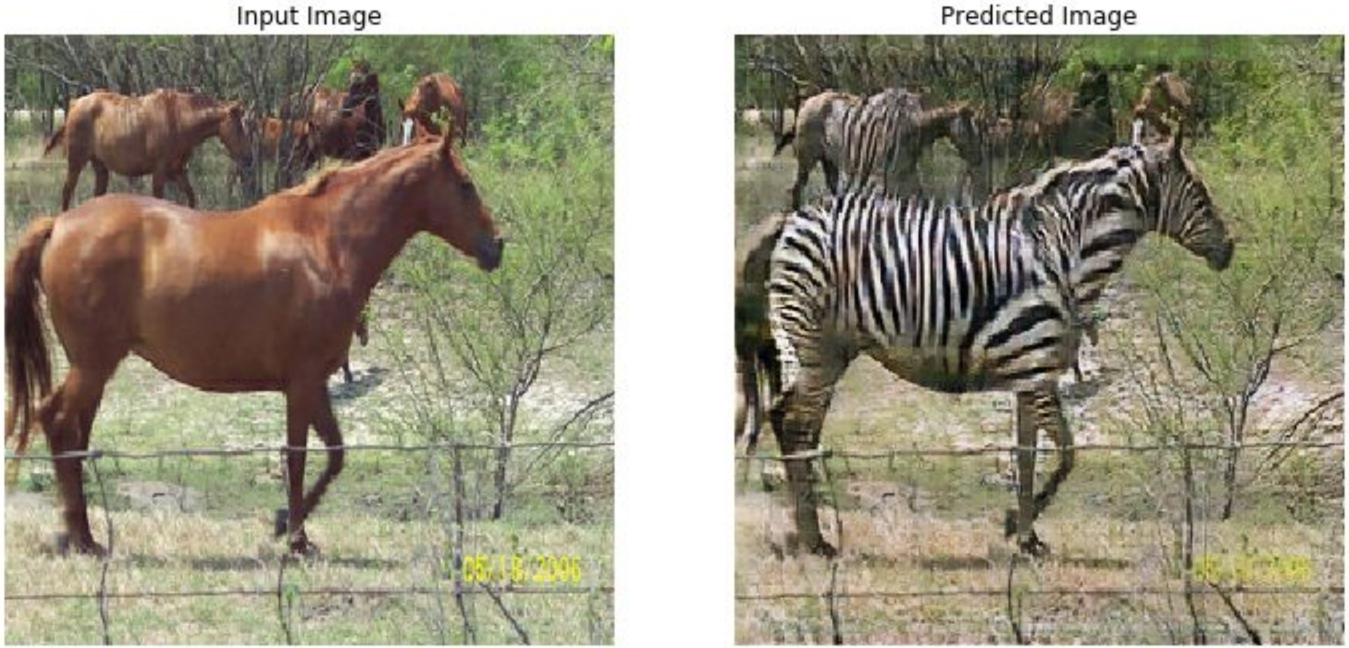




https://www.tensorflow.org/beta/tutorials/generative/dcgan



tensorflow.org/beta/tutorials/generative/pix2pix



tensorflow.org/beta/tutorials/generative/cyclegan

Quick demo

Embedding projector

projector.tensorflow.org

Under the hood

Let's make this faster

```
lstm_cell = tf.keras.layers.LSTMCell(10)



def fn(input, state):
  return lstm_cell(input, state)

input = tf.zeros([10, 10]); state = [tf.zeros([10, 10])] * 2
lstm_cell(input, state); fn(input, state) # warm up

# benchmark
timeit.timeit(lambda: lstm_cell(input, state), number=10) # 0.03
```

Let's make this faster

```
lstm_cell = tf.keras.layers.LSTMCell(10)

@tf.function
def fn(input, state):
  return lstm_cell(input, state)

input = tf.zeros([10, 10]); state = [tf.zeros([10, 10])] * 2
lstm_cell(input, state); fn(input, state) # warm up

# benchmark
timeit.timeit(lambda: lstm_cell(input, state), number=10) # 0.03
timeit.timeit(lambda: fn(input, state), number=10) # 0.004
```

AutoGraph makes this possible

```python
@tf.function
def f(x):
  while tf.reduce_sum(x) > 1:
    x = tf.tanh(x)
  return x

# you never need to run this (unless curious)
print(tf.autograph.to_code(f))
```

Generated code

```
def tf__f(x):
  def loop_test(x_1):
    with ag__.function_scope('loop_test'):
      return ag__.gt(tf.reduce_sum(x_1), 1)
  def loop_body(x_1):
    with ag__.function_scope('loop_body'):
      with ag__.utils.control_dependency_on_returns(tf.print(x_1)):
        tf_1, x = ag__.utils.alias_tensors(tf, x_1)
        x = tf_1.tanh(x)
        return x,
  x = ag__.while_stmt(loop_test, loop_body, (x,), (tf,))
  return x
```

Going big: tf.distribute.Strategy

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(64, input_shape=[10]),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')])

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Going big: Multi-GPU

```
strategy = tf.distribute.MirroredStrategy()

with strategy.scope():
  model = tf.keras.models.Sequential([
      tf.keras.layers.Dense(64, input_shape=[10]),
      tf.keras.layers.Dense(64, activation='relu'),
      tf.keras.layers.Dense(10, activation='softmax')])
  model.compile(optimizer='adam', loss='categorical_crossentropy',
                metrics=['accuracy'])
```

Learning more

Tutorials for TF2

- tensorflow.org/beta

Books

- Deep Learning with Python
- Hands-on ML with Scikit-Learn, Keras and TensorFlow **(2nd edition)**

jbgordon@google.com

Q&A

Josh Gordon (@random_forests)

BREAK

The course will resume at 10:55

TensorFlow

Graphics Inspired Differentiable Layers

Julien Valentin (@JPCValentin)

Christian Häne

What will be covered

1. TensorFlow Graphics: Computer Graphics meets Deep Learning
2. 3D and Graphics Layers in The State of The Art

TensorFlow

TensorFlow Graphics: Computer Graphics meets DL

Julien Valentin (@JPCValentin)

State of the art in generative modelling

Supervised

- Granular interpretability
  - e.g. make someone smile
- Data is expensive to acquire, both in time and money

State of the art in generative modelling

Supervised

- Granular interpretability
  - e.g. make someone smile
- Data is expensive to acquire, both in time and money

Self supervised

- GANs
  - Photo-realistic
  - Pixel synthesis - 2D
- Auto-encoders / VAE
  - Smooth reconstructions

- Lack of granular interpretability in the latent space - disentanglement & precise control is hard
- Lots of data, cheap to acquire

Domain knowledge / constraints

- Our world is inherently 3D

Domain knowledge / constraints

- Our world is inherently 3D
- Computer graphics is a very mature field and capable of rendering 3D scenes in a photorealistic manner

Domain knowledge / constraints

- Our world is inherently 3D
- Computer graphics is a very mature field and capable of rendering 3D scenes in a photorealistic manner
- A large amount of building blocks from this field are actually differentiable!
    - Reflectance functions (e.g. Phong / Lambertian)
    - Camera projection functions
    - Environment maps (e.g. spherical harmonics)

Domain knowledge / constraints

- Baking graphics constraints can bring
  - Increased interpretability / ease of manipulation of well understood quantities
    - Angle of rotation
    - Intensity of light

Domain knowledge / constraints

- Baking graphics constraints can bring
    - Increased interpretability / ease of manipulation of well understood quantities
        - Angle of rotation
        - Intensity of light
    - Decrease in the number of parameters in networks
        - Faster training / inference speed

Domain knowledge / constraints

- Baking graphics constraints can bring
  - Increased interpretability / ease of manipulation of well understood quantities
    - Angle of rotation
    - Intensity of light
  - Decrease in the number of parameters in networks
    - Faster training / inference speed
  - Large decrease in annotation needs / large increase in the number of training samples
    - More cost efficient / higher precision / less overfitting

Evidence in the literature

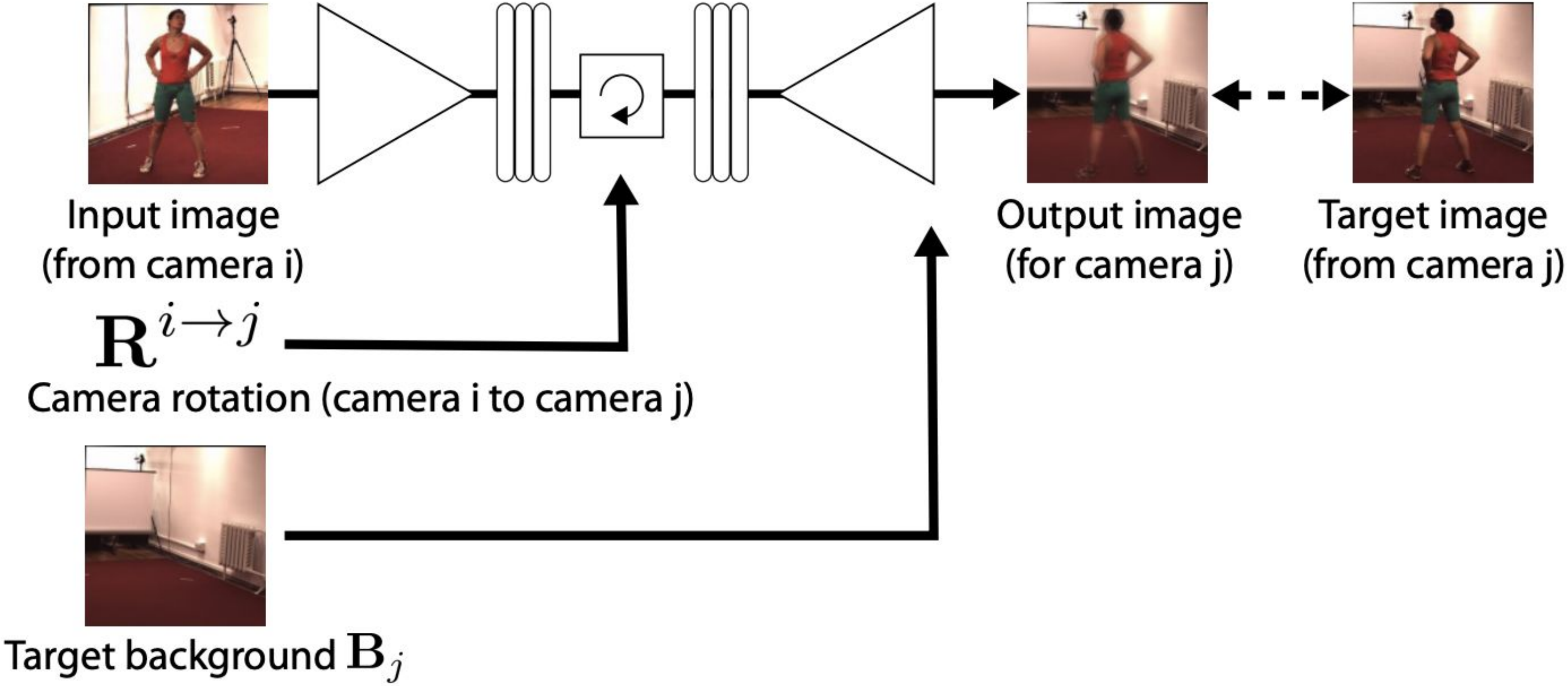
Unsupervised Geometry-Aware Representation for 3D Human Pose Estimation

Helge Rhodin, Mathieu Salzmann, and Pascal Fua

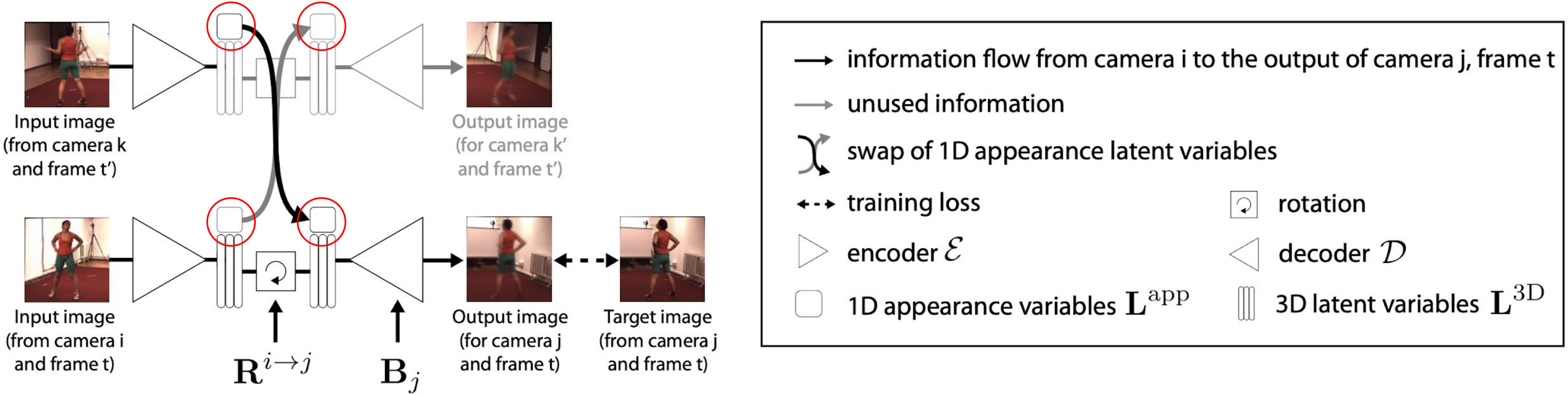
CVLab, EPFL, Lausanne, Switzerland

ECCV 2018

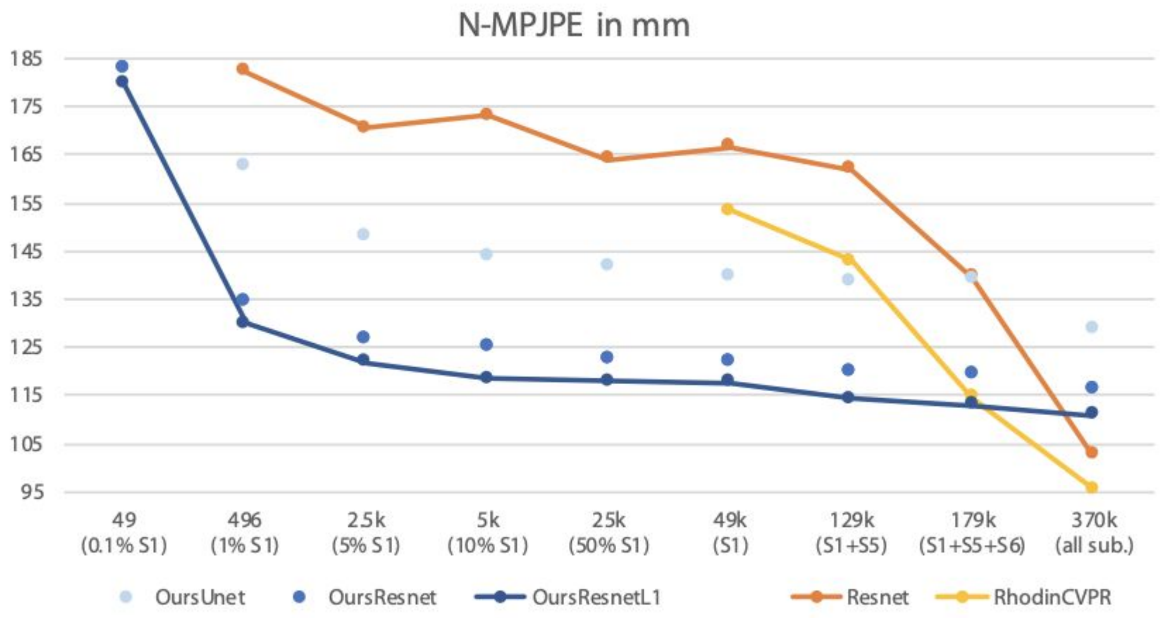
Unsupervised Geometry-Aware Representation for 3D Human Pose Estimation



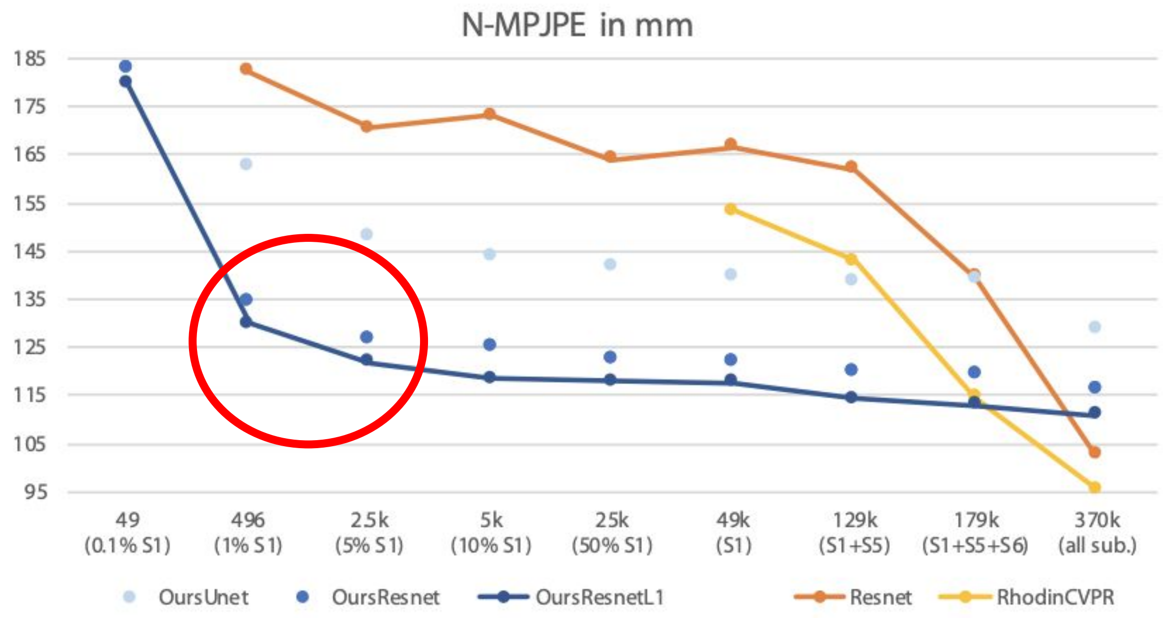
Unsupervised Geometry-Aware Representation for 3D Human Pose Estimation



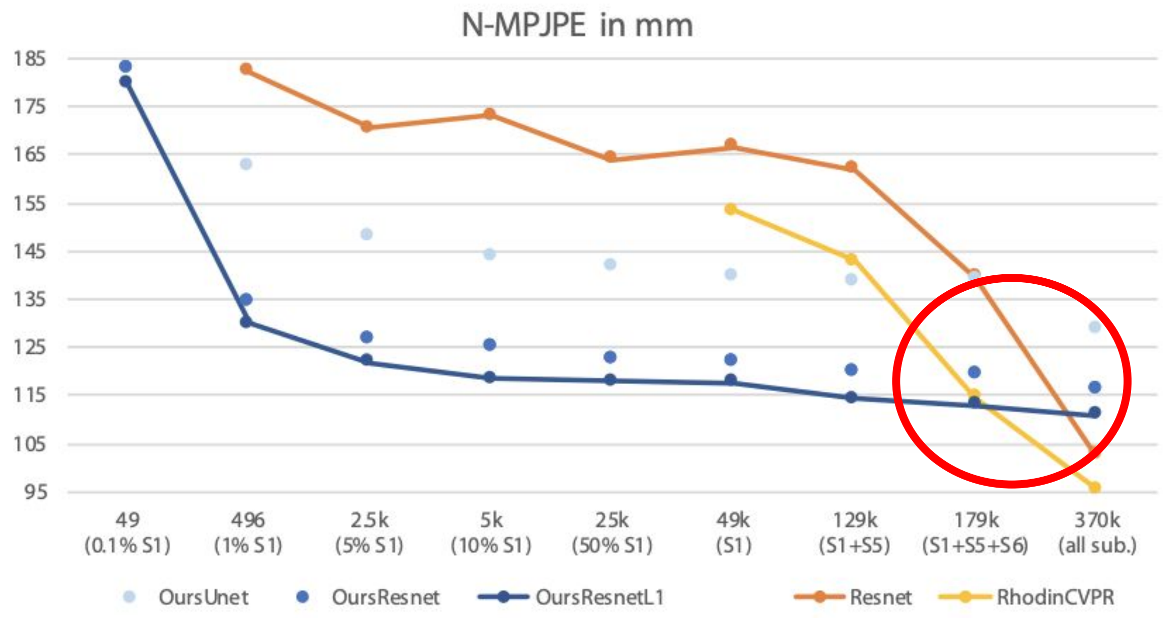
Using this model for pose estimation



Using this model for pose estimation



Using this model for pose estimation

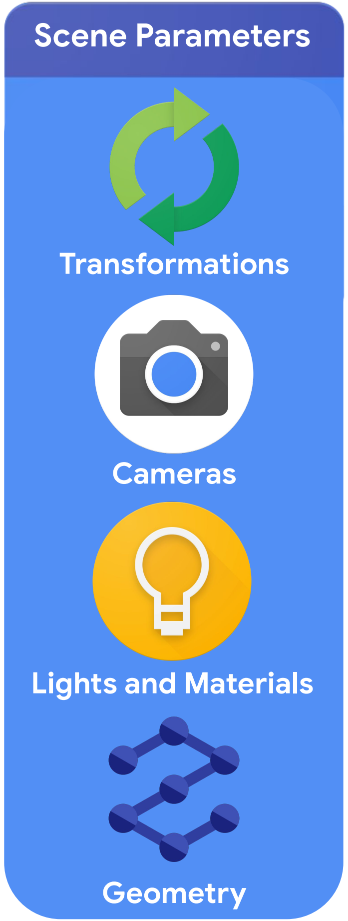


Computer graphics is a sub-field of computer science which studies methods for digitally **synthesizing** and **manipulating** visual content.

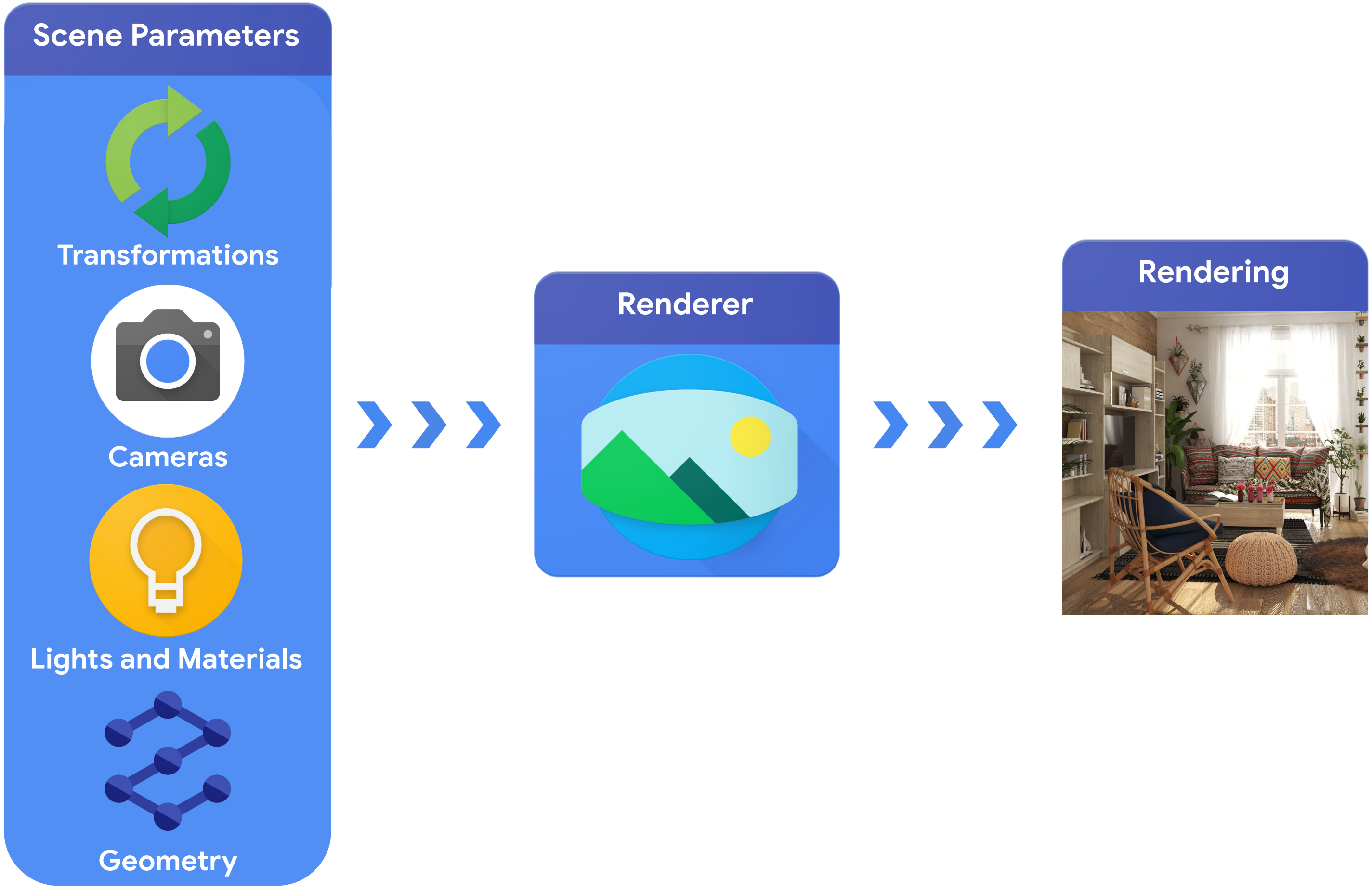
Wikipedia


CARS

Computer Graphics



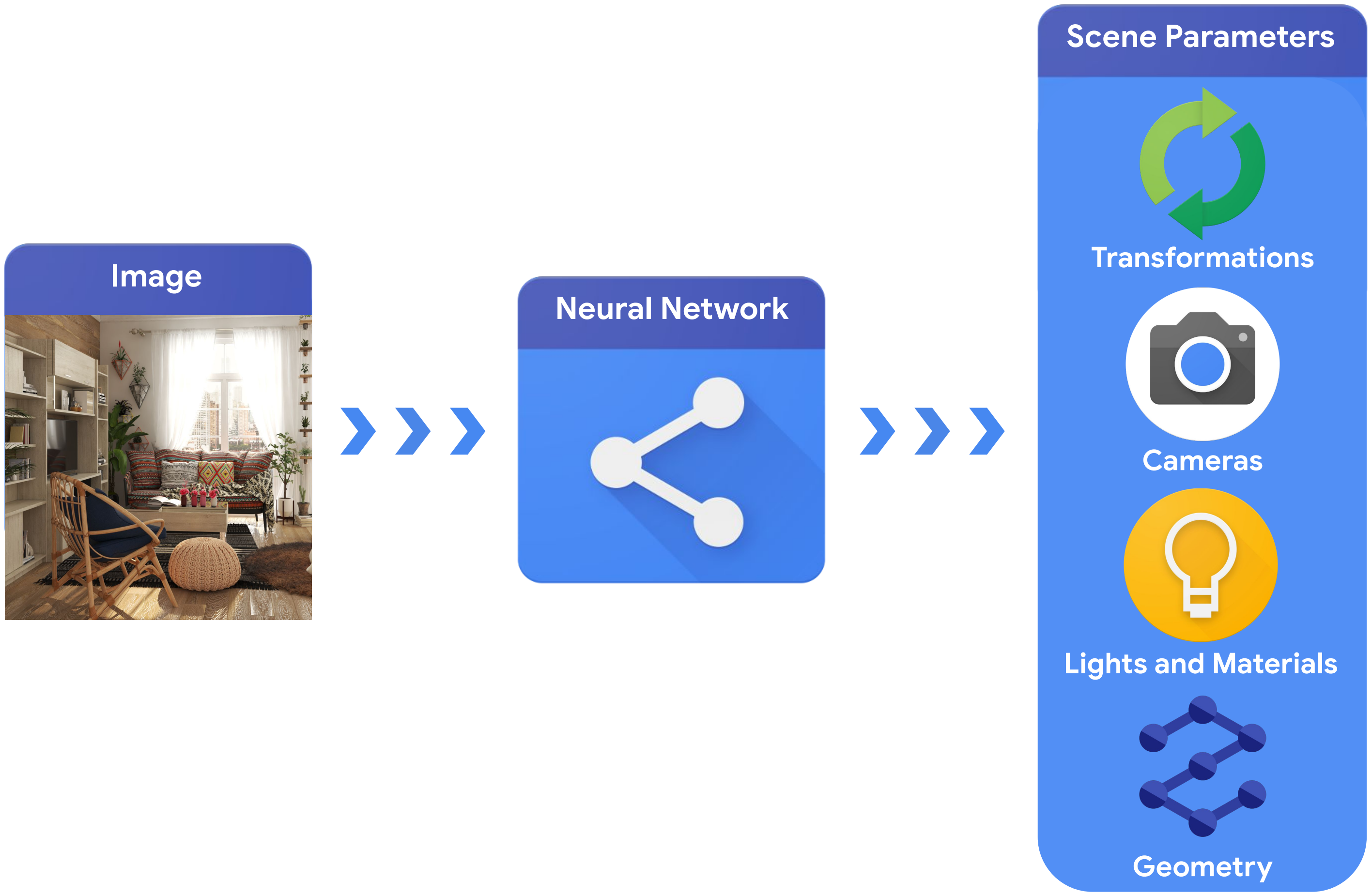
Computer Graphics



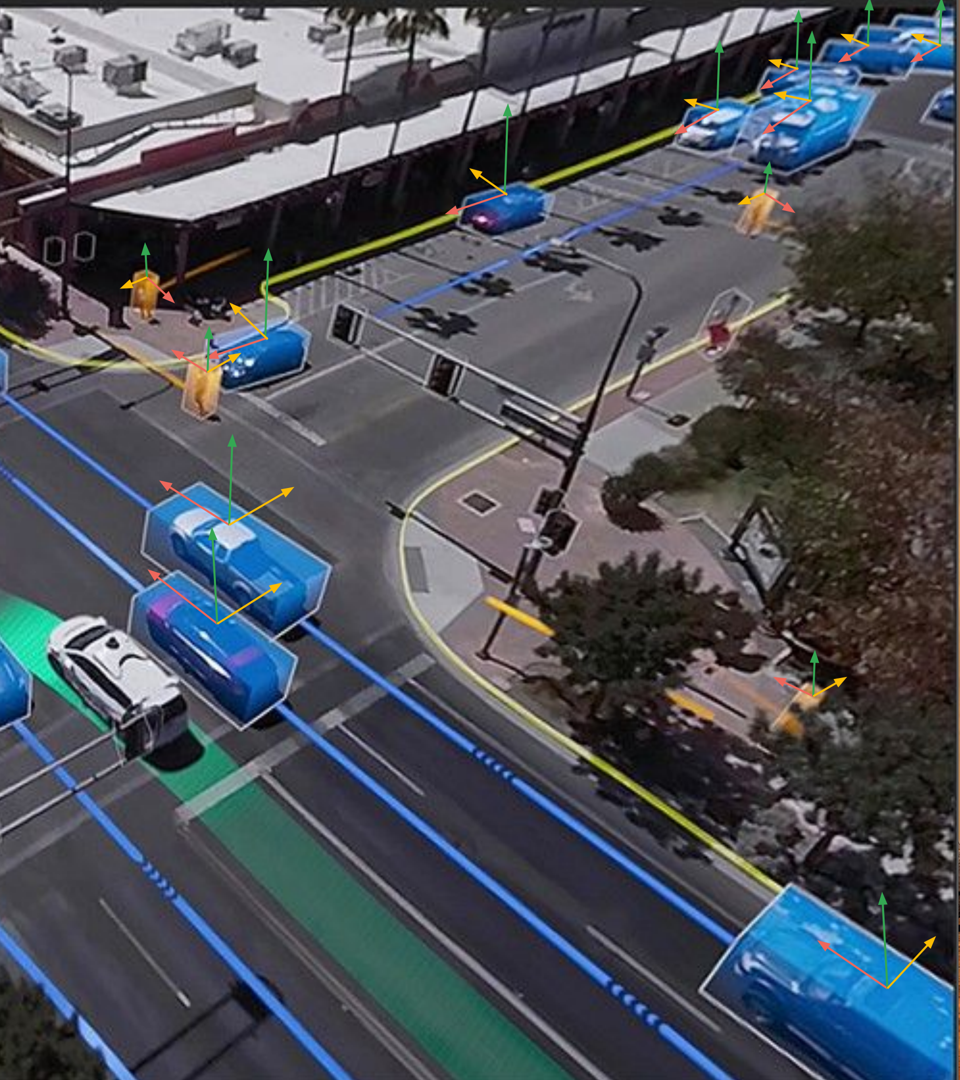
Computer vision is concerned with the theory behind artificial systems that **extract** information from images.

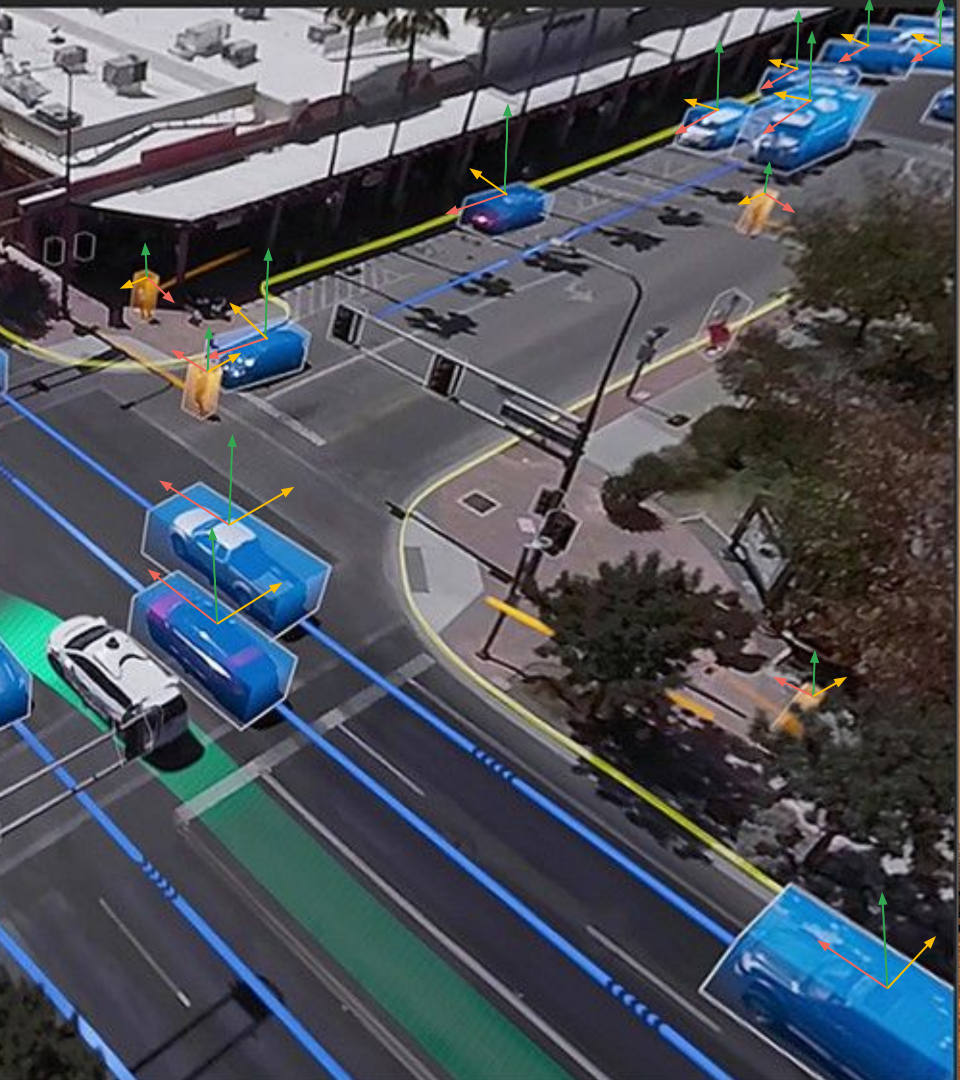
Wikipedia

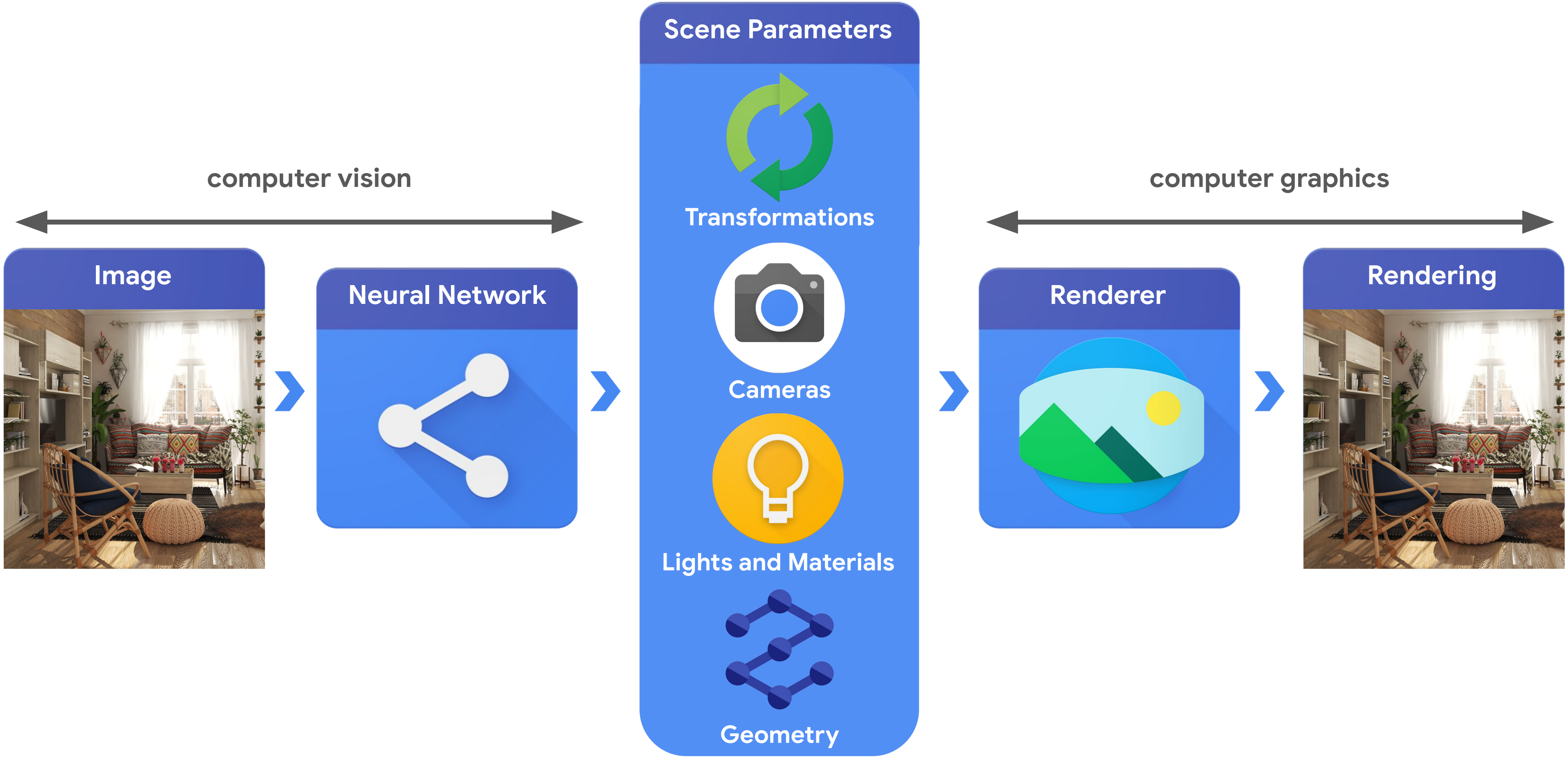
Computer Vision

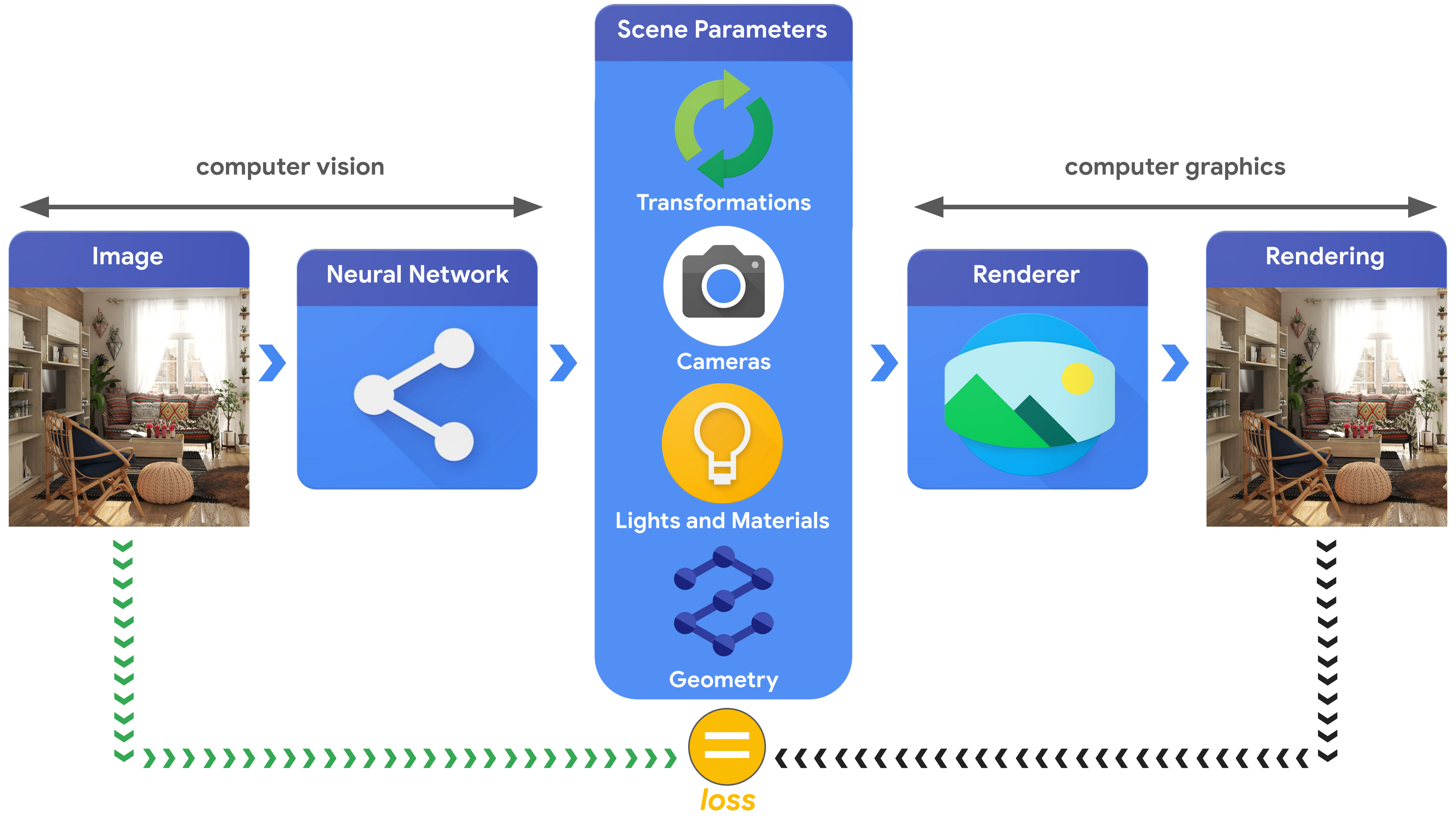


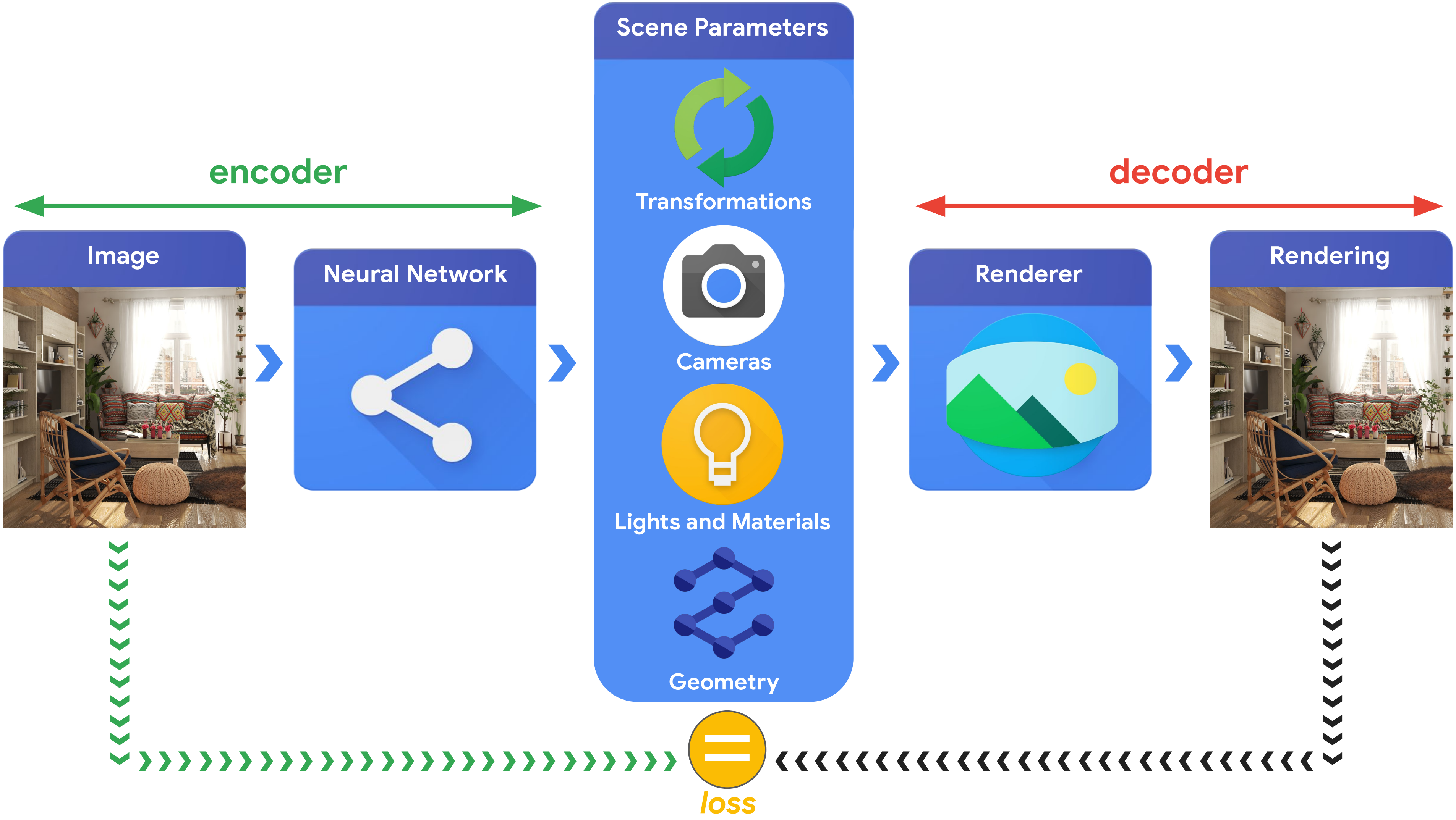
- Estimating the 3D position and orientation of an object.
- Understanding the material properties of an object.
- Recognizing an object based on its 3D geometry.






computer vision
computer graphics
Image
Neural Network
Scene Parameters
Transformations
Cameras
Lights and Materials
Geometry
Renderer
Rendering


Scene Parameters
Transformations
Cameras
Lights and Materials
Geometry
computer vision
computer graphics
Image
Neural Network
Renderer
Rendering
loss


encoder
Image
Neural Network
Scene Parameters
Transformations
Cameras
Lights and Materials
Geometry
decoder
Renderer
Rendering
loss

Differentiable Graphics Layers

Transformations	Cameras	Lights and Materials	Geometry	Renderers

Differentiable Graphics Layers

Transformations

Cameras

Lights and Materials

Geometry

Differentiable Graphics Layers

Transformations | Cameras | Lights and Materials | Geometry

- *Estimating the 3D orientation of an object.*
- *Understanding the material properties of an object.*
- *Recognizing an object based on its 3D geometry.*

TensorFlow
Graphics

Differentiable Graphics Layers

Transformations

Cameras

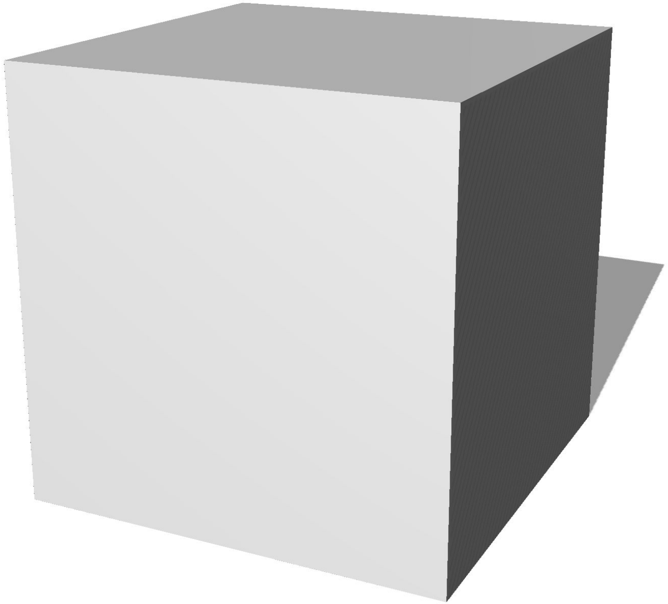
Materials

Geometry

Transformations

2D- 3D Rotations

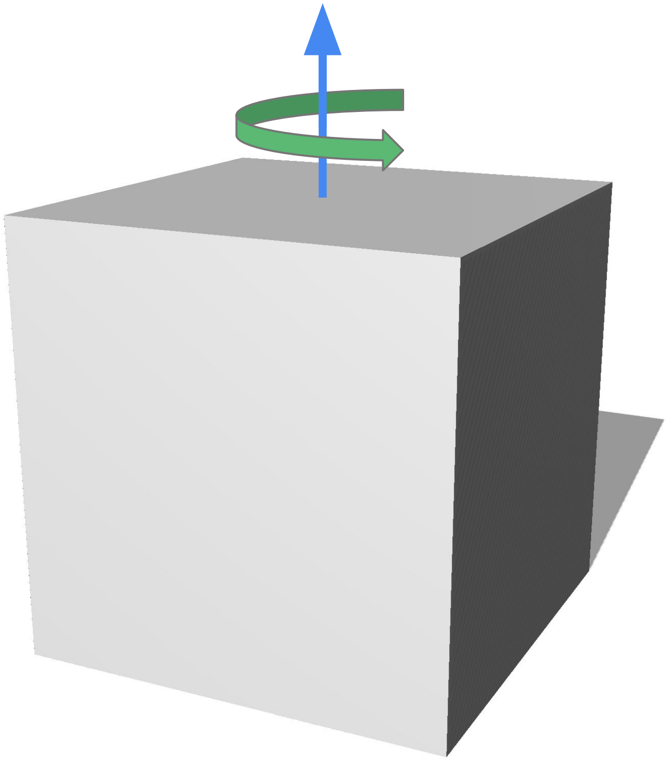
- Rotation matrices
- Euler angles
- Quaternions
- Axis-angles



Transformations

2D- 3D Rotations

- Rotation matrices
- Euler angles
- Quaternions
- **Axis-angles**

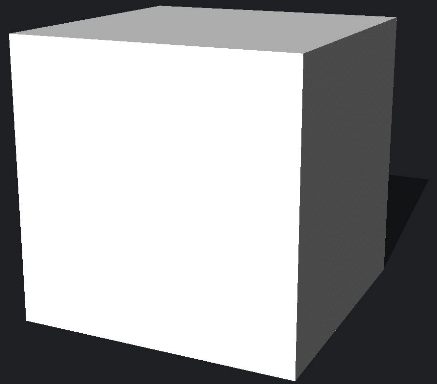


axis-angle cube rotation:

```python
import tensorflow_graphics.geometry.transformation as tfg_transformation


cube = load_cube() # cube vertices.
axis = (0., 1., 0.) # y axis.
angle = (np.pi / 4.,) # 45 degree angle.
cube_rotated = tfg_transformation.axis_angle.rotate(cube, axis, angle)
```

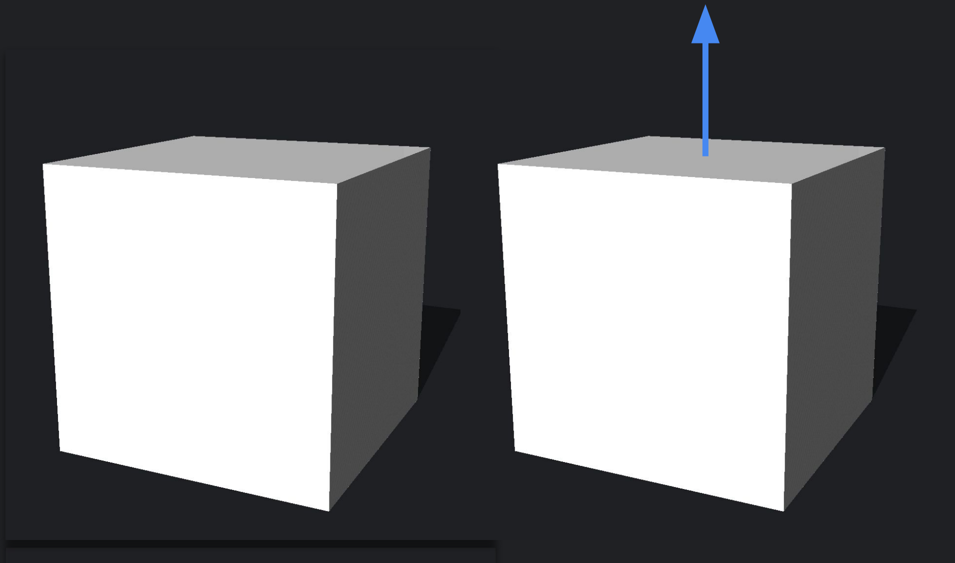
axis-angle rotation:

```python
import tensorflow_graphics.geometry.transformation as tfg_transformation


cube = load_cube() # cube vertices.
axis = (0., 1., 0.) # y axis.
angle = (np.pi / 4.,) # 45 degree angle.
cube_rotated = tfg_transformation.axis_angle.rotate(cube, axis, angle)
```



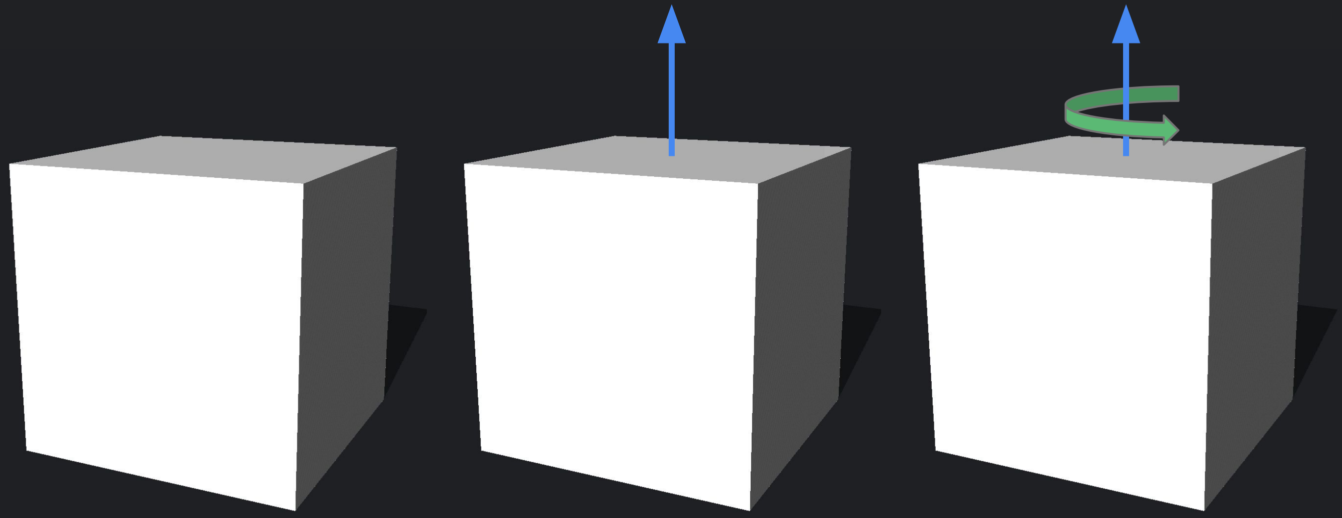
axis-angle rotation:

```
import tensorflow_graphics.geometry.transformation as tfg_transformation


cube = load_cube() # cube vertices.
axis = (0., 1., 0.) # y axis.
angle = (np.pi / 4.,) # 45 degree angle.
cube_rotated = tfg_transformation.axis_angle.rotate(cube, axis, angle)
```



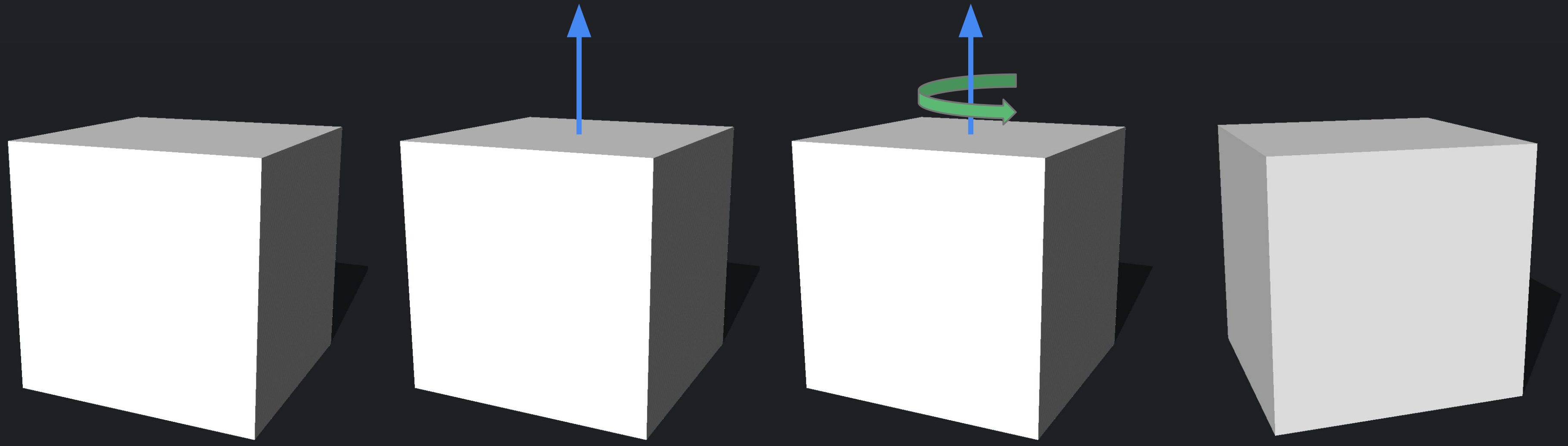
axis-angle rotation:

```
import tensorflow_graphics.geometry.transformation as tfg_transformation


cube = load_cube() # cube vertices.
axis = (0., 1., 0.) # y axis.
angle = (np.pi / 4.,) # 45 degree angle.
cube_rotated = tfg_transformation.axis_angle.rotate(cube, axis, angle)
```



axis-angle rotation:

```
import tensorflow_graphics.geometry.transformation as tfg_transformation


cube = load_cube() # cube vertices.
axis = (0., 1., 0.) # y axis.
angle = (np.pi / 4.,) # 45 degree angle.
cube_rotated = tfg_transformation.axis_angle.rotate(cube, axis, angle)
```



Differentiable Graphics Layers

Transformations

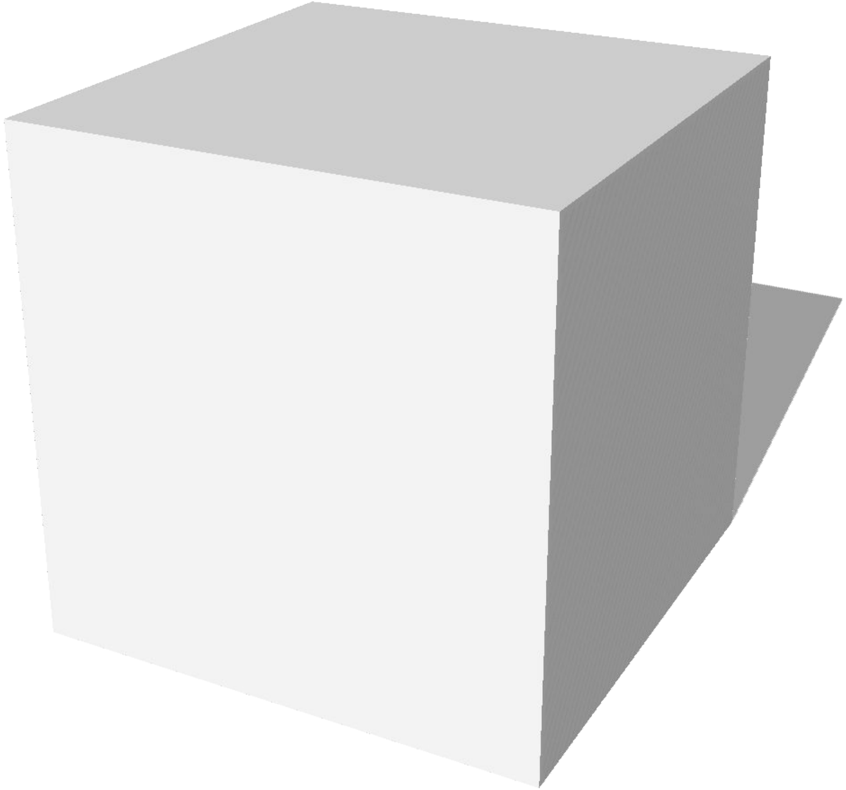
Cameras

Materials

Geometry

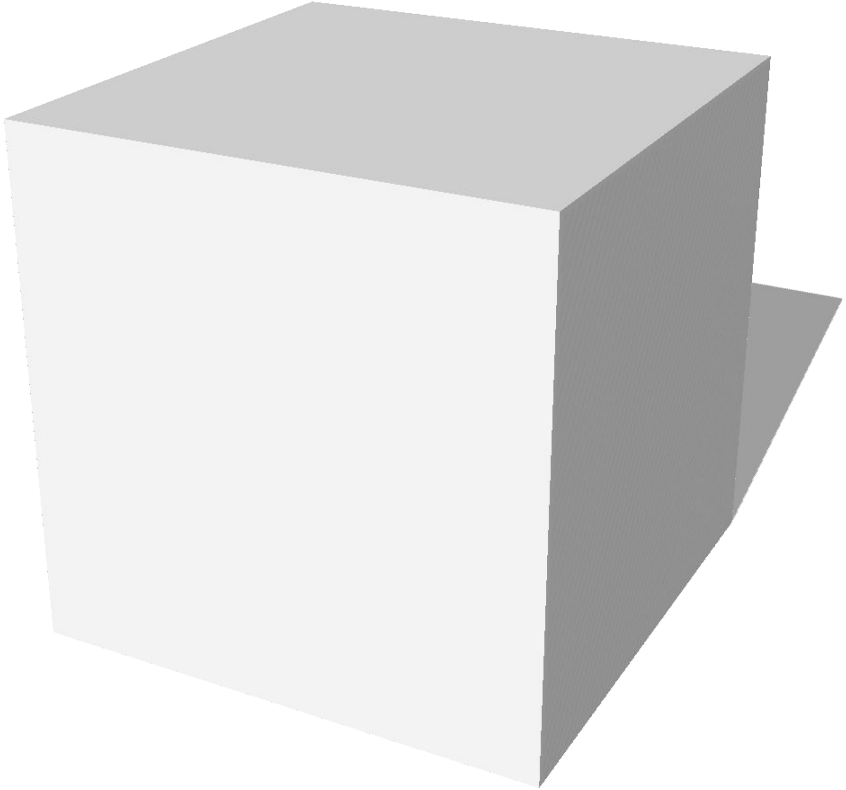
Cameras

- Orthographic
- Perspective



Cameras

- Orthographic
- **Perspective**

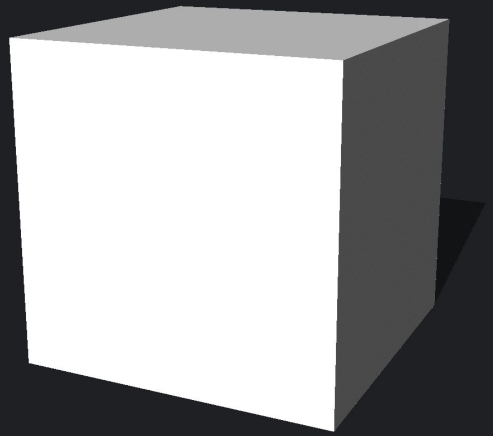


perspective camera projection:

```python
import tensorflow_graphics.rendering.camera as tfg_camera

cube = load_cube() # cube vertices.
focal = (100., 100.) # focal length of the camera.
principal_point = (256., 256.) # principal point of the camera.
projected_cube = tfg_camera.perspective.project(points, focal, principal_point)
```

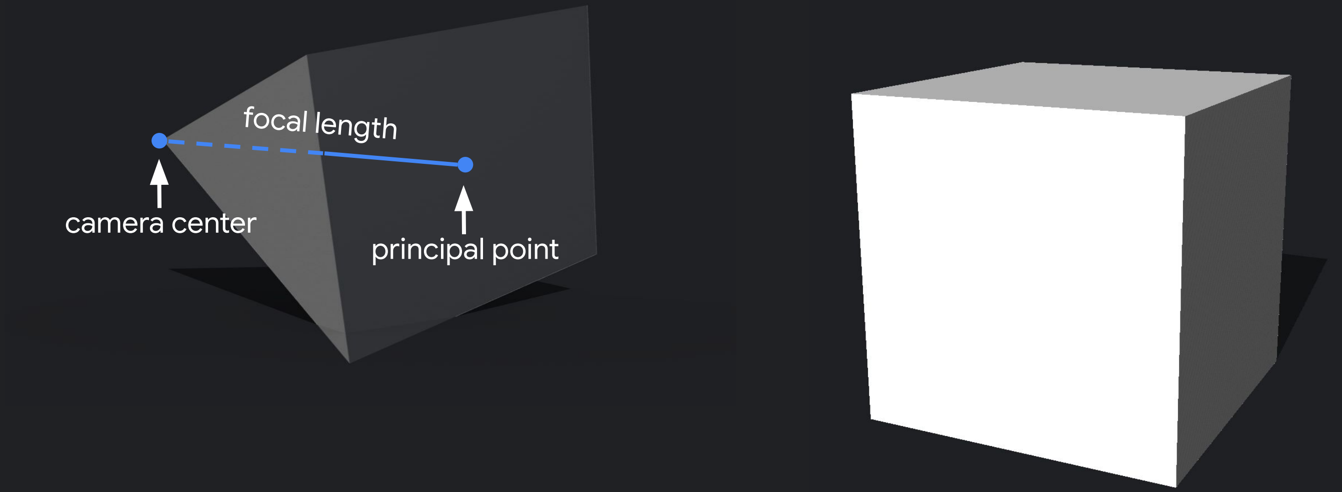
perspective camera projection:

```
import tensorflow_graphics.rendering.camera as tfg_camera

cube = load_cube() # cube vertices.
focal = (100., 100.) # focal length of the camera.
principal_point = (256., 256.) # principal point of the camera.
projected_cube = tfg_camera.perspective.project(points, focal, principal_point)
```



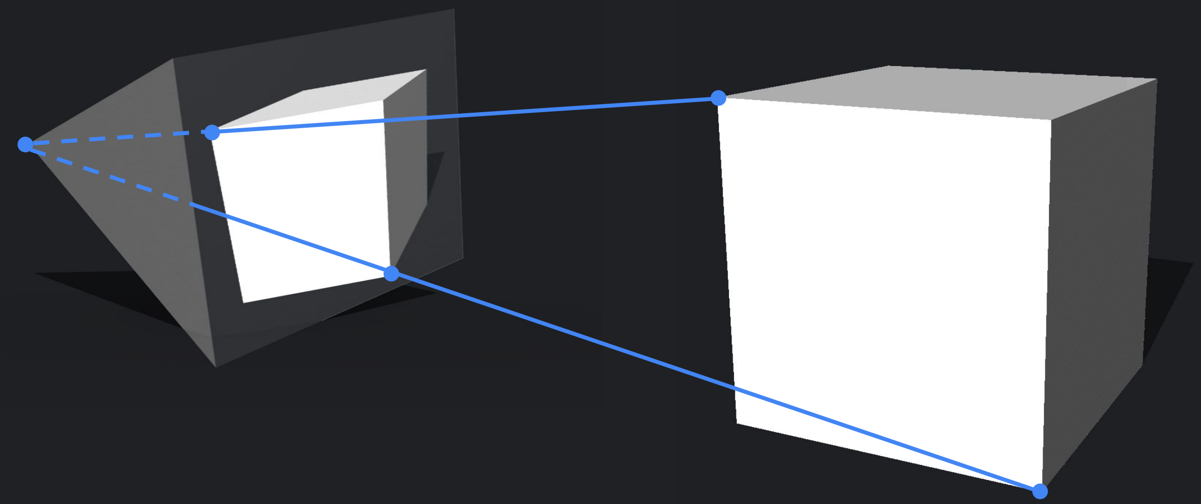
perspective camera projection:

```
import tensorflow_graphics.rendering.camera as tfg_camera

cube = load_cube() # cube vertices.
focal = (100., 100.) # focal length of the camera.
principal_point = (256., 256.) # principal point of the camera.
projected_cube = tfg_camera.perspective.project(points, focal, principal_point)
```



perspective camera projection:

```
import tensorflow_graphics.rendering.camera as tfg_camera

cube = load_cube() # cube vertices.
focal = (100., 100.) # focal length of the camera.
principal_point = (256., 256.) # principal point of the camera.
projected_cube = tfg_camera.perspective.project(points, focal, principal_point)
```



Differentiable Graphics Layers

Transformations

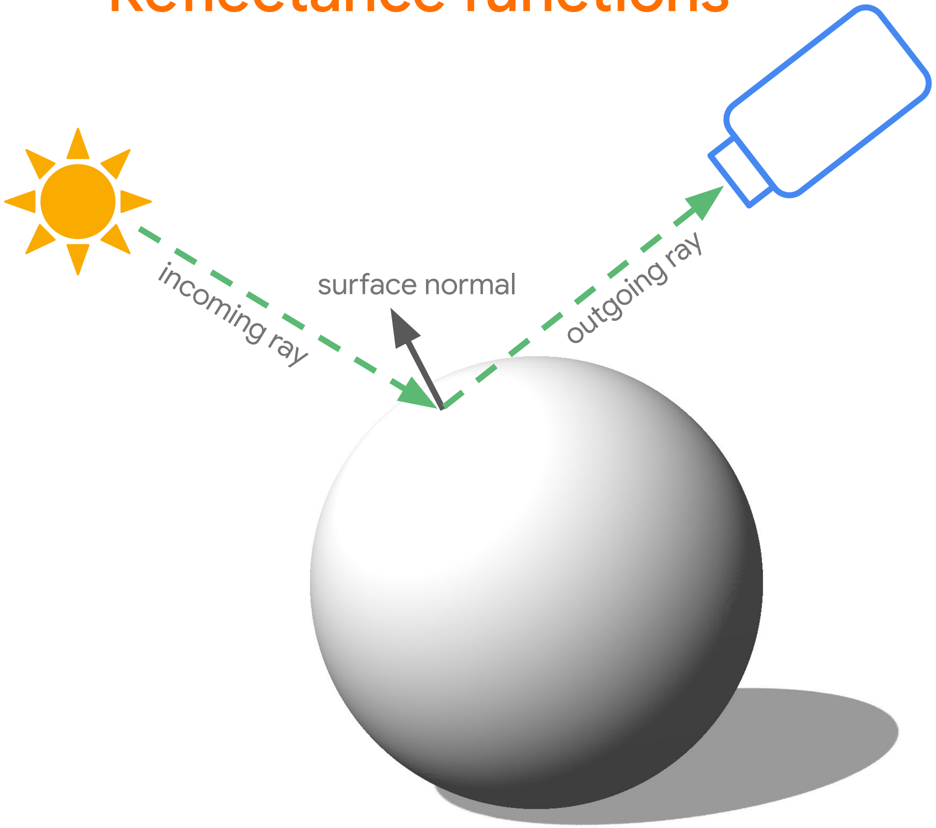
Cameras

Materials

Geometry

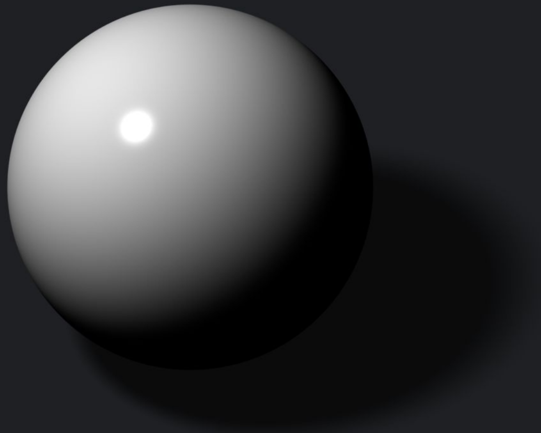
Materials

Reflectance functions



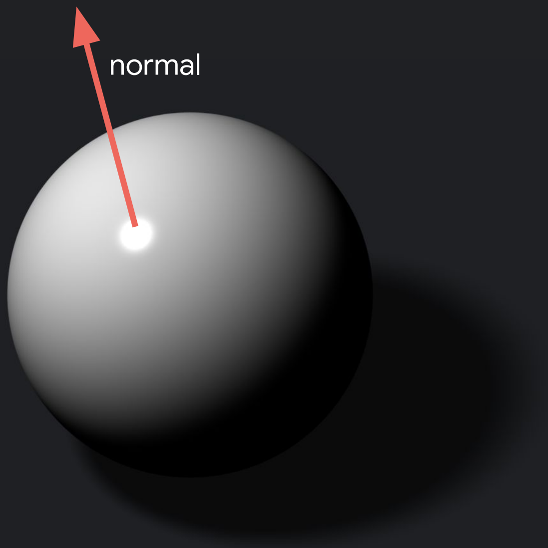
material model:

```python
import tensorflow_graphics.rendering.reflectance as tfg_reflectance

surface_normal = (0., 1., 0.) # surface normal.
incoming_ray = (100., 100.) # incoming ray from the light.
outgoing_ray = (256., 256.) # outgoing ray toward the camera.
color = (1., 1., 1.) # color of the surface.
shininess = (0.5,) # shininess of the surface.
output_color = tfg_reflectance.blinn_phong.brdf(incoming_ray, outgoing_ray,
                                                surface_normal, shininess, color)
```



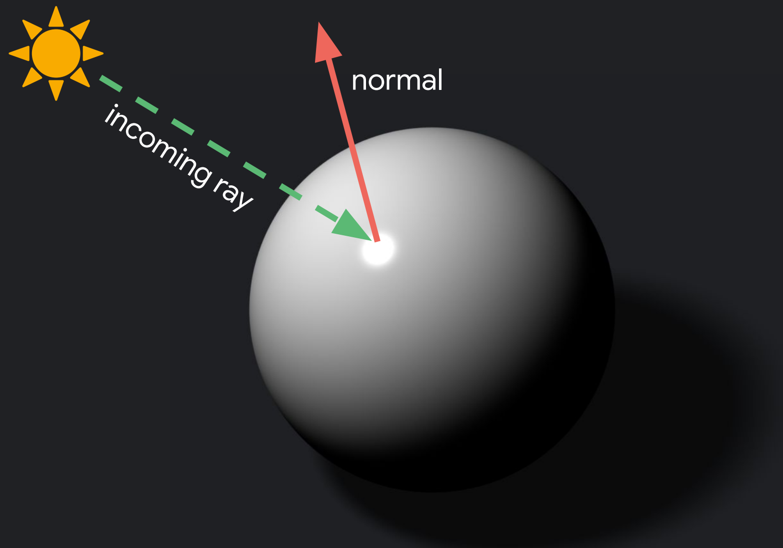
material model:

```
import tensorflow_graphics.rendering.reflectance as tfg_reflectance

surface_normal = (0., 1., 0.) # surface normal.
incoming_ray = (100., 100.) # incoming ray from the light.
outgoing_ray = (256., 256.) # outgoing ray toward the camera.
color = (1., 1., 1.) # color of the surface.
shininess = (0.5,) # shininess of the surface.
output_color = tfg_reflectance.blinn_phong.brdf(incoming_ray, outgoing_ray,
                                                 surface_normal, shininess, color)
```



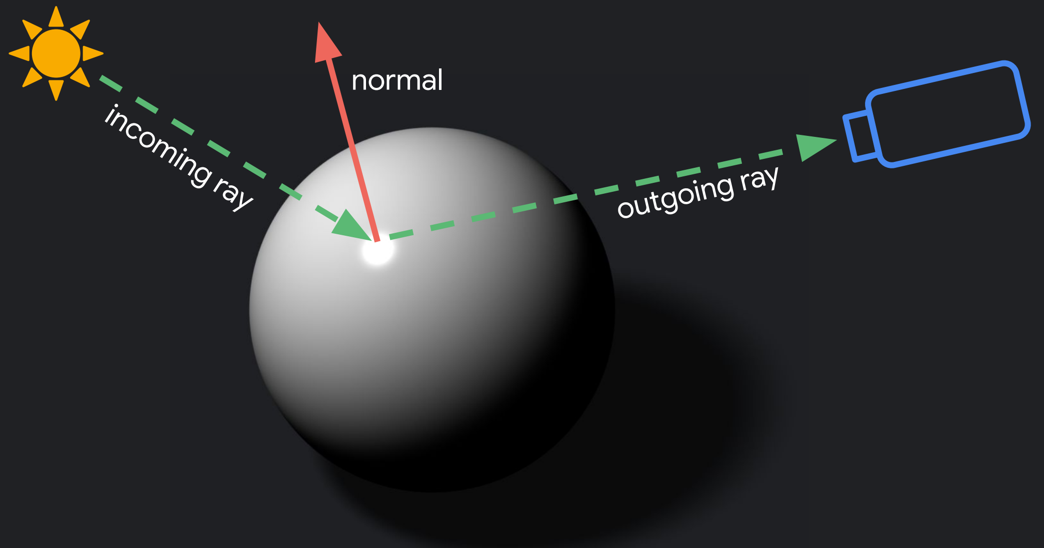
material model:

```
import tensorflow_graphics.rendering.reflectance as tfg_reflectance

surface_normal = (0., 1., 0.) # surface normal.
incoming_ray = (100., 100.) # incoming ray from the light.
outgoing_ray = (256., 256.) # outgoing ray toward the camera.
color = (1., 1., 1.) # color of the surface.
shininess = (0.5,) # shininess of the surface.
output_color = tfg_reflectance.blinn_phong.brdf(incoming_ray, outgoing_ray,
                                                 surface_normal, shininess, color)
```



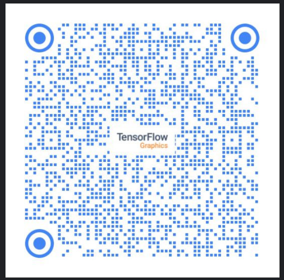
material model:

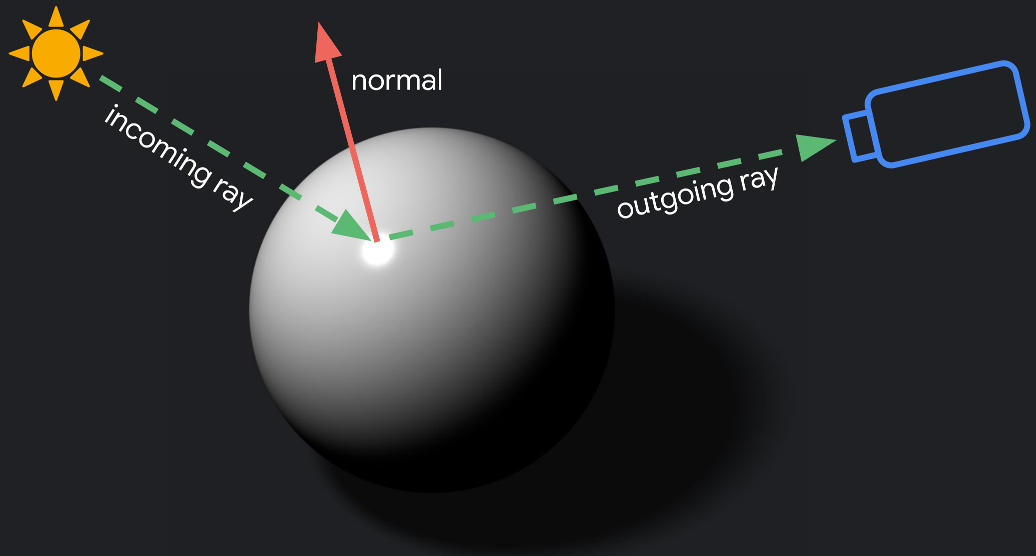
```
import tensorflow_graphics.rendering.reflectance as tfg_reflectance

surface_normal = (0., 1., 0.) # surface normal.
incoming_ray = (100., 100.) # incoming ray from the light.
outgoing_ray = (256., 256.) # outgoing ray toward the camera.
color = (1., 1., 1.) # color of the surface.
shininess = (0.5,) # shininess of the surface.
output_color = tfg_reflectance.blinn_phong.brdf(incoming_ray, outgoing_ray,
                                                surface_normal, shininess, color)
```



material model:

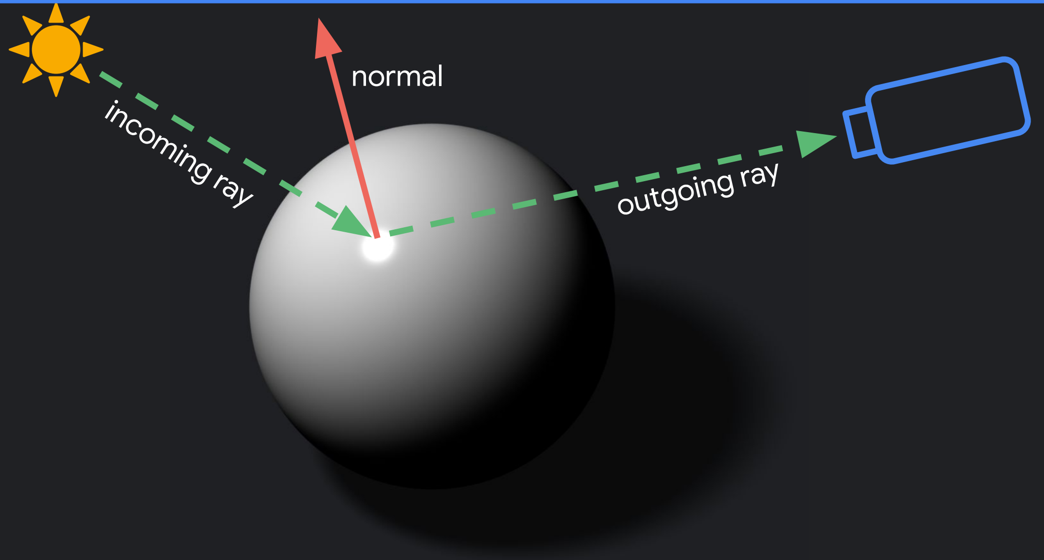
```python
import tensorflow_graphics.rendering.reflectance as tfg_reflectance

surface_normal = (0., 1., 0.) # surface normal.
incoming_ray = (100., 100.) # incoming ray from the light.
outgoing_ray = (256., 256.) # outgoing ray toward the camera.
color = (1., 1., 1.) # color of the surface.
shininess = (0.5,) # shininess of the surface.
output_color = tfg_reflectance.blinn_phong.brdf(incoming_ray, outgoing_ray,
                                                surface_normal, shininess, color)
```





material model:

```
import tensorflow_graphics.rendering.reflectance as tfg_reflectance

surface_normal = (0., 1., 0.) # surface normal.
incoming_ray = (100., 100.) # incoming ray from the light.
outgoing_ray = (256., 256.) # outgoing ray toward the camera.
color = (1., 1., 1.) # color of the surface.
shininess = (0.5,) # shininess of the surface.
output_color = tfg_reflectance.blinn_phong.brdf(incoming_ray, outgoing_ray,
                                                surface_normal, shininess, color)
```

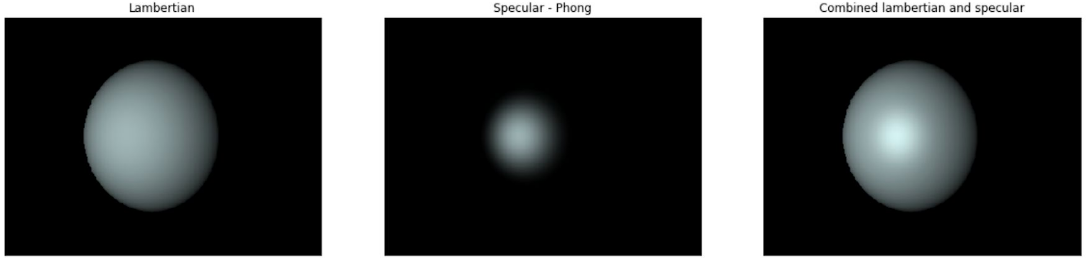


Colab code sample: reflectance

Controllable lighting of a sphere

Controls

`light_x_position:`	-0.4
`albedo_red:`	0.7
`albedo_green:`	1
`albedo_blue:`	1
`light_red:`	1
`light_green:`	1
`light_blue:`	1
`specular_percentage:`	0.25
`shininess:`	4



Differentiable Graphics Layers

Transformations

Cameras

Materials

Geometry

Image Convolution

A basic building block of Deep Learning

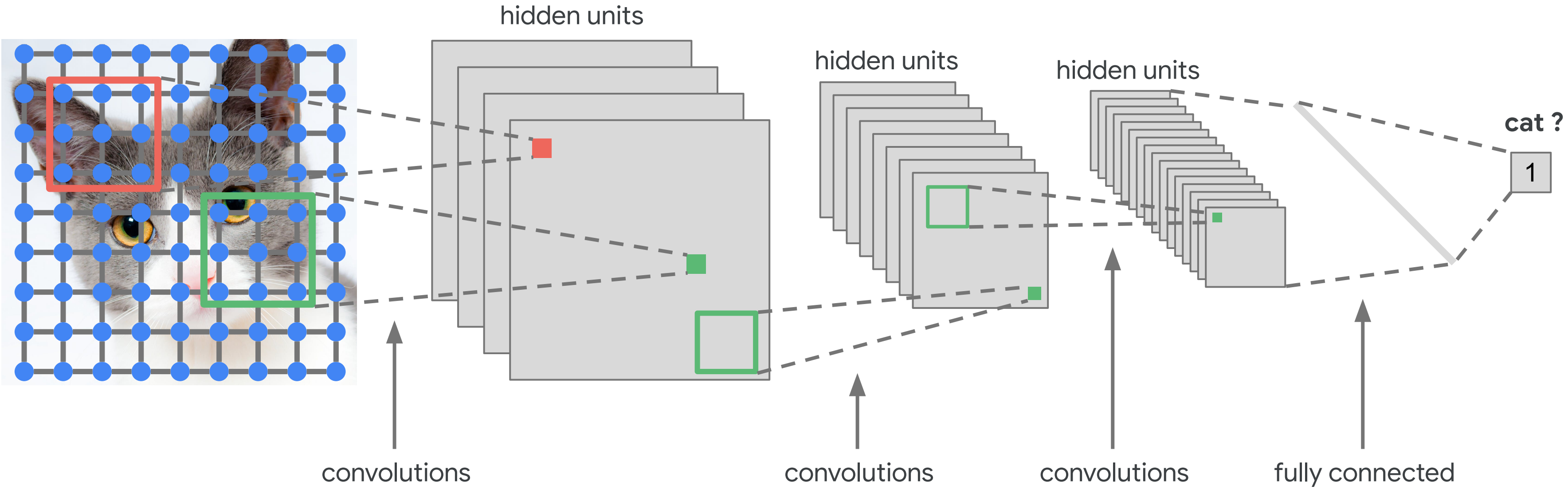
cat?

cat?

cat?

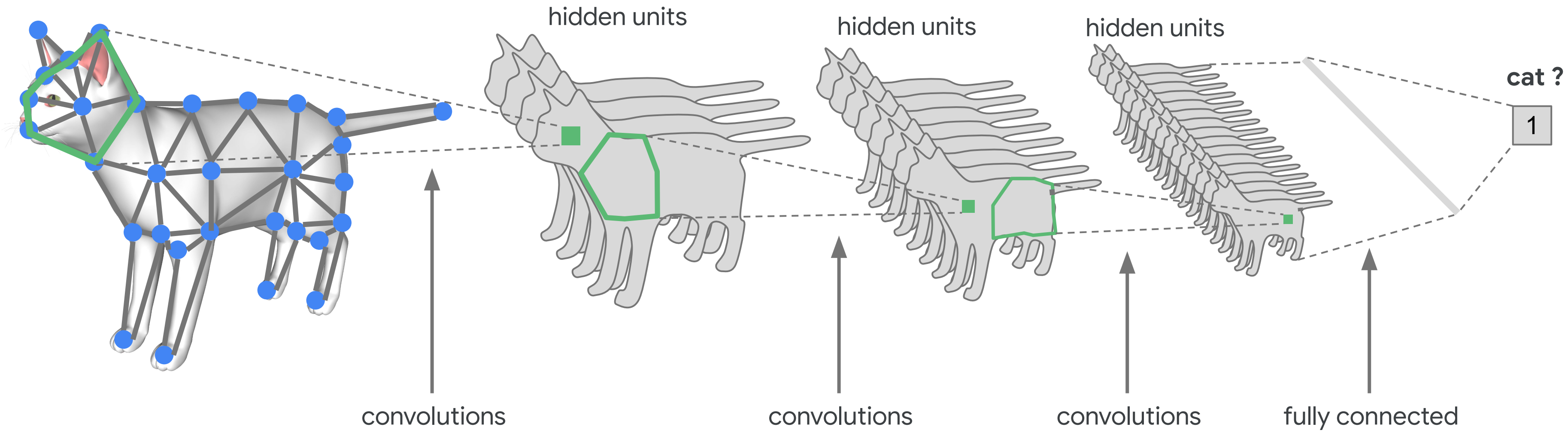
Image Convolution

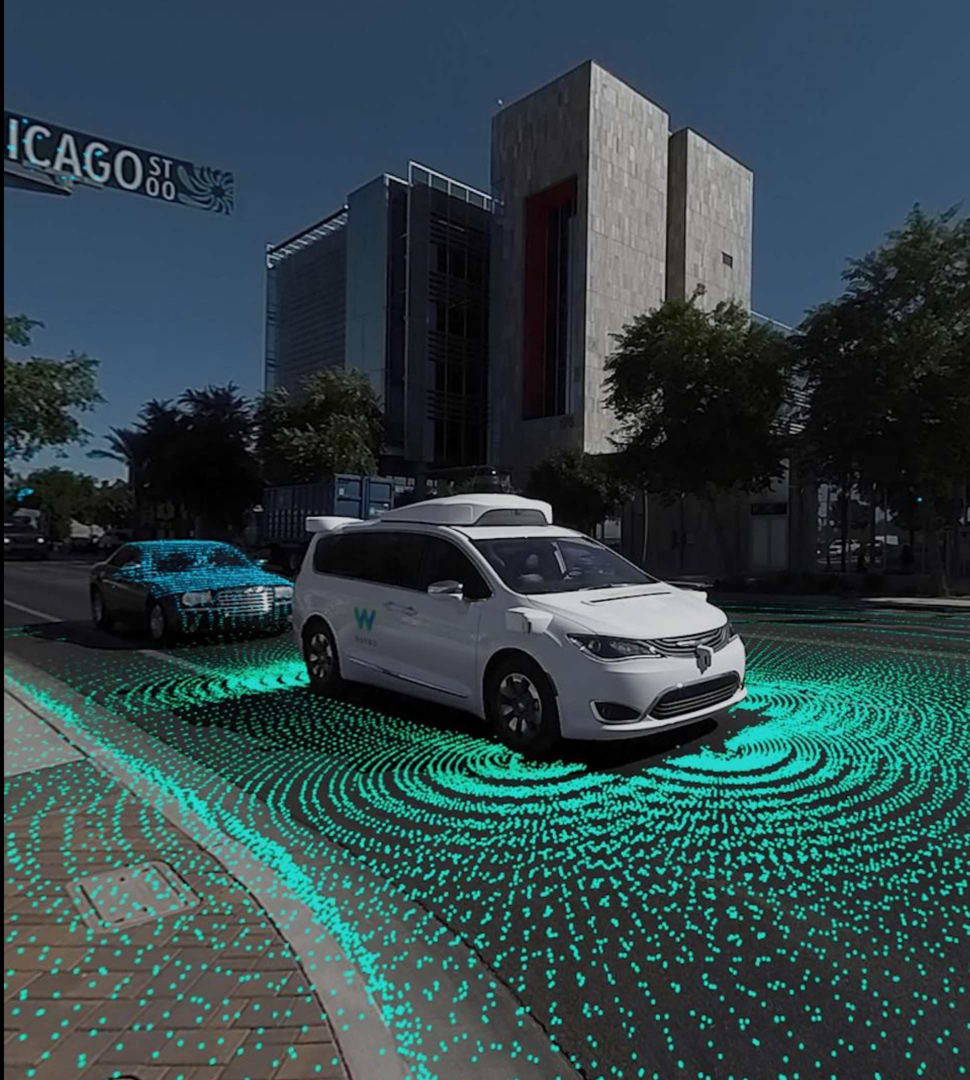
A basic building block of Deep Learning



Graph Convolution

A basic building block of Deep Learning



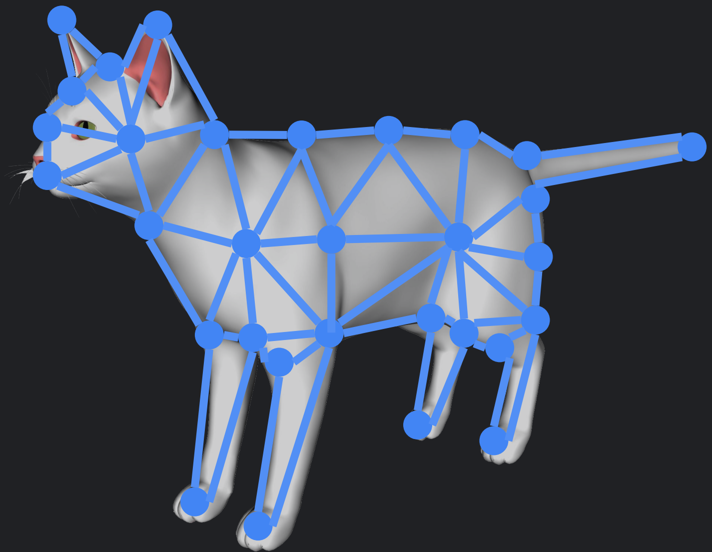

ICAGO ST
00

graph convolution:

```
import tensorflow as tf
import tensorflow_graphics.nn.layer.graph_convolution as tf_graph_conv

vertices, connectivity = load_mesh() # mesh vertices and connectivity.
output = tf_graph_conv.feature_steered_convolution_layer(vertices, connectivity)
output = tf.nn.relu(output)
```

graph convolution:

```python
import tensorflow as tf
import tensorflow_graphics.nn.layer.graph_convolution as tf_graph_conv

vertices, connectivity = load_mesh() # mesh vertices and connectivity.
output = tf_graph_conv.feature_steered_convolution_layer(vertices, connectivity)
output = tf.nn.relu(output)
```



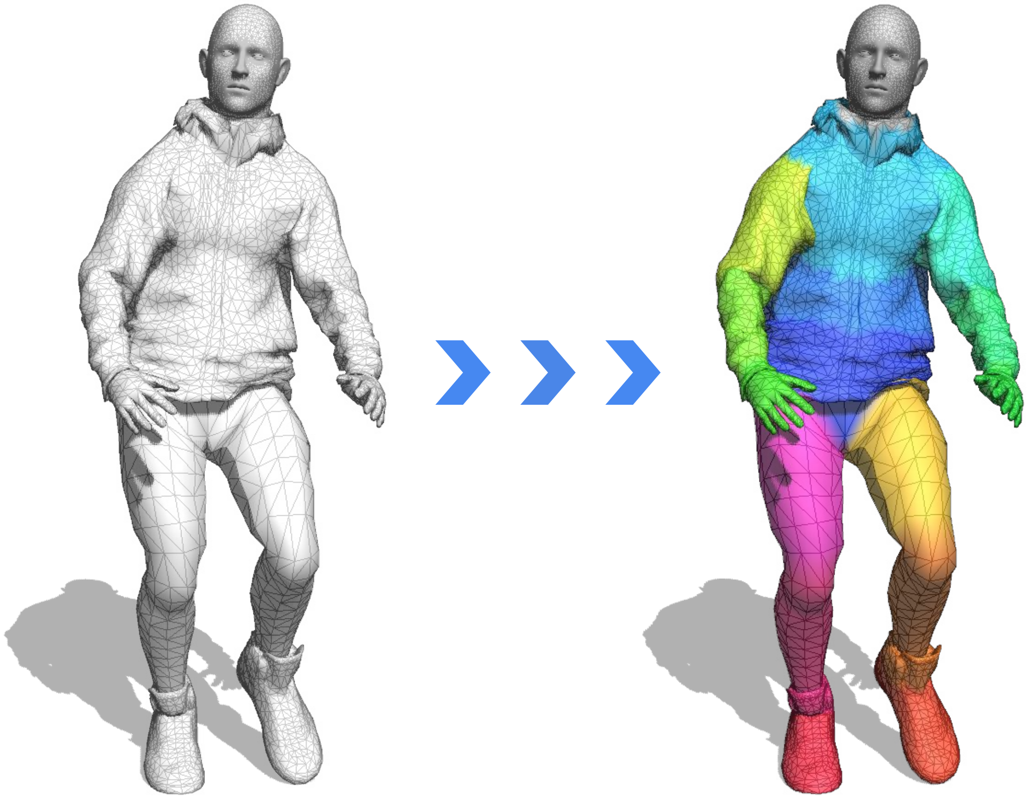
graph convolution:

```python
import tensorflow as tf
import tensorflow_graphics.nn.layer.graph_convolution as tf_graph_conv

vertices, connectivity = load_mesh() # mesh vertices and connectivity.
output = tf_graph_conv.feature_steered_convolution_layer(vertices, connectivity)
output = tf.nn.relu(output)
```

graph convolution:

```
import tensorflow as tf
import tensorflow_graphics.nn.layer.graph_convolution as tf_graph_conv

vertices, connectivity = load_mesh() # mesh vertices and connectivity.
output = tf_graph_conv.feature_steered_convolution_layer(vertices, connectivity)
output = tf.nn.relu(output)
```

Colab code sample
3D Semantic Segmentation



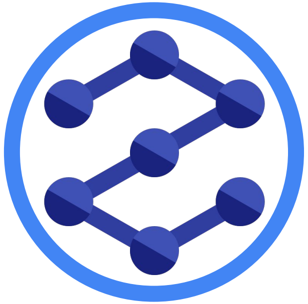
- head
- chest
- abdomen
- pelvis
- Left upper arm
- left lower arm
- left hand
- right upper arm
- right lower arm
- right hand
- left upper leg
- left lower leg
- left foot
- right upper leg
- right lower leg
- right foot

Differentiable Graphics Layers

Transformations

Cameras

Materials



Geometry

Differentiable Graphics Layers

Transformations

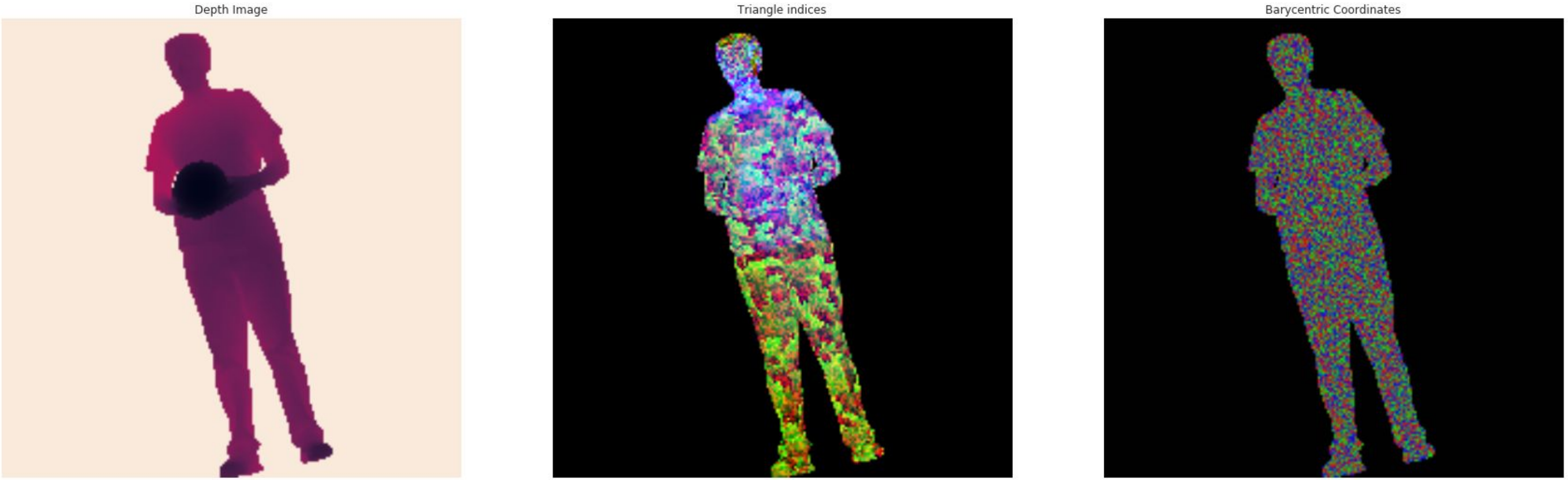
Cameras

Lights and Materials

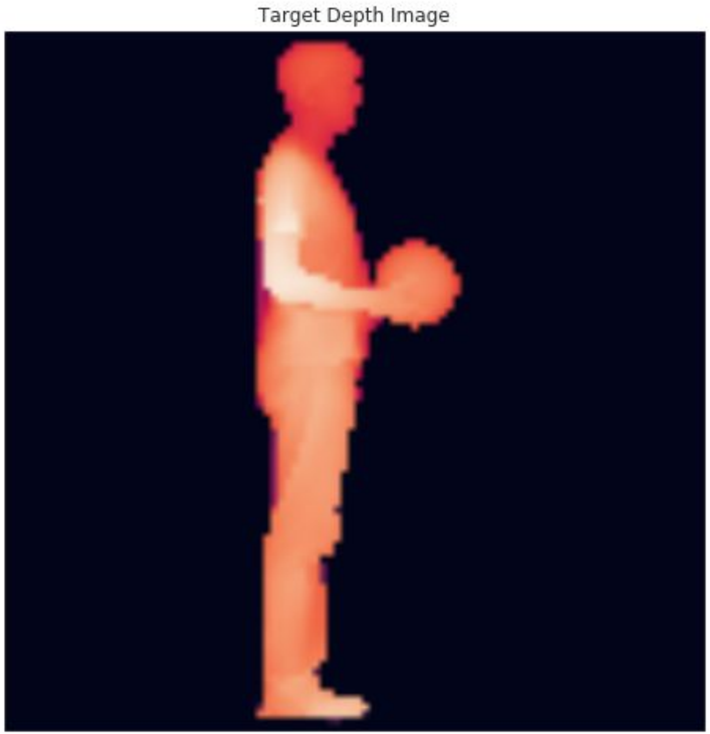
Geometry

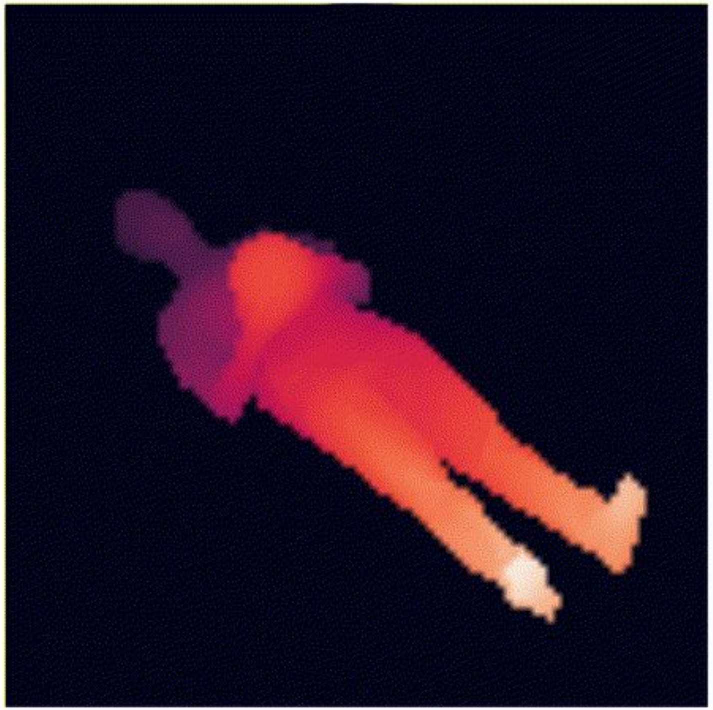
Renderers

Differentiable rasterizer

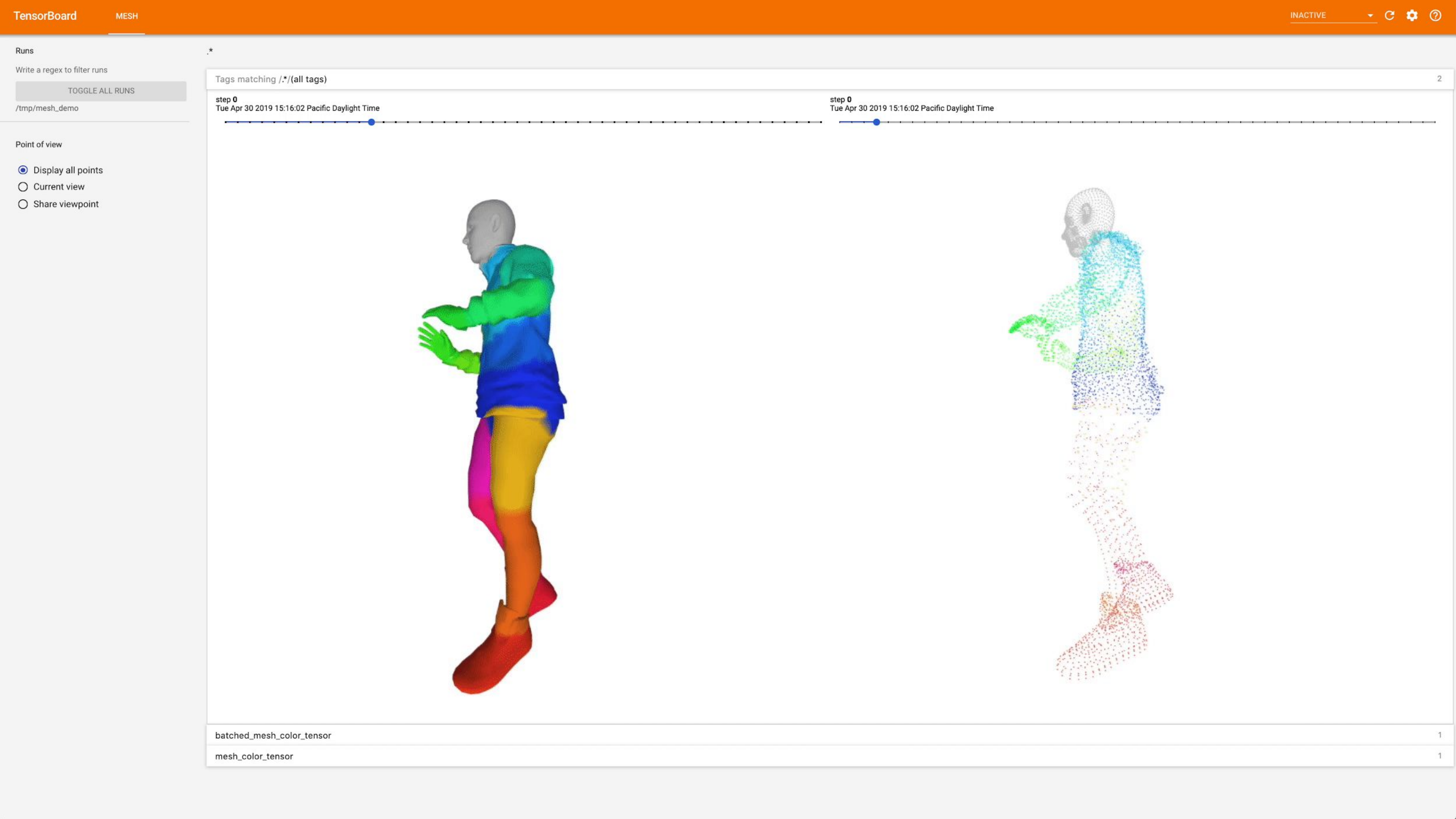


Differentiable rasterizer





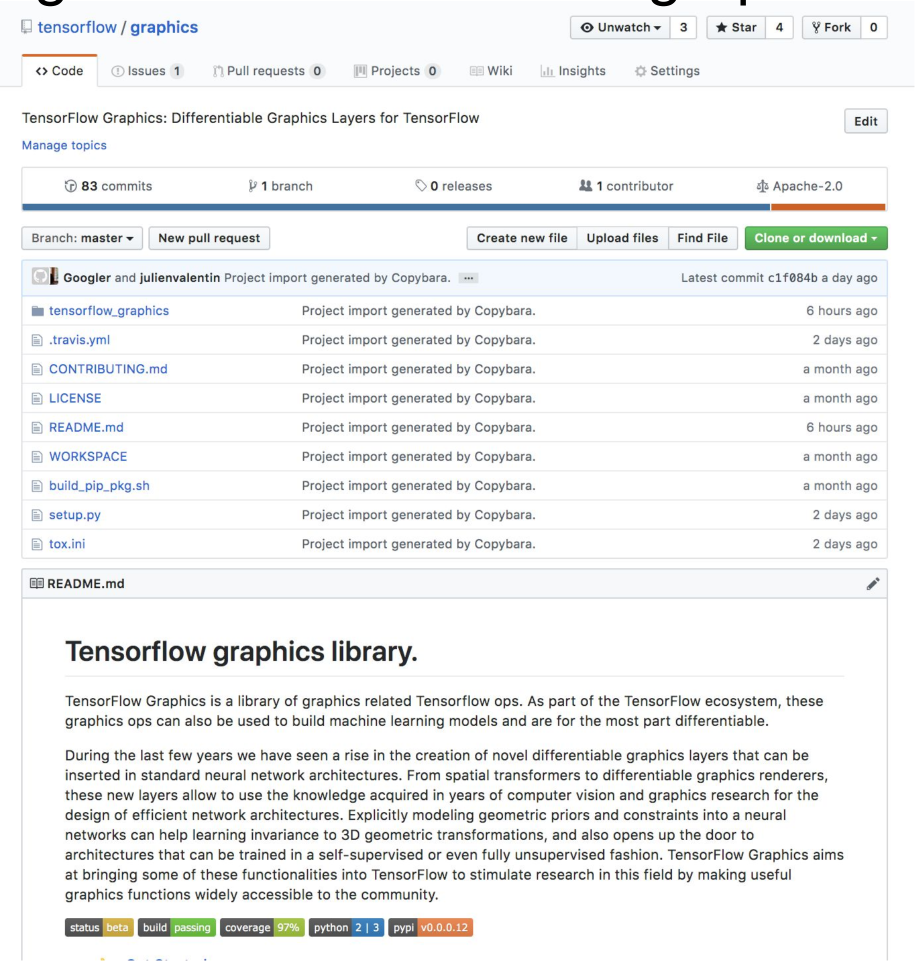
One more thing...


TensorBoard
MESH
INACTIVE
Runs
Write a regex to filter runs
TOGGLE ALL RUNS
/tmp/mesh_demo
Point of view
Display all points
Current view
Share viewpoint
.*
Tags matching /.*/(all tags)
2
step 0
Tue Apr 30 2019 15:16:02 Pacific Daylight Time
step 0
Tue Apr 30 2019 15:16:02 Pacific Daylight Time
batched_mesh_color_tensor
1
mesh_color_tensor
1

Get Started Today!

pip install tensorflow-graphics

github.com/tensorflow/graphics


tensorflow / graphics

Unwatch 3 | Star 4 | Fork 0

Code | Issues 1 | Pull requests 0 | Projects 0 | Wiki | Insights | Settings

TensorFlow Graphics: Differentiable Graphics Layers for TensorFlow

Edit

Manage topics

83 commits | 1 branch | 0 releases | 1 contributor | Apache-2.0

Branch: master | New pull request | Create new file | Upload files | Find File | Clone or download

Googler and julienvalentin Project import generated by Copybara. ... Latest commit c1f084b a day ago

tensorflow_graphics	Project import generated by Copybara.	6 hours ago
.travis.yml	Project import generated by Copybara.	2 days ago
CONTRIBUTING.md	Project import generated by Copybara.	a month ago
LICENSE	Project import generated by Copybara.	a month ago
README.md	Project import generated by Copybara.	6 hours ago
WORKSPACE	Project import generated by Copybara.	a month ago
build_pip_pkg.sh	Project import generated by Copybara.	a month ago
setup.py	Project import generated by Copybara.	2 days ago
tox.ini	Project import generated by Copybara.	2 days ago

README.md

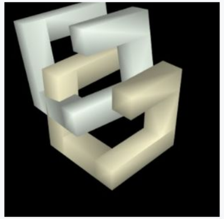
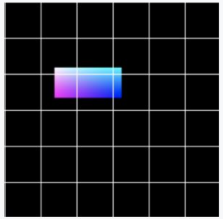
Tensorflow graphics library.

TensorFlow Graphics is a library of graphics related Tensorflow ops. As part of the TensorFlow ecosystem, these graphics ops can also be used to build machine learning models and are for the most part differentiable.

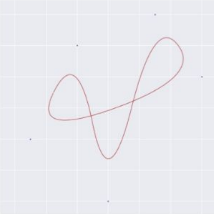
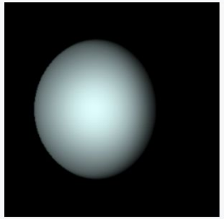
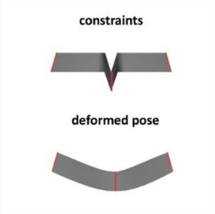
During the last few years we have seen a rise in the creation of novel differentiable graphics layers that can be inserted in standard neural network architectures. From spatial transformers to differentiable graphics renderers, these new layers allow to use the knowledge acquired in years of computer vision and graphics research for the design of efficient network architectures. Explicitly modeling geometric priors and constraints into a neural networks can help learning invariance to 3D geometric transformations, and also opens up the door to architectures that can be trained in a self-supervised or even fully unsupervised fashion. TensorFlow Graphics aims at bringing some of these functionalities into TensorFlow to stimulate research in this field by making useful graphics functions widely accessible to the community.

status beta | build passing | coverage 97% | python 2 | 3 | pypi v0.0.0.12

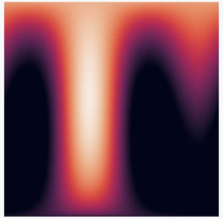
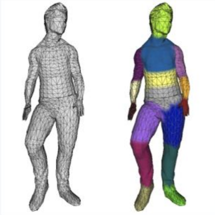
Beginner

Object pose estimation	Camera intrinsics optimization
	

Intermediate

B-spline and slerp interpolation	Reflectance	Non-rigid surface deformation
		

Advanced

Spherical harmonics rendering	Environment map optimization	Semantic mesh segmentation
		

TensorFlow Graphics

Julien Valentin

Google, @JPCValentin

Cem Keskin

Google

Pavel Pidlypenskyi

Google, @podlipensky

Ameesh Makadia

Google, @kiamada

Avneesh Sud

Google, @AvneeshSud

Sofien Bouaziz

Google, @_sofien_

TensorFlow

Advances in using 3D and Graphics Layers For Deep Learning

Christian Häne

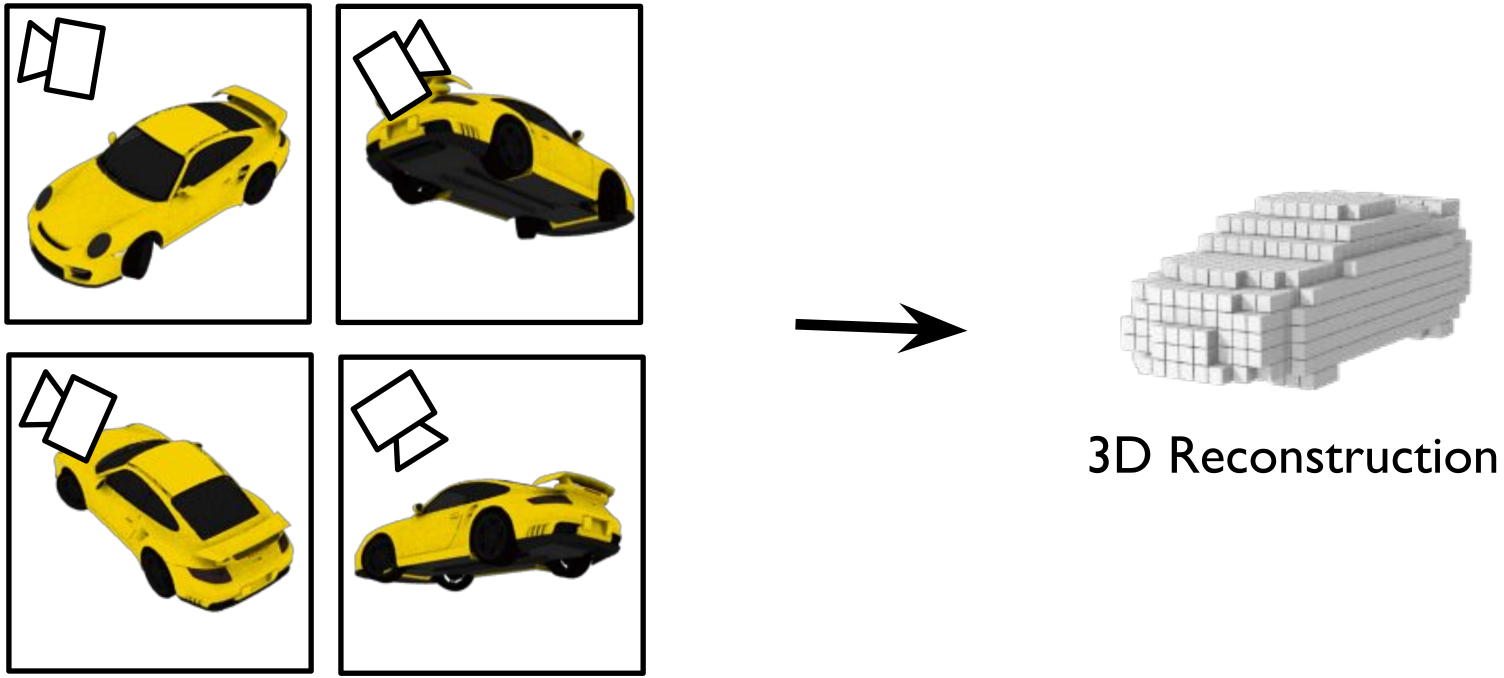
Overview

Two examples of using differentiable geometry in neural networks

- Feature matching in multi-view stereo
  Learning a multi-view stereo machine [Kar, Haene, Malik, NIPS, 2017]

- Supervision through image synthesis
  Learning Independent Object Motion from Unlabelled Stereoscopic Videos
  [Cao, Kar, Haene, Malik, CVPR 2019]

Multi-View Stereo

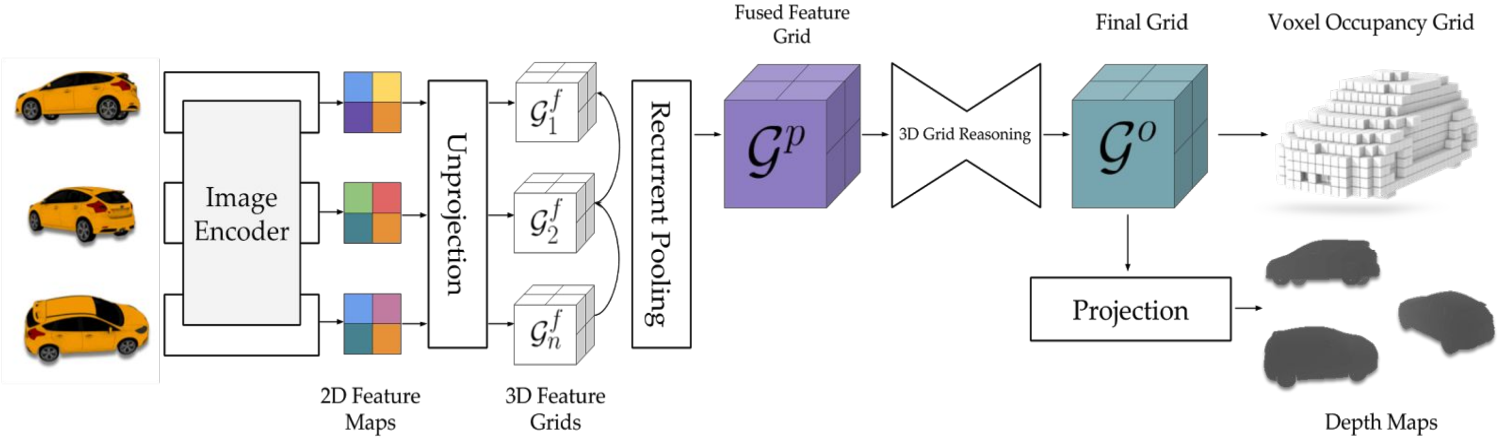


Input: Images and known camera poses

Learnt Stereo Machine (LSM)

https://github.com/akar43/lsm

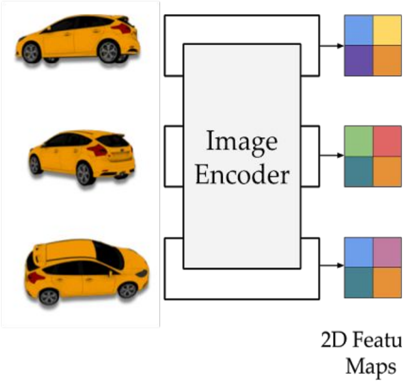
End-to-end learnt multi-view stereo system



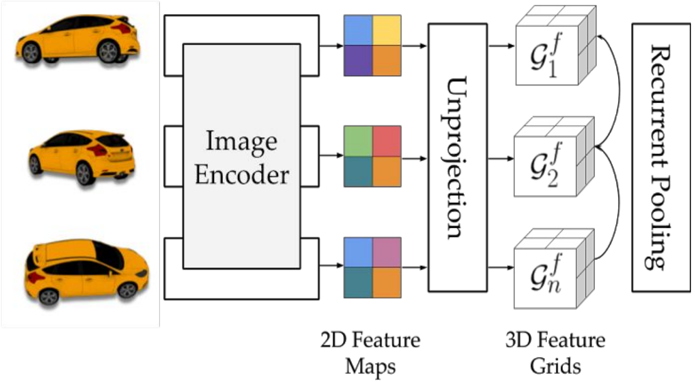
Kar, Haene, Malik, NIPS 2017

- Follows classical (non-learnt) pipeline
- No handcrafted features and priors
- Geometry integrated into the network architecture

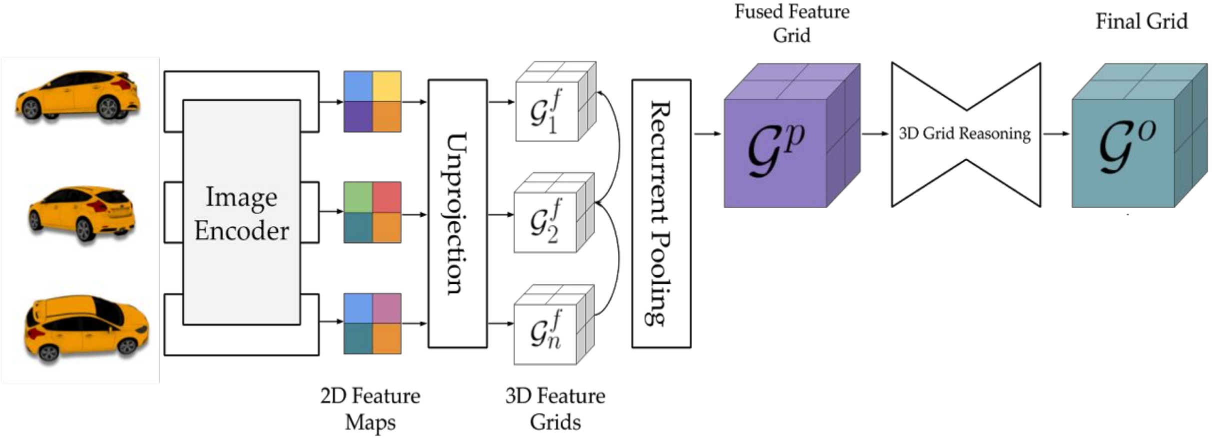
Feature Extraction



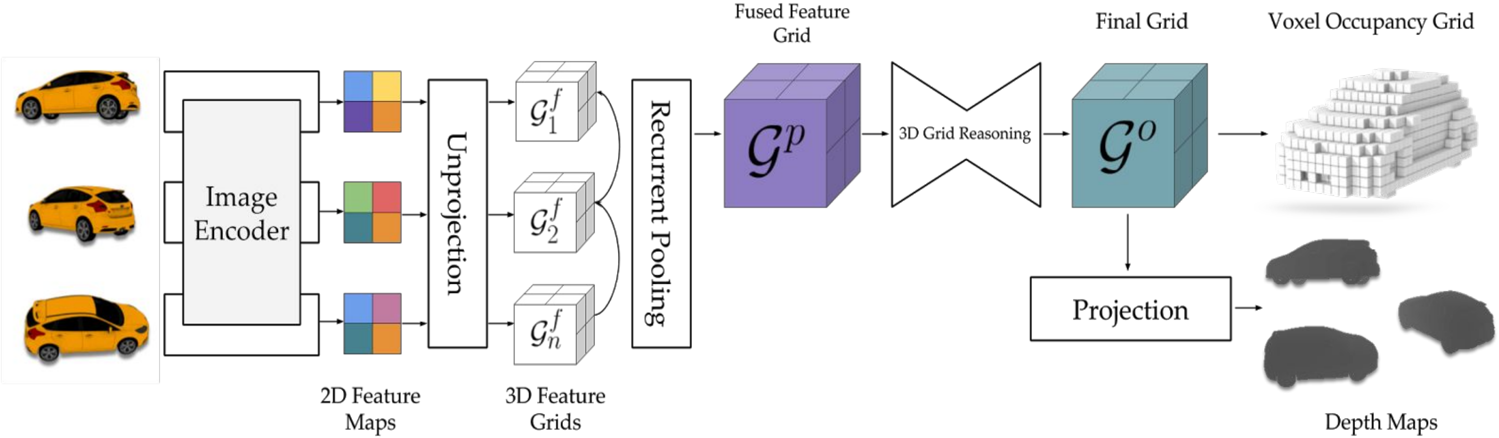
Feature Matching



3D Grid Reasoning

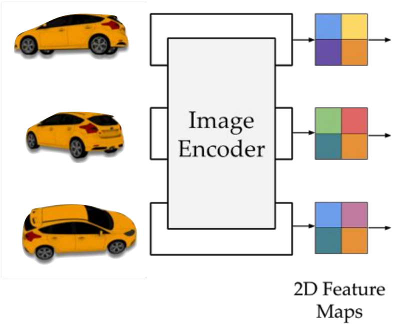


Final Shape Extraction



Kar, Haene, alik, NIPS 2017

Feature Extraction

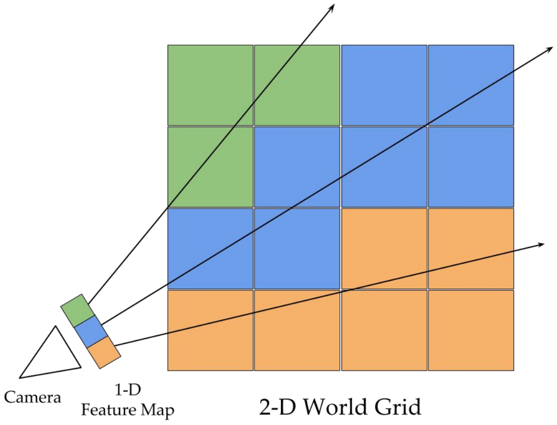


- Image features for matching
- Extracted using CNN

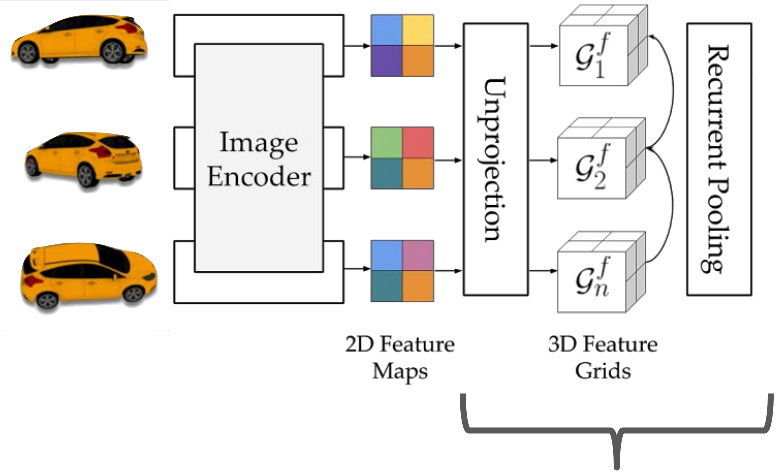
Feature Matching

Two Step Procedure

- Unprojection
- Matching in 3D



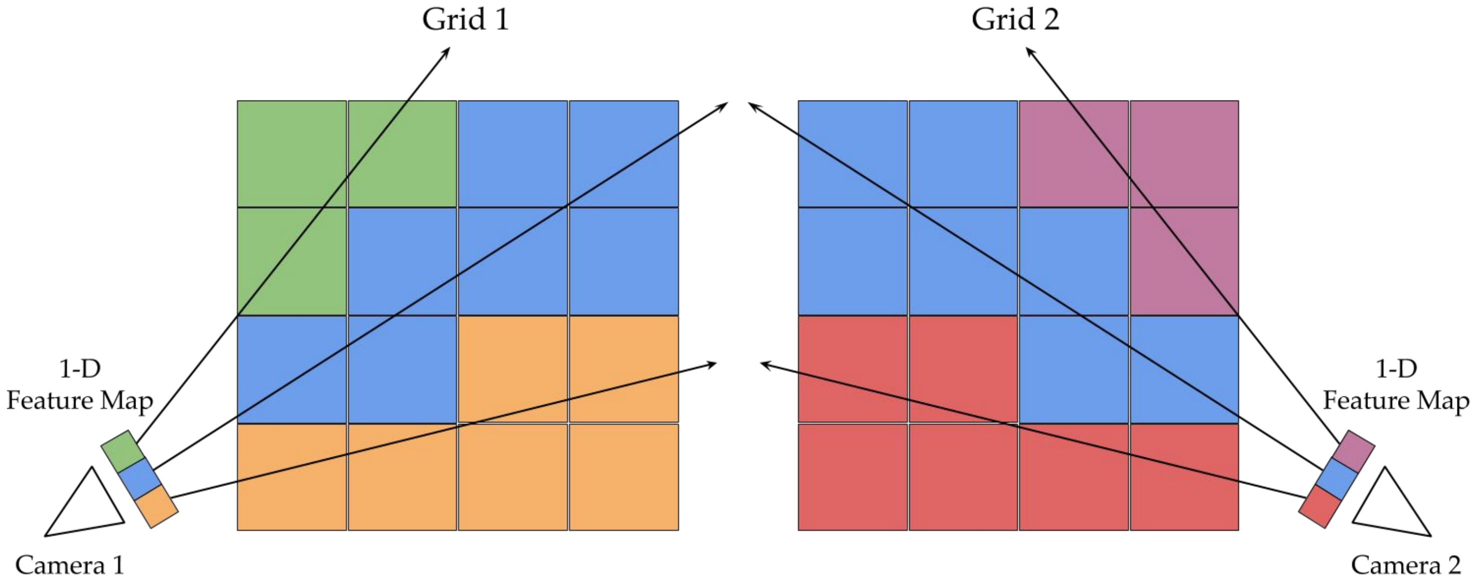
Unprojection



Feature Matching

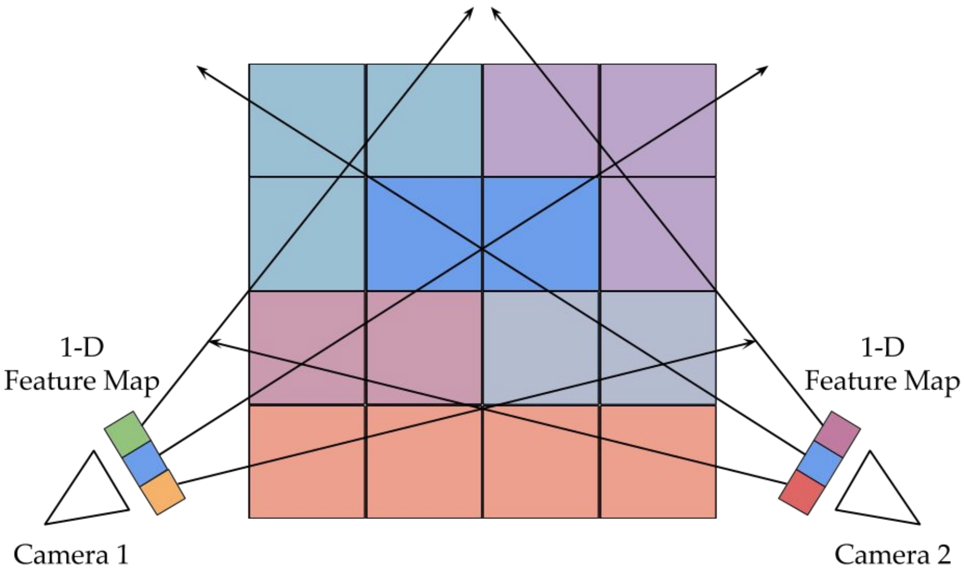
Feature Matching

- After Unprojection features line up.



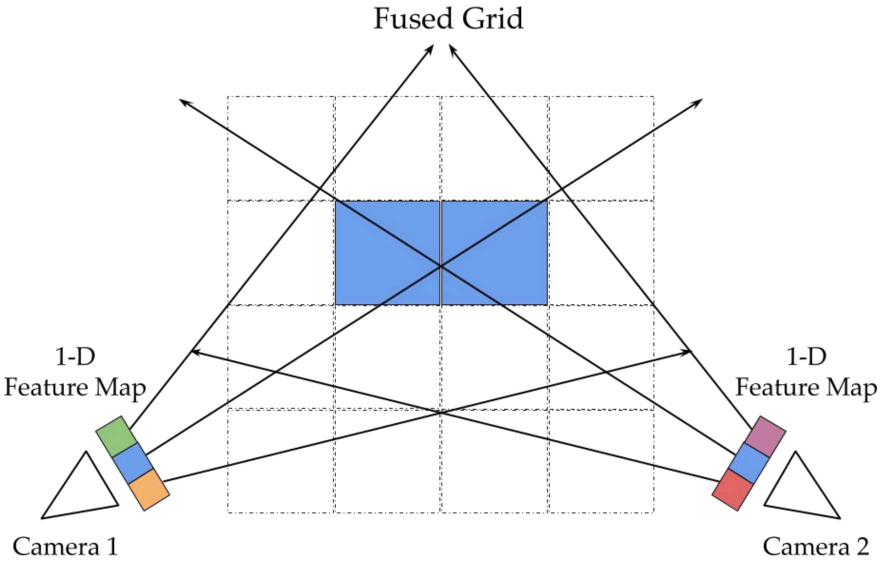
Feature Matching

- After Unprojection features line up



Feature Matching

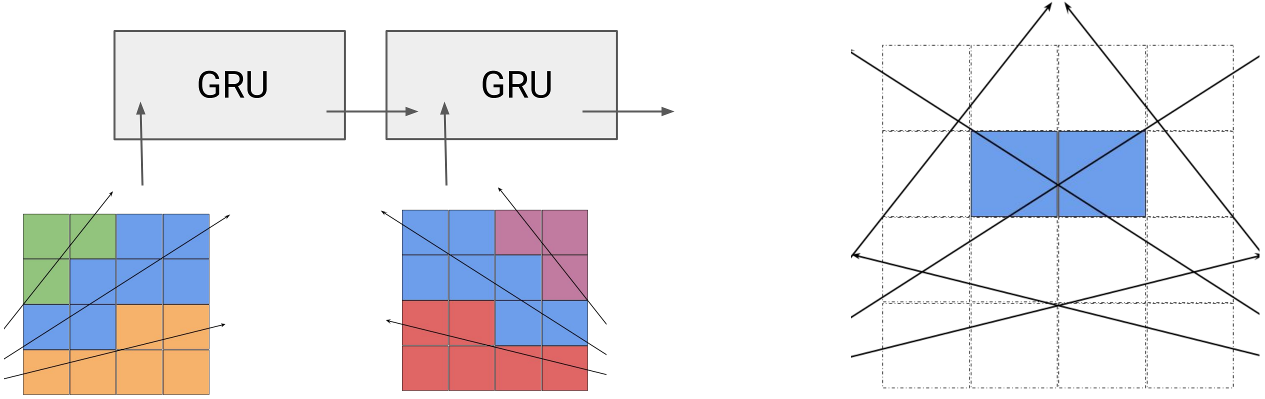
- After Unprojection features line up
- Only local matching in 3D necessary.



Local Matching

3D Convolutional Gated Recurrent Unit (GRU)

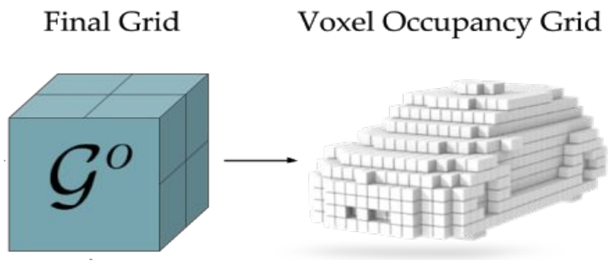
- Frames processed sequentially
- Recurrent cell accumulates information
- Variable number of input images



Fused Grid

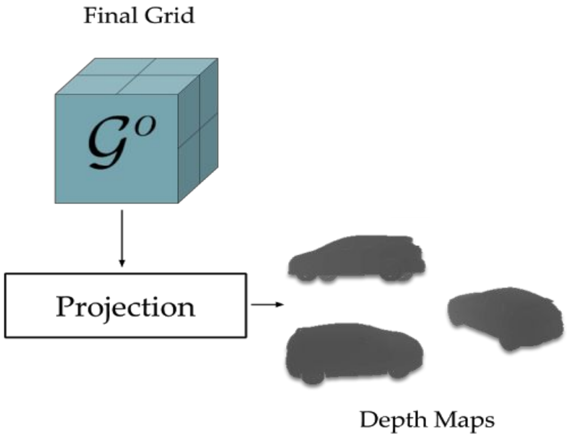
Final Shape Extraction

Voxel LSM



Binary Voxel Occupancy Grid

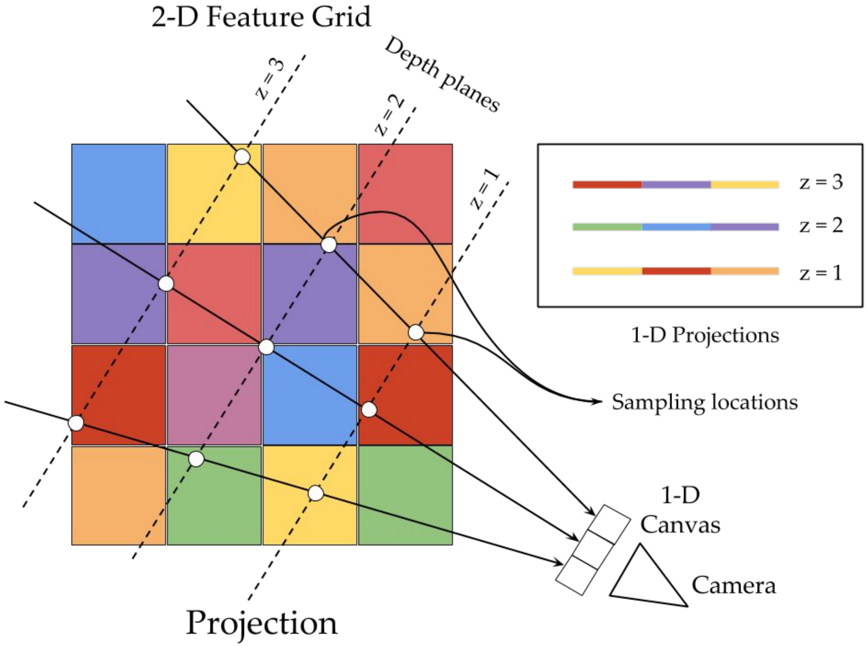
Depth LSM



Depth map for each input view

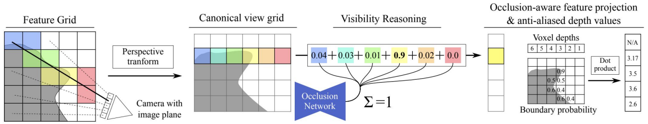
Projection

Same idea used for novel view synthesis



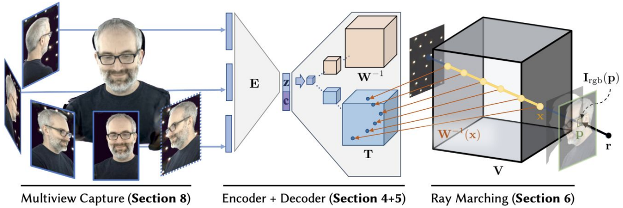
DeepVoxels: Learning Persistent 3D Feature Embeddings

Vincent Sitzmann[1], Justus Thies[2], Felix Heide[3],
Matthias Nießner[2], Gordon Wetzstein[1], Michael Zollhöfer[1]
[1]Stanford University, [2]Technical University of Munich, [3]Princeton University



Neural Volumes: Learning Dynamic Renderable Volumes from Images

STEPHEN LOMBARDI, Facebook Reality Labs
TOMAS SIMON, Facebook Reality Labs
JASON SARAGIH, Facebook Reality Labs
GABRIEL SCHWARTZ, Facebook Reality Labs
ANDREAS LEHRMANN, Facebook Reality Labs
YASER SHEIKH, Facebook Reality Labs



Result

Trained and tested on synthetic ShapeNet dataset

LSM
ours

3D-R2N2
[Choy et al. 2016]
(no geometry layers)

Result

Trained and tested on synthetic ShapeNet dataset

Input Images

LSM
ours

3D-R2N2
[Choy et al. 2016]
(no geometry layers)

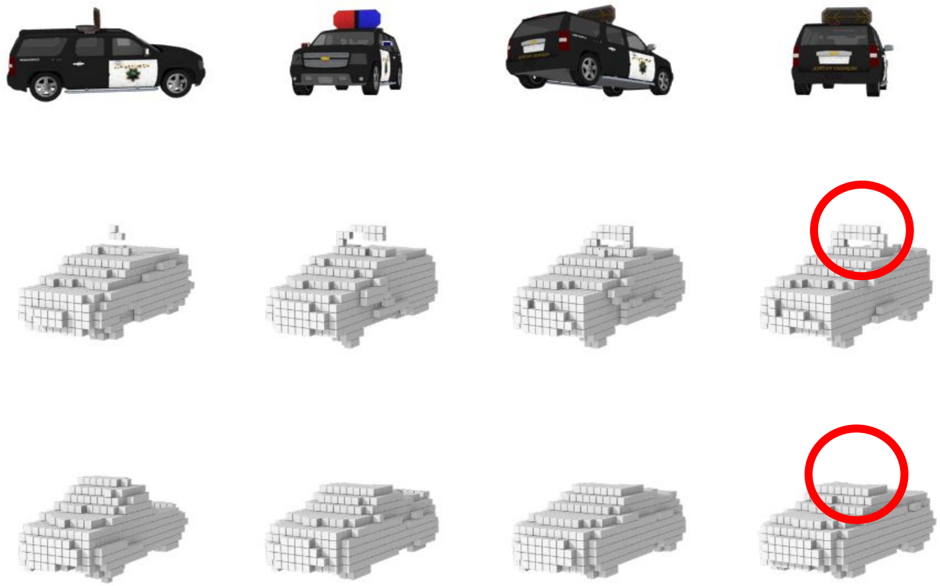
Result

Trained and tested on the synthetic ShapeNet dataset

Input Images

LSM
ours

3D-R2N2
[Choy et al. 2016]
(no geometry layers)



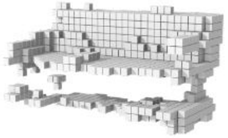
For more details see, ***Learning a multi-view stereo machine***, Kar, Haene, Malik, NIPS 2017

Result

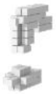
Trained and tested on synthetic ShapeNet dataset

Input Images

LSM
ours



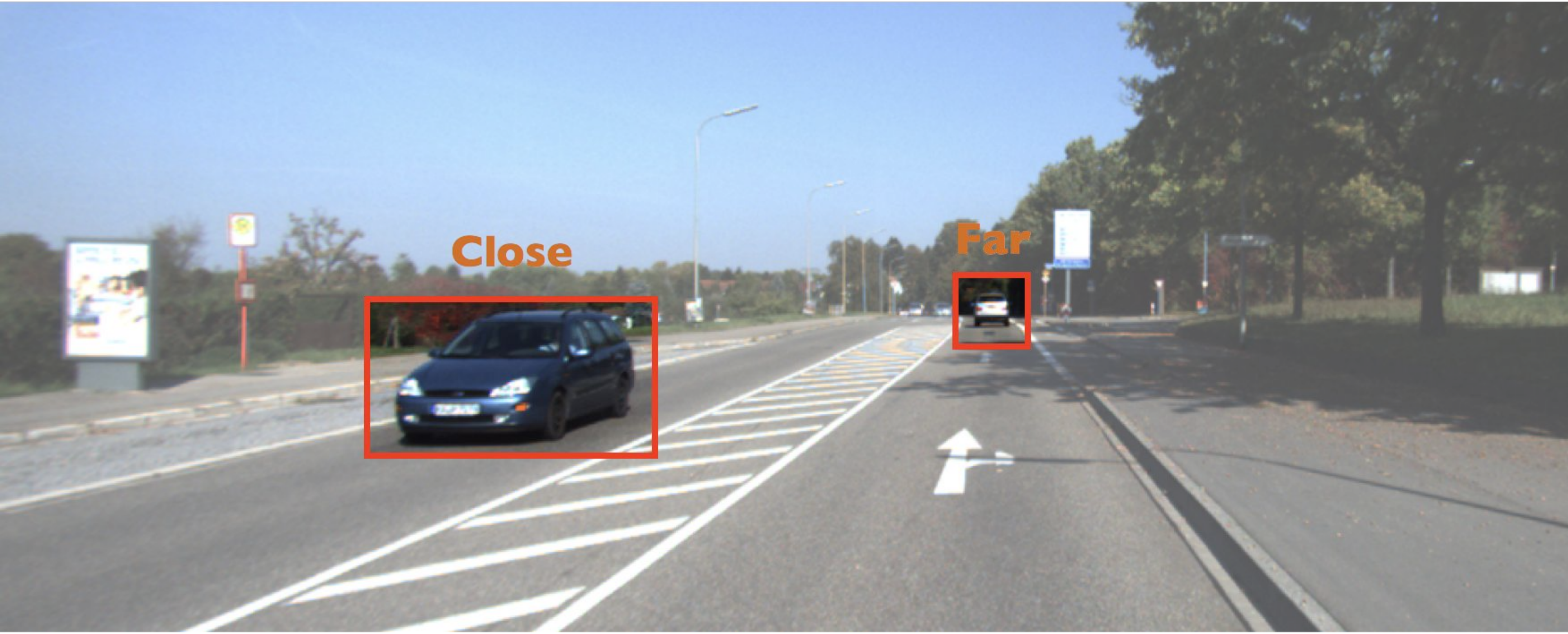
3D-R2N2
[Choy et al. 2016]
(no geometry layers)



For more details see, ***Learning a multi-view stereo machine***, Kar, Haene, Malik, NIPS 2017

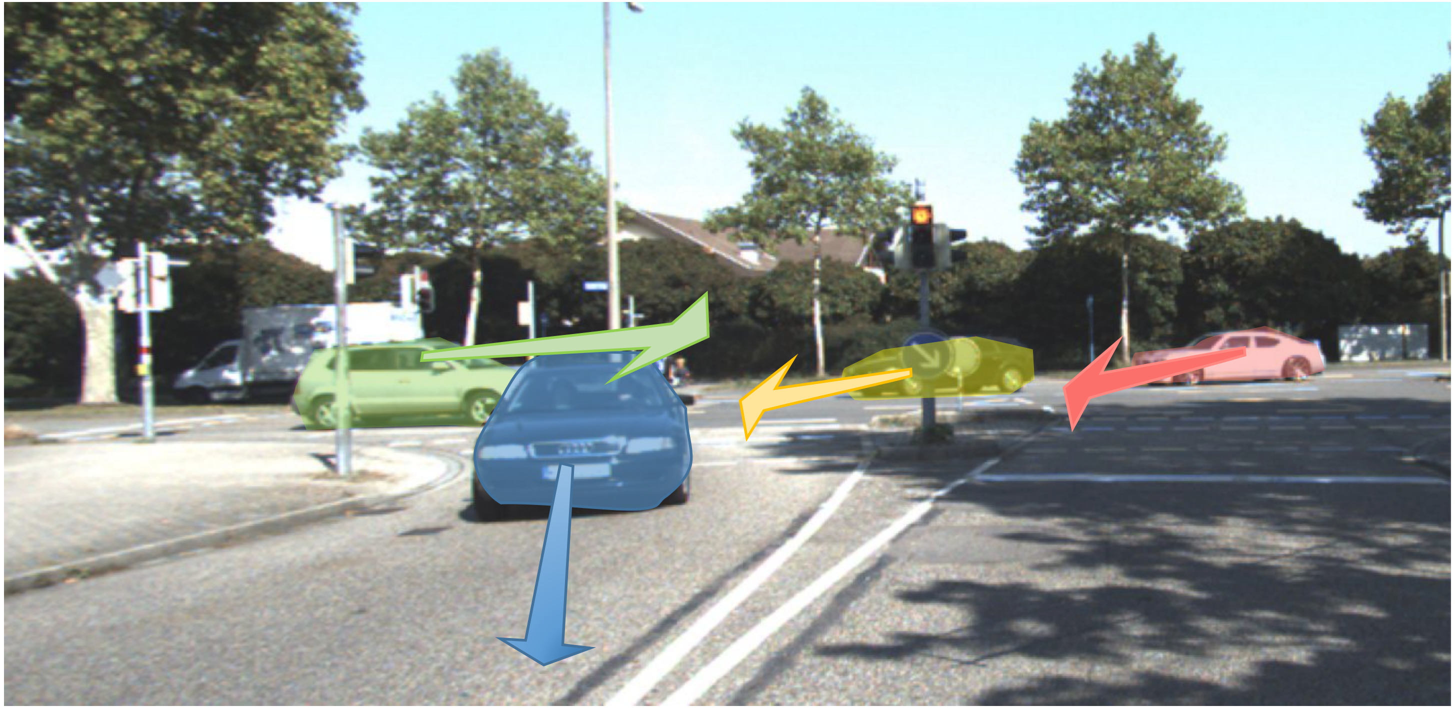
3D Understanding of Road Scenes from 2D Images

3D Understanding of Road Scenes from 2D Images



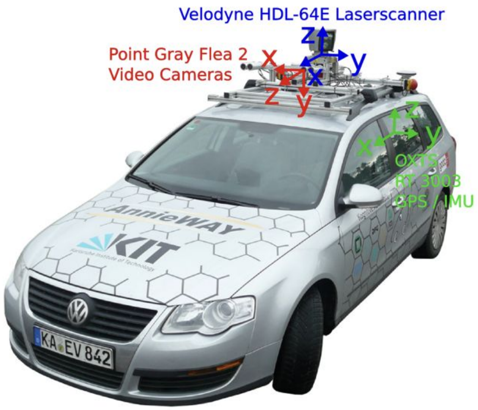
3D Understanding of Road Scenes from 2D Images

Moving Objects



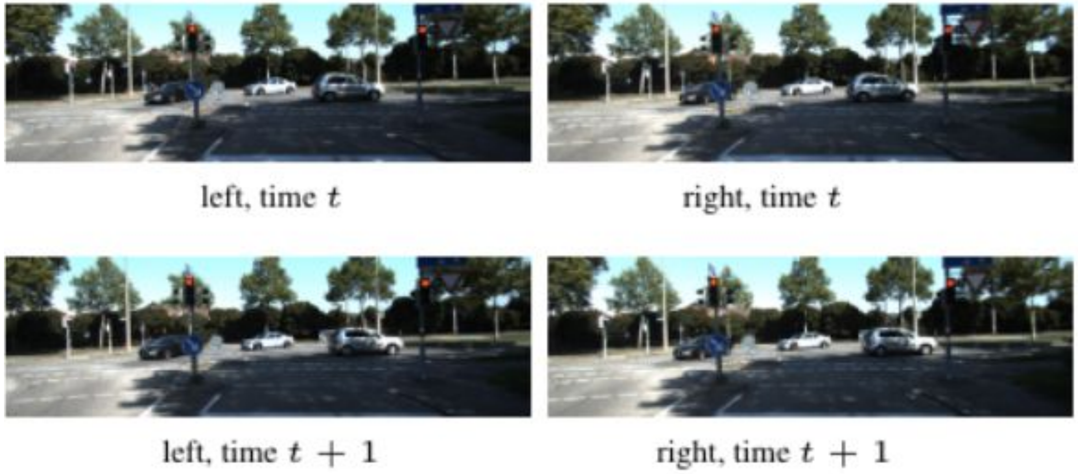
Real-World Ground Truth not Available

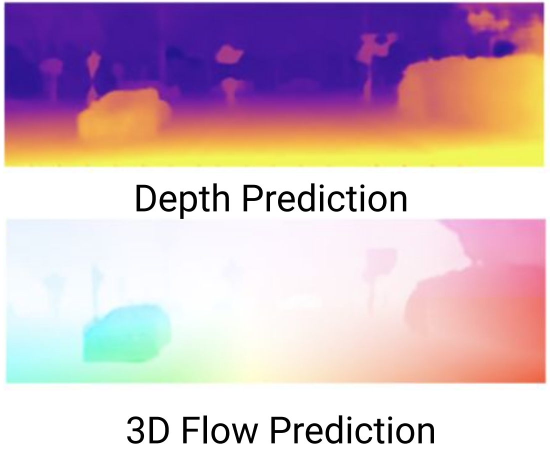
- Expensive capture setup for capturing sparse ground truth depth
- Flow ground truth hand annotated by fitting CAD models



Geiger et al. 2015

Predicting Depth and 3D Flow

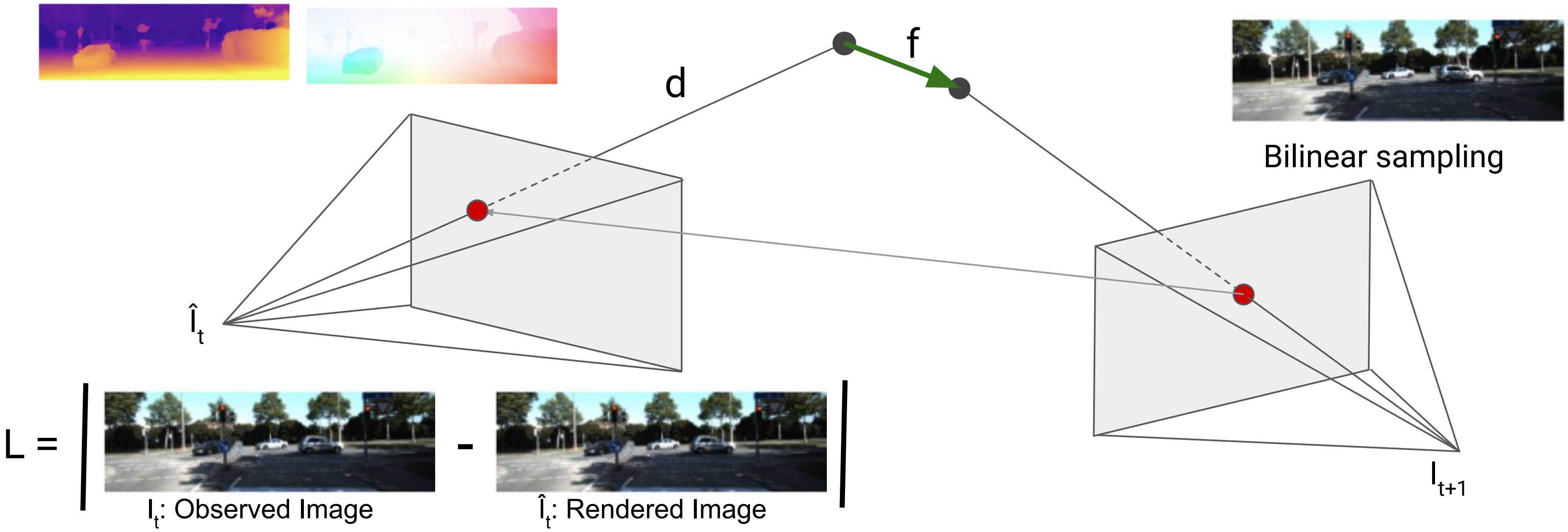




Supervise without ground truth?

Render Novel Images from Predictions

Warping image I_{t+1} to frame t using 3d flow (f) and depth (d) predictions for frame t



Analysis by synthesis: Supervision by comparing rendered image and observed image

Network Architecture

Similar architecture used for view synthesis

Deep View Synthesis from Sparse Photometric Images

ZEXIANG XU, University of California, San Diego
SAI BI, University of California, San Diego
KALYAN SUNKAVALLI, Adobe Research
SUNIL HADAP*, Lab126, Amazon
HAO SU, University of California, San Diego
RAVI RAMAMOORTHI, University of California, San Diego

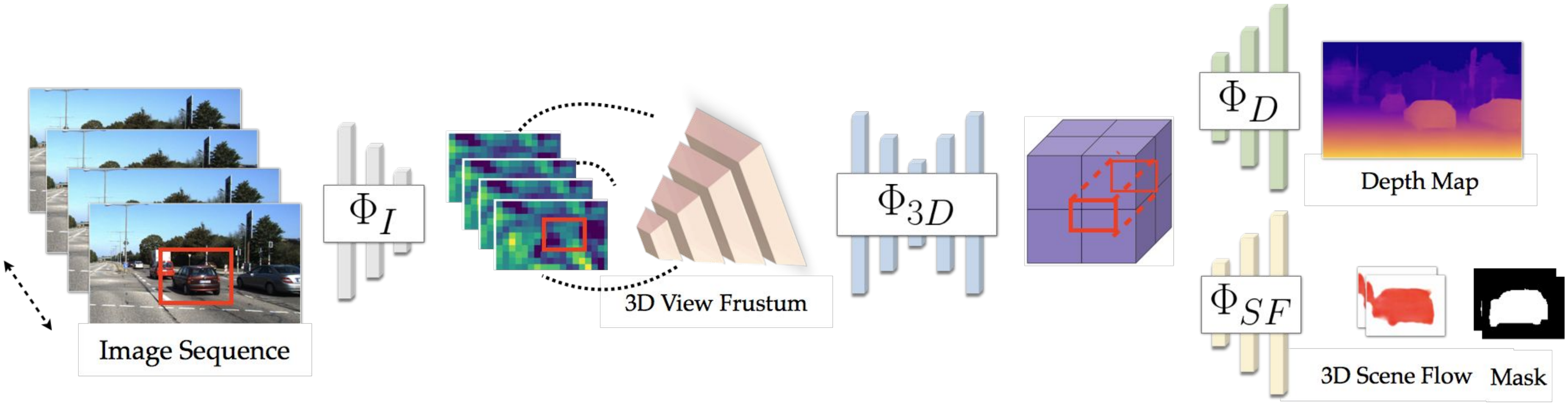


Image Encoding

Feature Unprojection and Pooling

3D Grid Reasoning

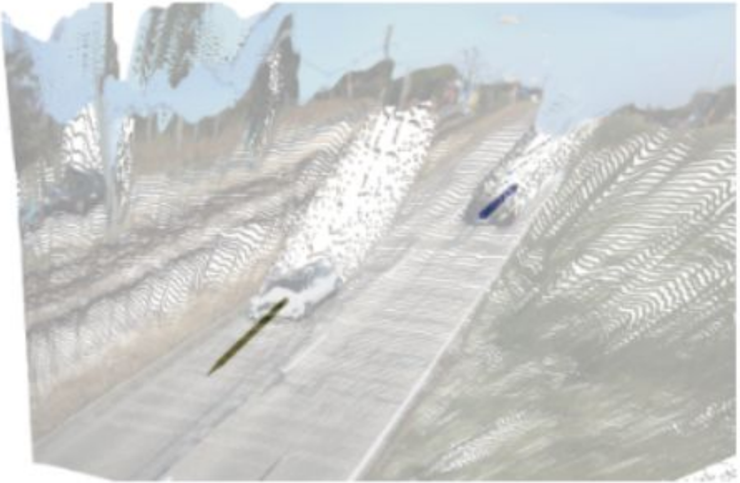
Final Prediction

Results 3D Visualization

left, time t

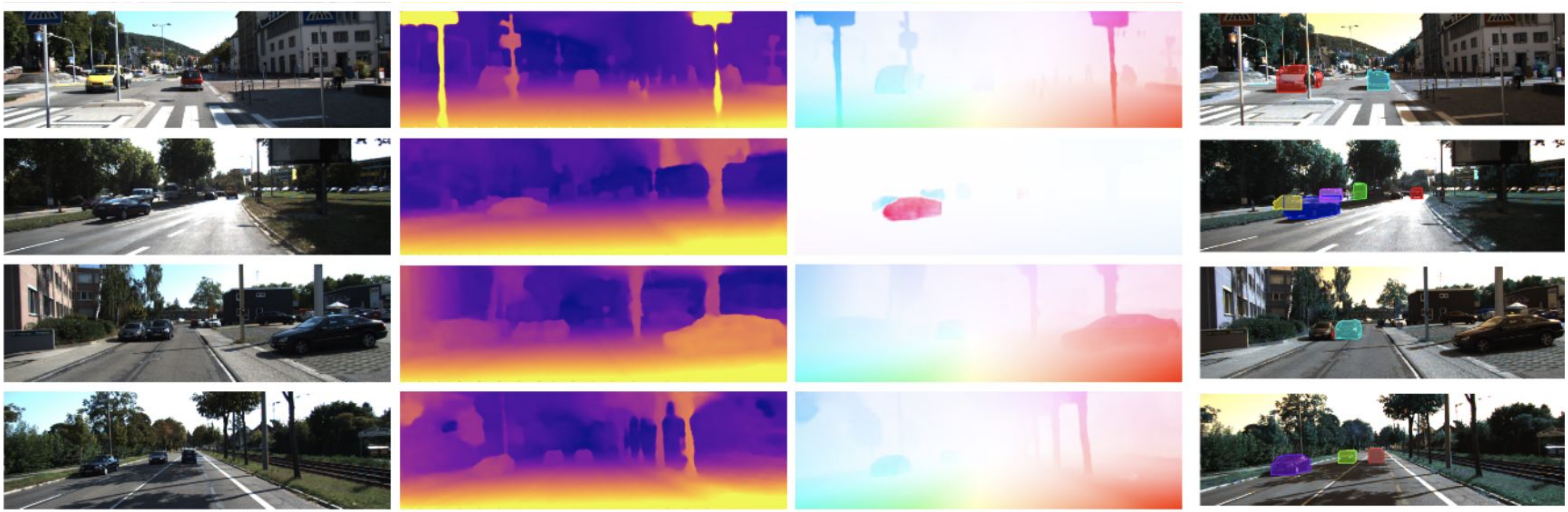
right, time t

3D Point Cloud



3D Flow Vectors (Average per Object)

Results 2D Visualization



Reference Image | Predicted Depth | Predicted Flow | Predicted Object Masks

For more details see, ***Learning Independent Object Motion from Unlabelled Stereoscopic Videos***, Cao, Kar, Haene, Malik, CVPR 2019

Summary

3D Geometry is important for learning 3D computer vision tasks.

- As differentiable modules within the network architecture
  - Camera model
  - Occupancy Grid / 3D convolution
- As supervisory signal via view synthesis
  (analysis by synthesis)

TensorFlow

Q&A

Julien Valentin (@JPCValentin)

Christian Häne

TensorFlow

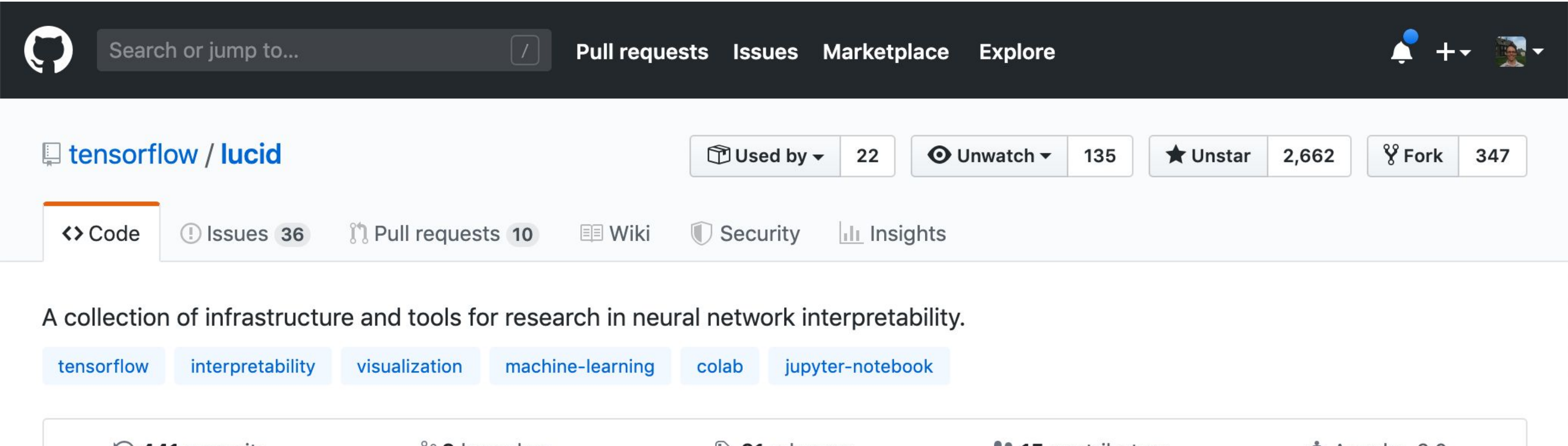
Visualization and Interpretation of Neural Networks

Shan Carter

OpenAI

TensorFlow

Lucid
github.com/tensorflow/lucid



Chris Olah OpenAI
Ludwig Schubert OpenAI
Alexander Mordvintsev Google
Arvind Satyanarayan MIT
Zan Armstrong Google
Ian Johnson Google

“Why can’t TensorFlow see my GPU?”

“Why c sorFlow
see PU?”

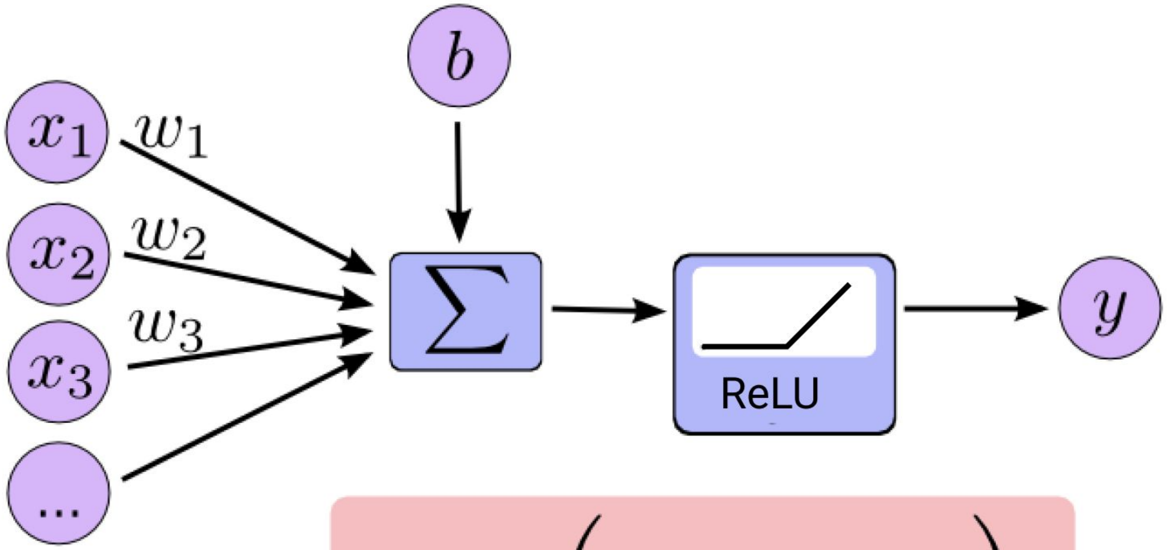
“dingo”

“What has my network learned?”


Inception V1
“dingo”

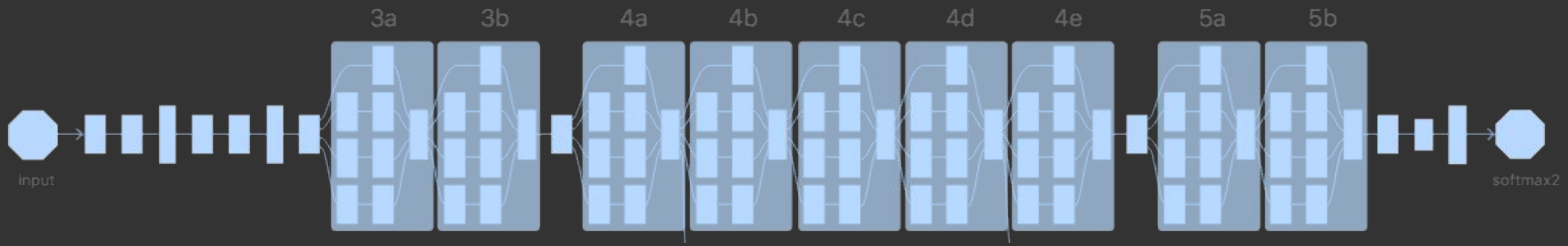
Inception V1

LABEL	PROBABILITY
dingo	40.51%
Ibizan hound	33.03%
Siberian husky	8.22%
Eskimo dog	6.69%
kelpie	5.45%
Cardigan	1.92%
malamute	0.99%
German shepherd	0.88%
Pembroke	0.87%
white wolf	0.62%
timber wolf	0.21%
Chihuahua	0.07%
collie	0.06%
basenji	0.05%
coyote	0.04%
Labrador retriever	**0.04%**
tennis ball	0.03%
kuvasz	0.02%
Border collie	0.02%

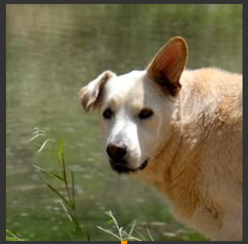


$$y = \mathrm{ReLU}\left(\sum_i w_i x_i + b\right)$$

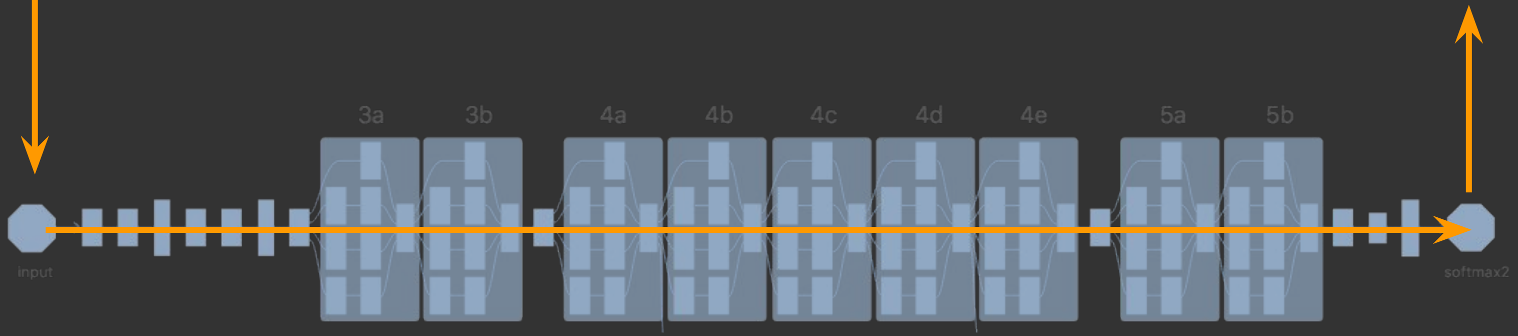
```python
end_point = 'Conv2d_1a_7x7'
net = layers.conv2d(inputs, 64, [7, 7], stride=2, scope=end_point)
end_points[end_point] = net
if final_endpoint == end_point:
  return net, end_points
end_point = 'MaxPool_2a_3x3'
net = layers_lib.max_pool2d(net, [3, 3], stride=2, scope=end_point)
end_points[end_point] = net
if final_endpoint == end_point:
  return net, end_points
end_point = 'Conv2d_2b_1x1'
net = layers.conv2d(net, 64, [1, 1], scope=end_point)
end_points[end_point] = net
if final_endpoint == end_point:
  return net, end_points
```



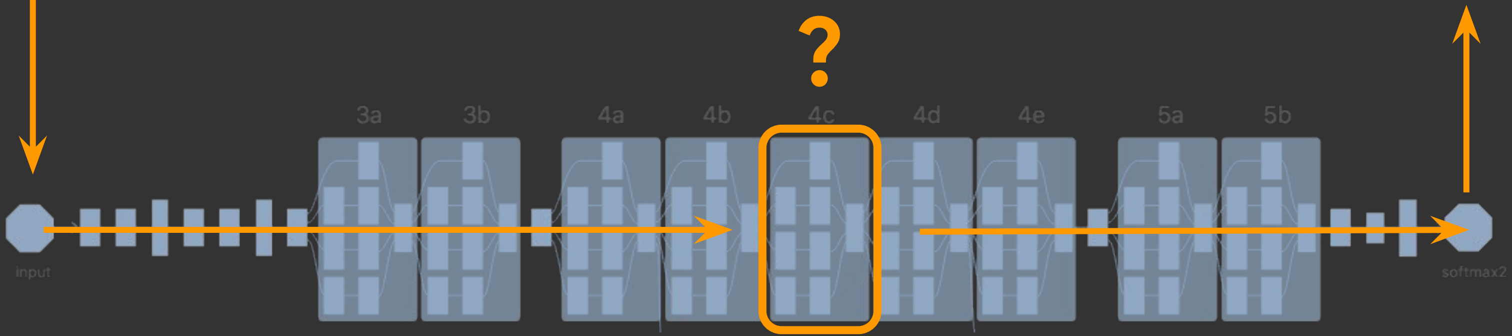
3a
3b
4a
4b
4c
4d
4e
5a
5b
input
softmax2



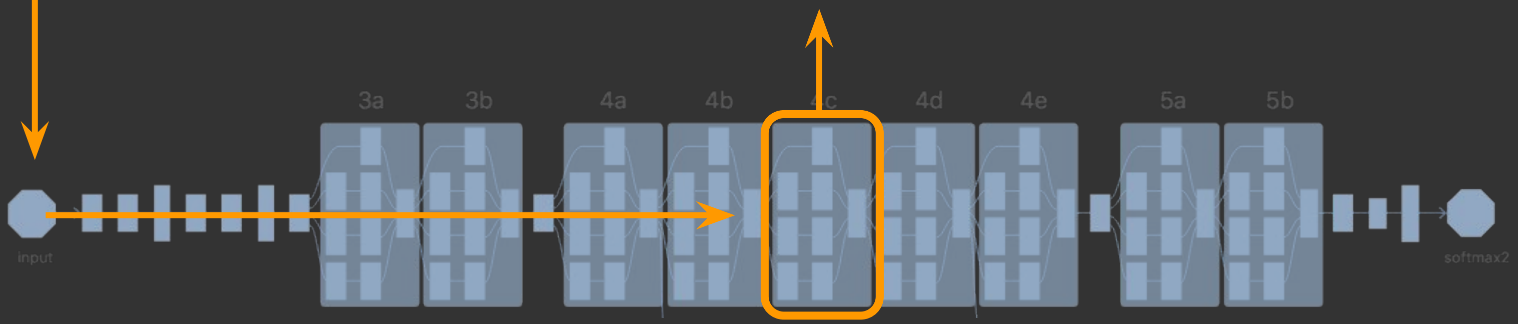
LABEL	PROBABILITY
dingo	40.51%
Ibizan hound	33.03%
Siberian husky	8.22%
Labrador retriever	0.04%



LABEL	PROBABILITY
dingo	40.51%
Ibizan hound	33.03%
Siberian husky	8.22%
Eskimo dog	6.69%



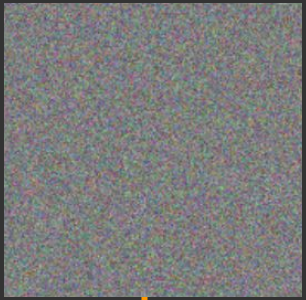
Neuron 1:	0.40439647176792093
Neuron 2:	-0.8331816413002056
Neuron 3:	0.6005791021121634
Neuron 4:	-0.11789122711380684
...	...
Neuron 512:	0.05040674324147387



How can we hope to reason about a 512 dimensional vector?

Neuron 1:	0.40439647176792093
Neuron 2:	-0.8331816413002056
Neuron 3:	0.6005791021121634
Neuron 4:	-0.11789122711380684
...	...
Neuron 512:	0.05040674324147387

IMAGE Step 0



ACTIVATION

Neuron 1:	0.404396
Neuron 2:	0.833181
Neuron 3:	0.600579
Neuron 4:	0.117891
...	...
Neuron 512:	0.050406

TARGET ACTIVATION

Neuron 1:	1.000000
Neuron 2:	0.000000
Neuron 3:	0.000000
Neuron 4:	0.000000
...	...
Neuron 512:	0.000000

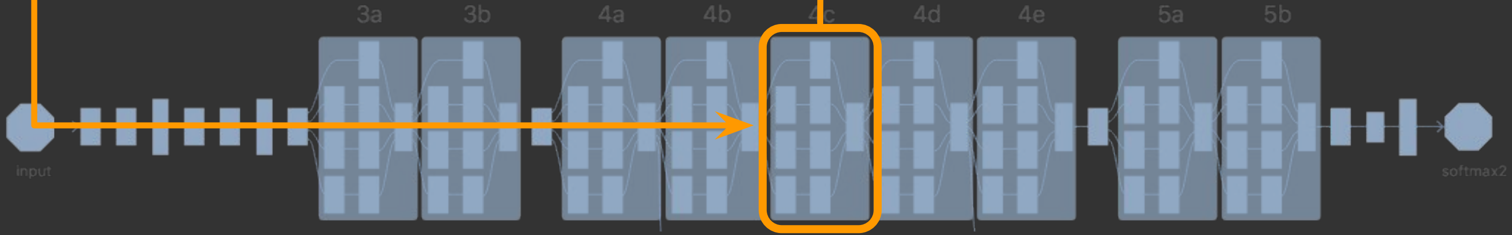


IMAGE Step 4

ACTIVATION

Neuron 1:	0.817395
Neuron 2:	0.456190
Neuron 3:	0.108737
Neuron 4:	0.094583
...	...
Neuron 512:	0.098432

TARGET ACTIVATION

Neuron 1:	1.000000
Neuron 2:	0.000000
Neuron 3:	0.000000
Neuron 4:	0.000000
...	...
Neuron 512:	0.000000

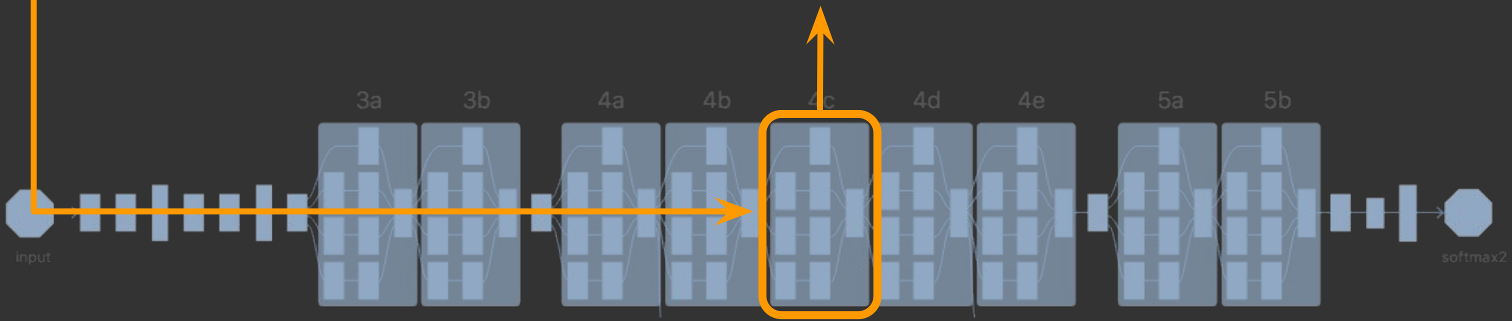
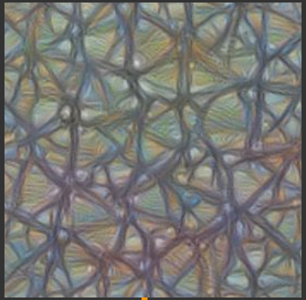


IMAGE Step 48



ACTIVATION

Neuron 1:	0.834368
Neuron 2:	0.021434
Neuron 3:	0.056354
Neuron 4:	0.043551
...	...
Neuron 512:	0.000234

TARGET ACTIVATION

Neuron 1:	1.000000
Neuron 2:	0.000000
Neuron 3:	0.000000
Neuron 4:	0.000000
...	...
Neuron 512:	0.000000

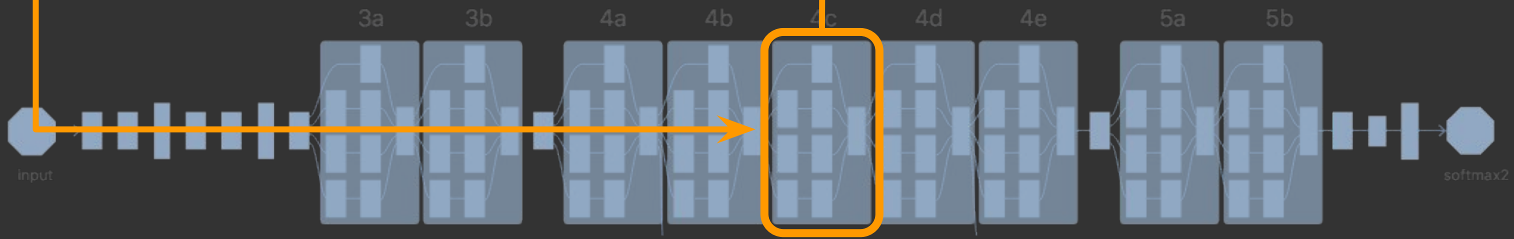
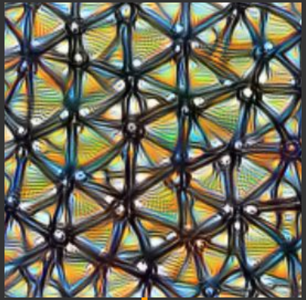


IMAGE Step 2048

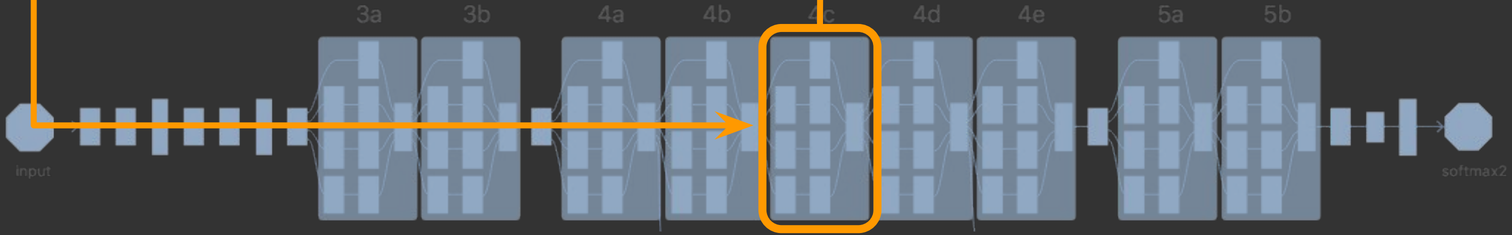


ACTIVATION

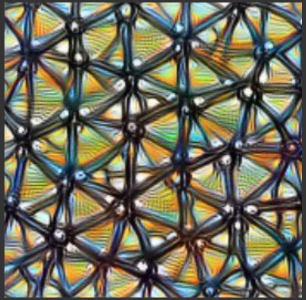
Neuron 1:	10.01340
Neuron 2:	0.045623
Neuron 3:	0.001034
Neuron 4:	0.009874
...	...
Neuron 512:	0.013490

TARGET ACTIVATION

Neuron 1:	1.000000
Neuron 2:	0.000000
Neuron 3:	0.000000
Neuron 4:	0.000000
...	...
Neuron 512:	0.000000



IMAGE



TARGET ACTIVATION

Neuron 1:	1.000000
Neuron 2:	0.000000
Neuron 3:	0.000000
Neuron 4:	0.000000
...	...
Neuron 512:	0.000000

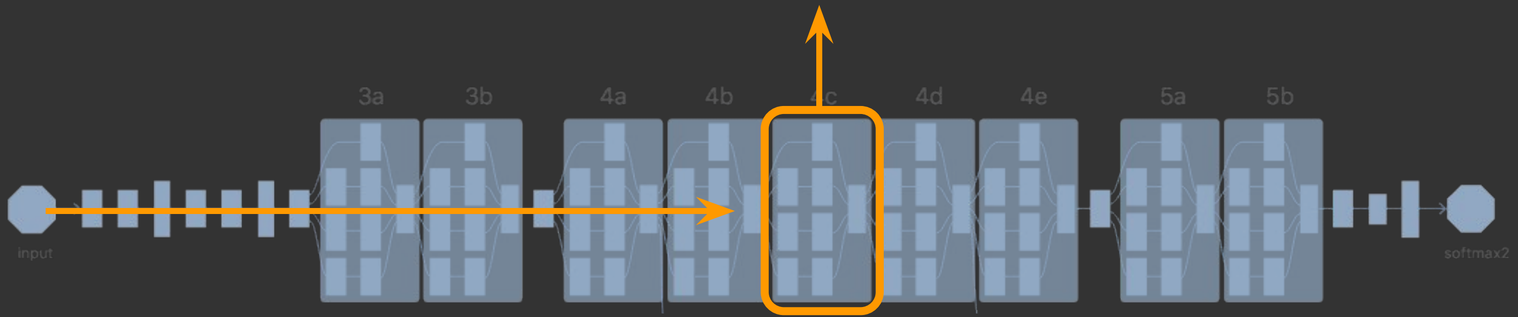
Feature visualization https://distill.pub/2017/feature-visualization/

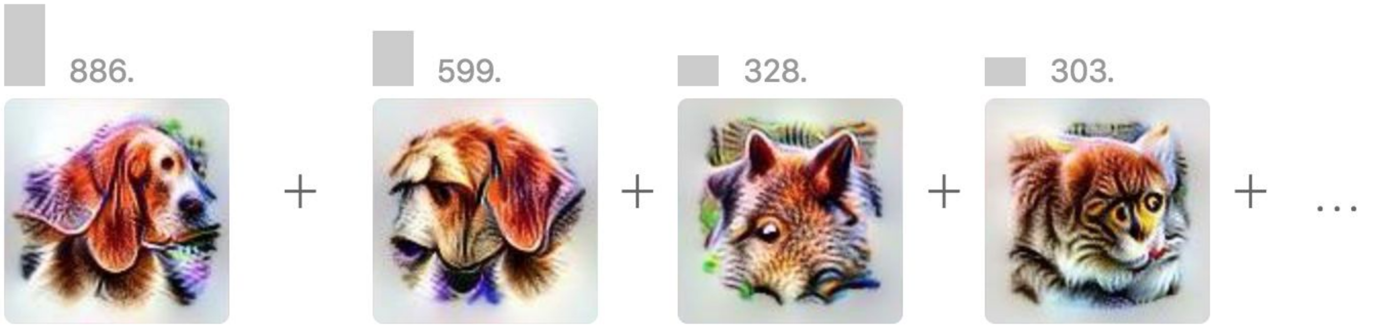
Neuron 1: 0.40439647176792093

Neuron 2: 0.8331816413002056

Neuron 3: 0.6005791021121634

... ...

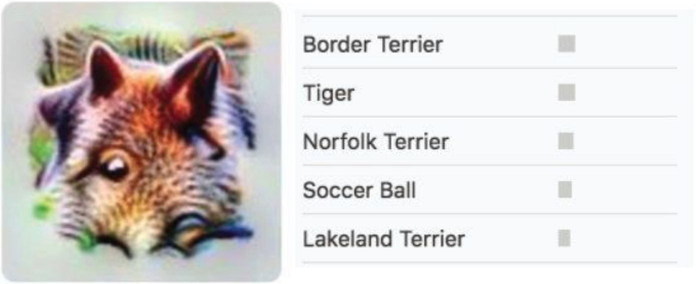


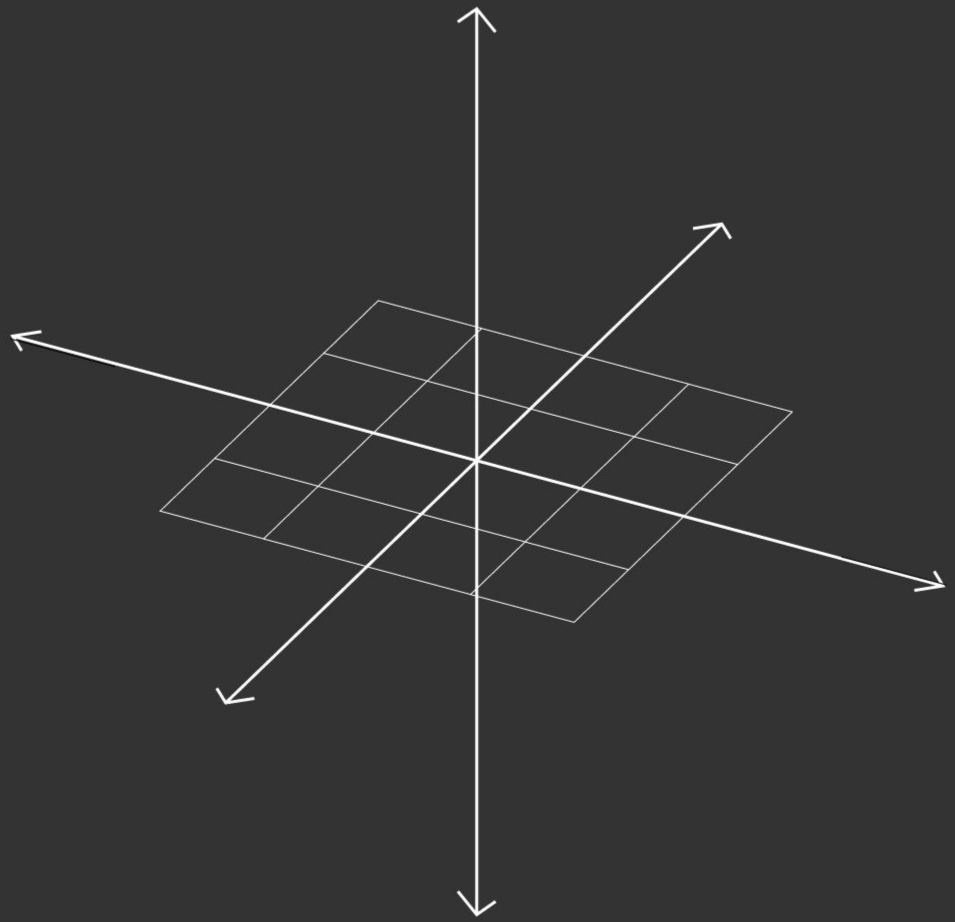


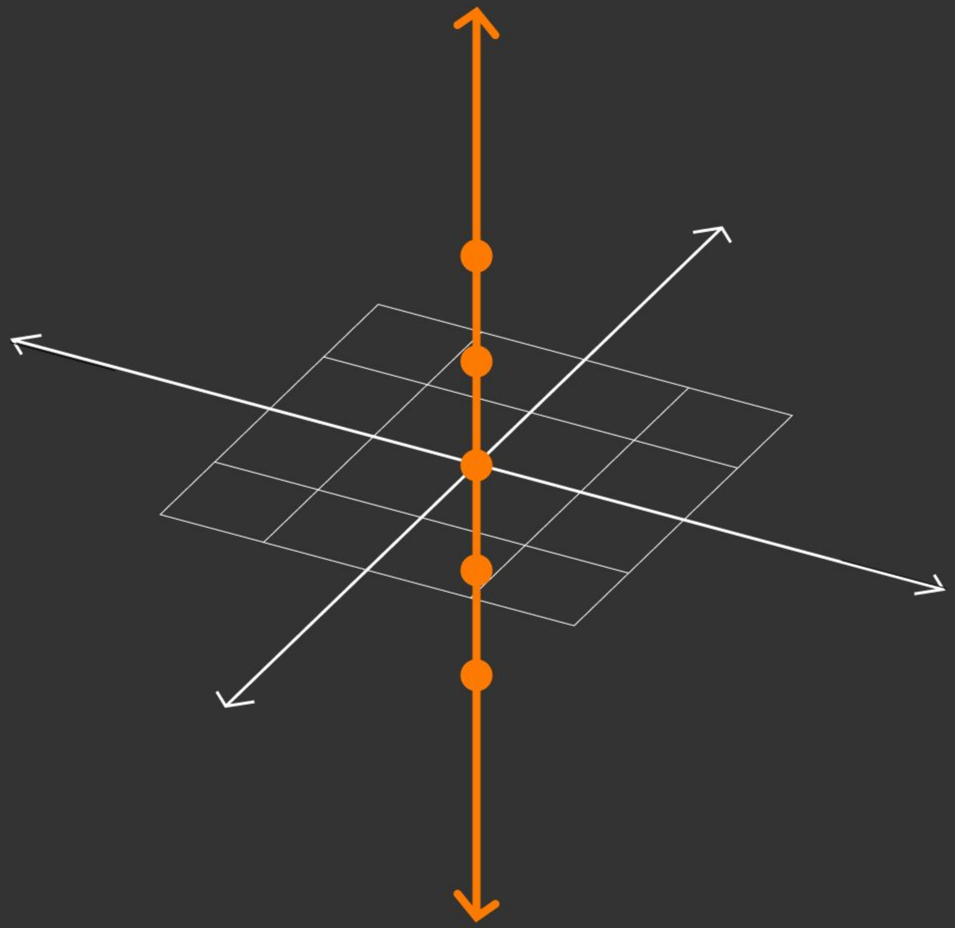
Channels

Ear detectors seem to help distinguish between animals

Note that the effect of each detector is sensitive to how much it fires, which other features fire, etc.





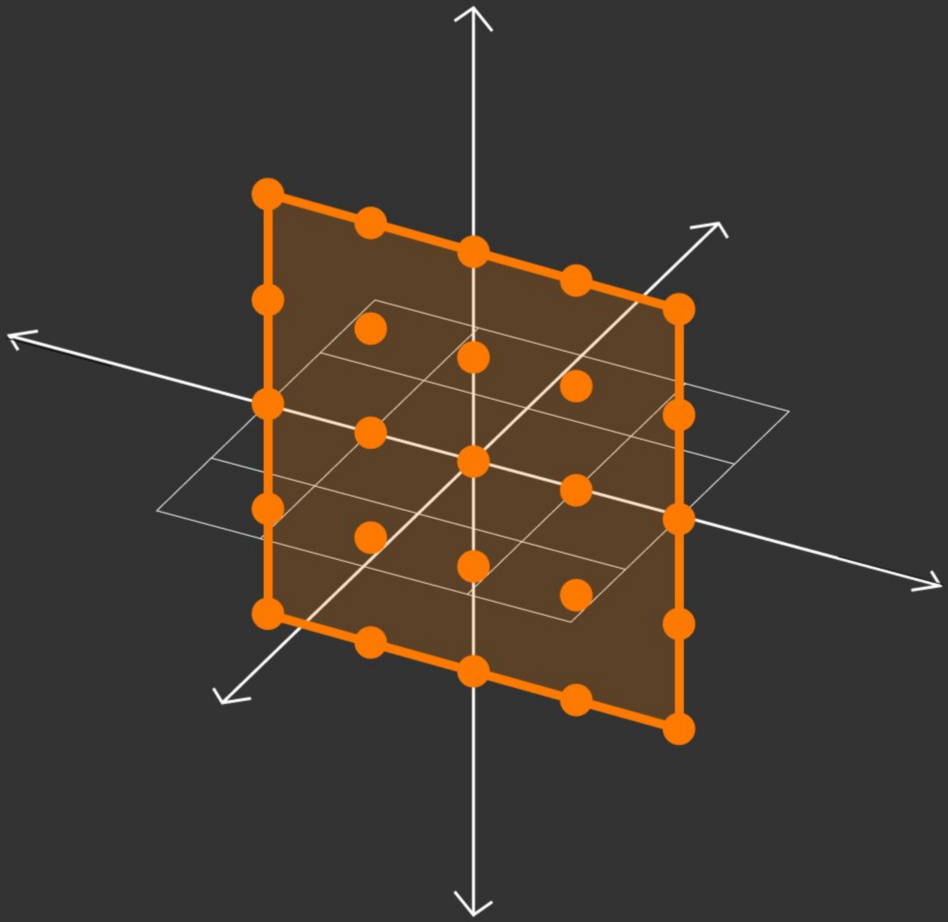


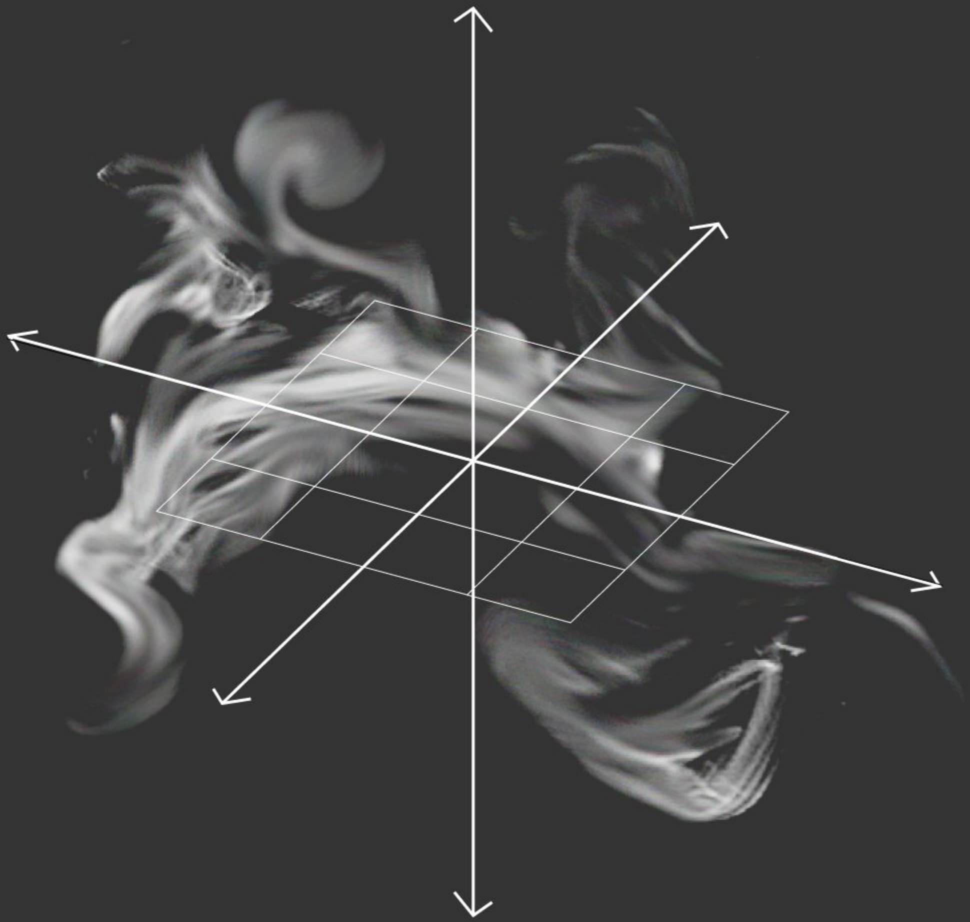
How neurons interact

https://distill.pub/2017/feature-visualization/#interaction

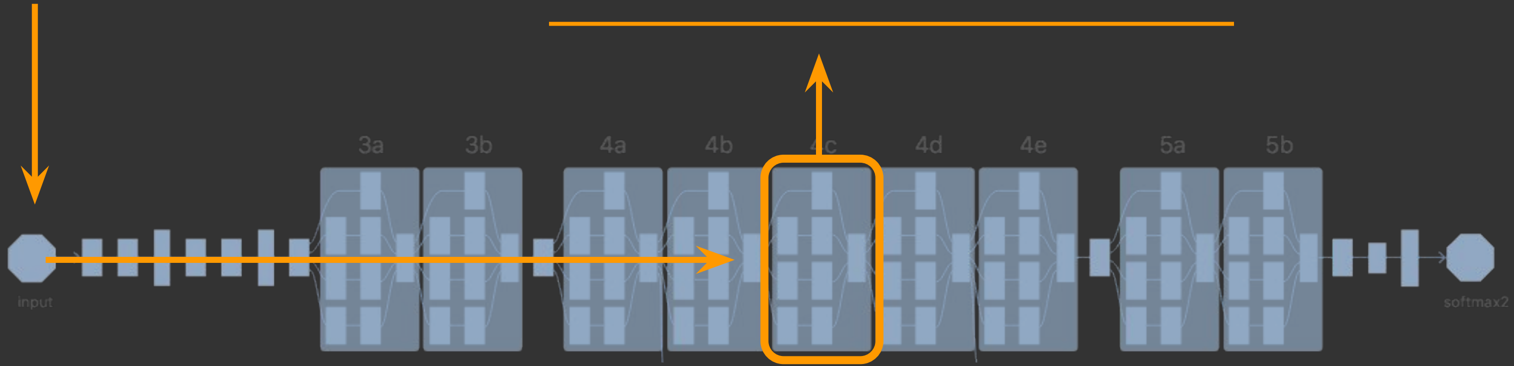
How neurons interact

https://distill.pub/2017/feature-visualization/#interaction

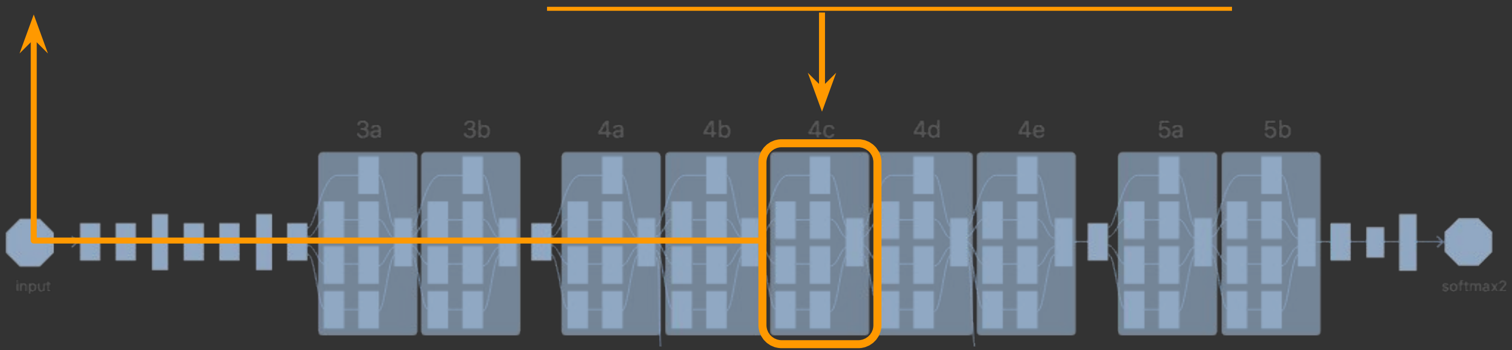


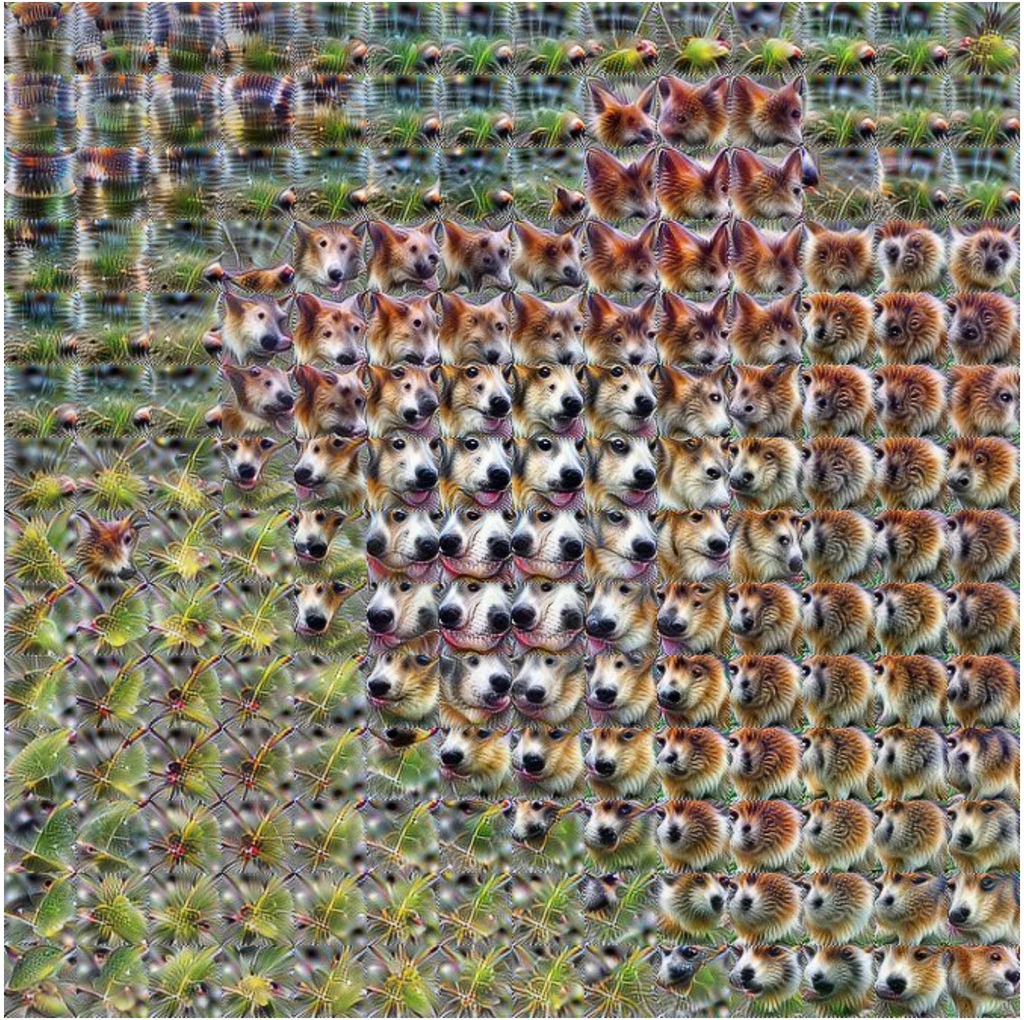


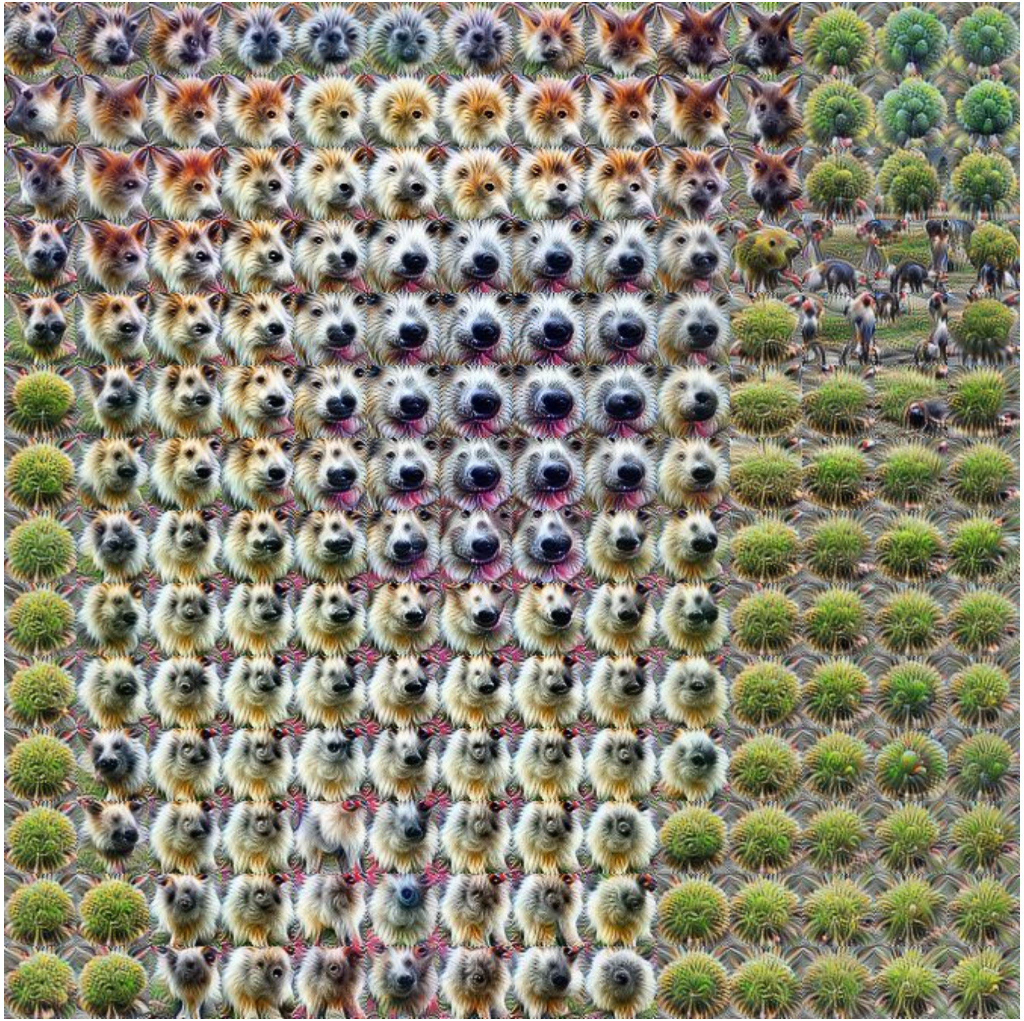
Neuron 1:	0.40439647176792093
Neuron 2:	-0.8331816413002056
Neuron 3:	0.6005791021121634
Neuron 4:	-0.11789122711380684
...	...
Neuron 512:	0.05040674324147387



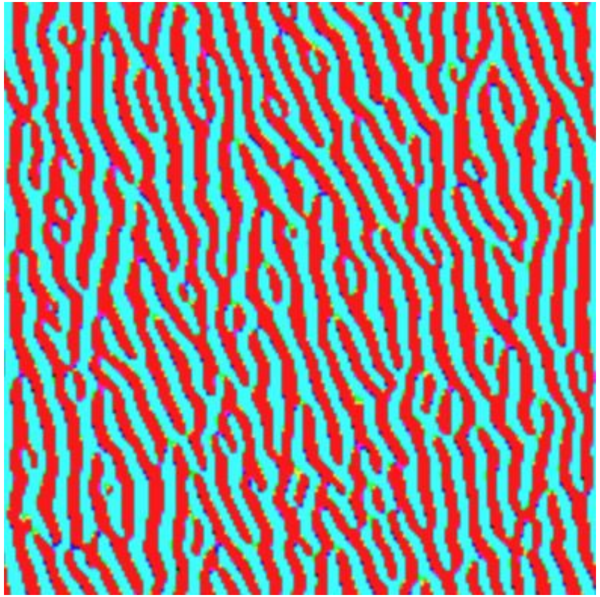
Neuron 1:	8.40439647176792093
Neuron 2:	-16.8331816413002056
Neuron 3:	12.6005791021121634
Neuron 4:	-2.11789122711380684
...	...
Neuron 512:	0.05040674324147387



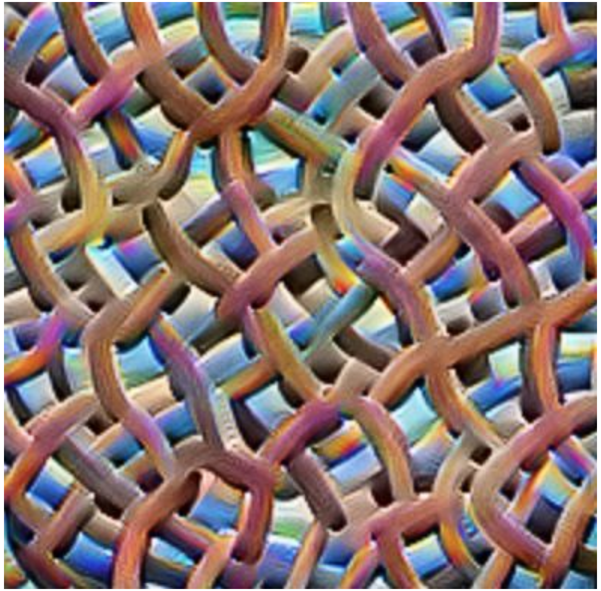



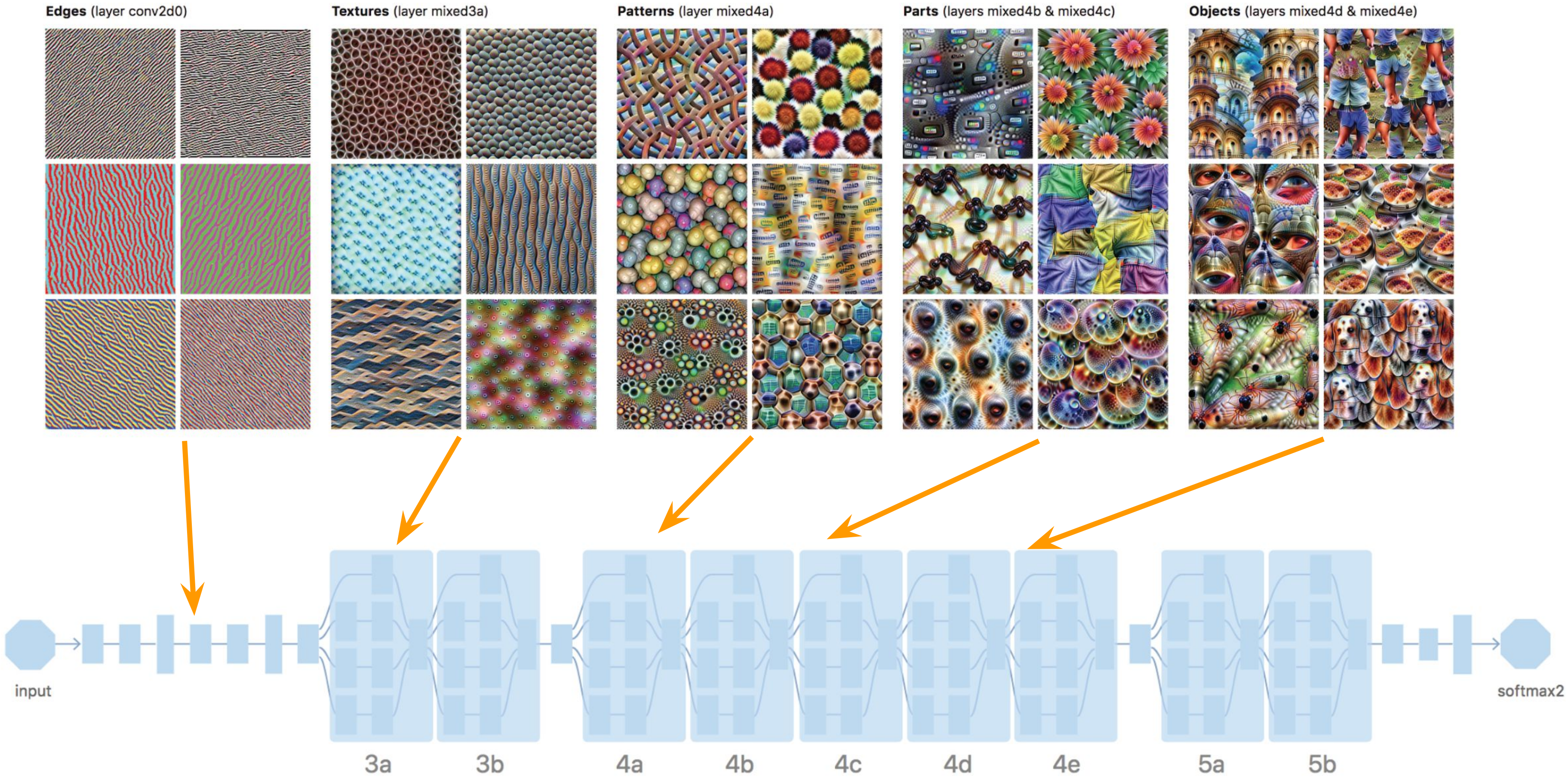
Beginning



Middle

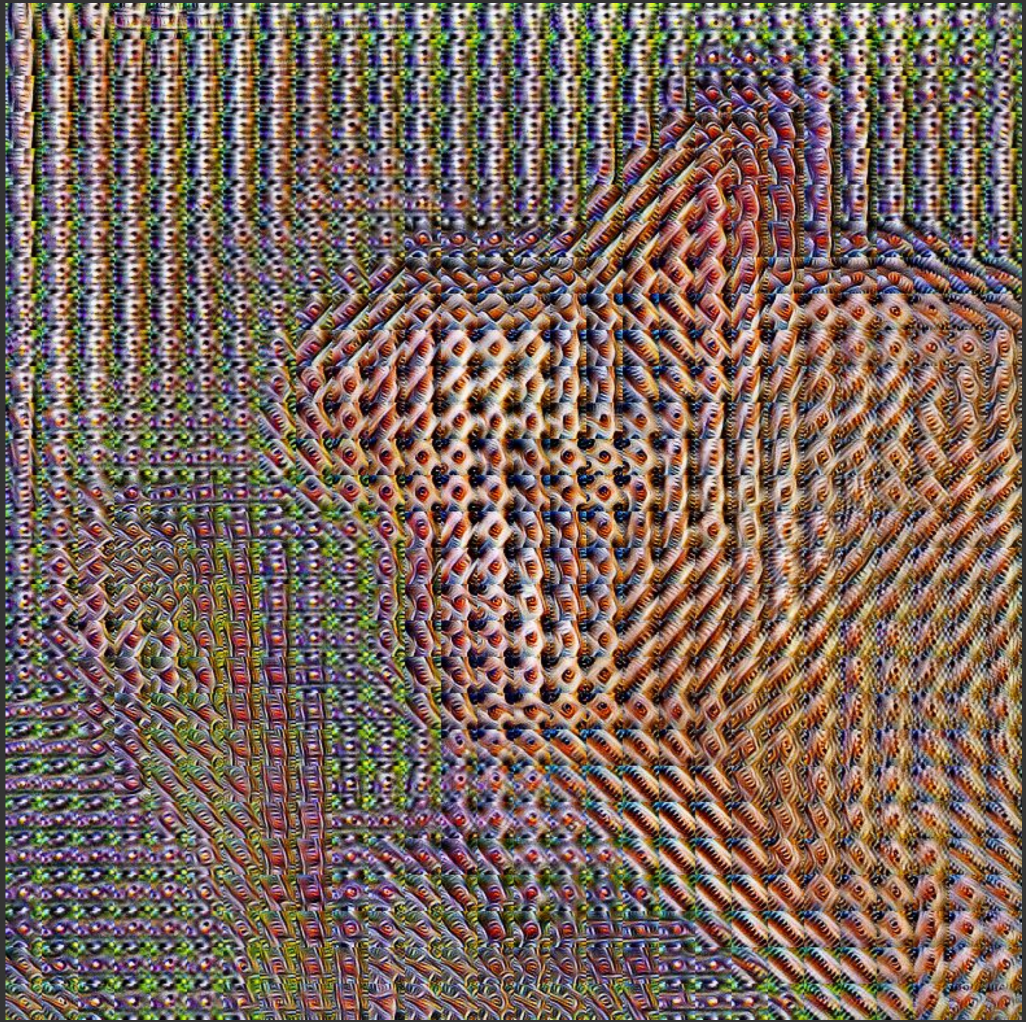


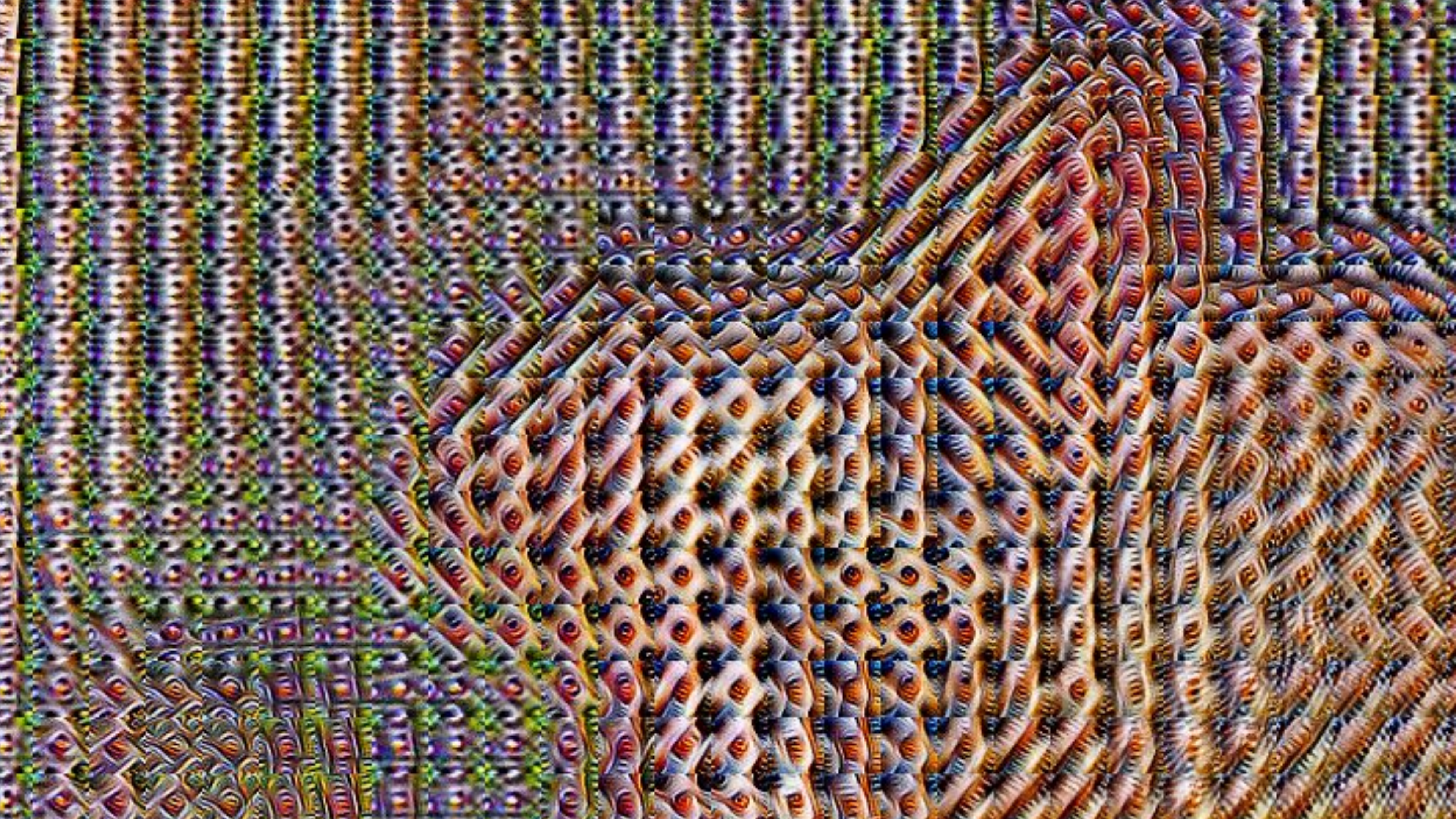
End

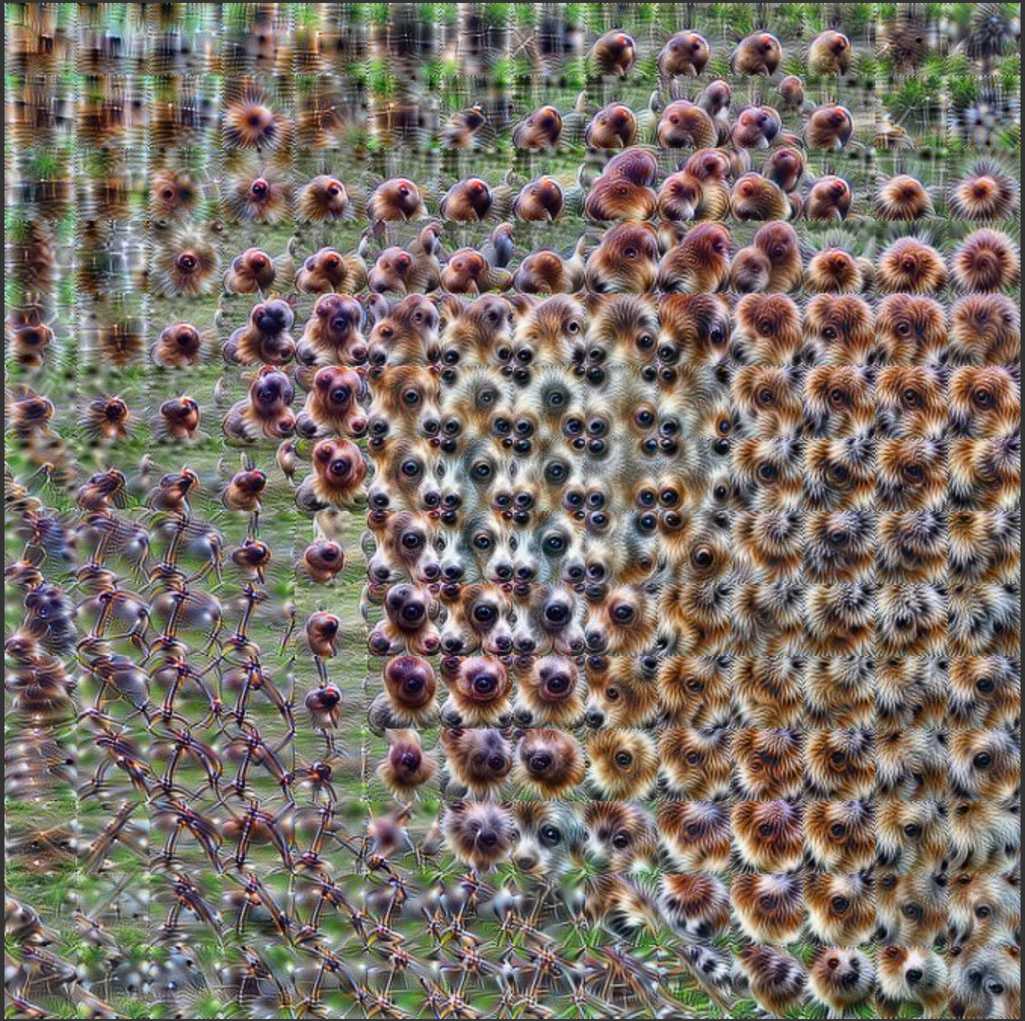


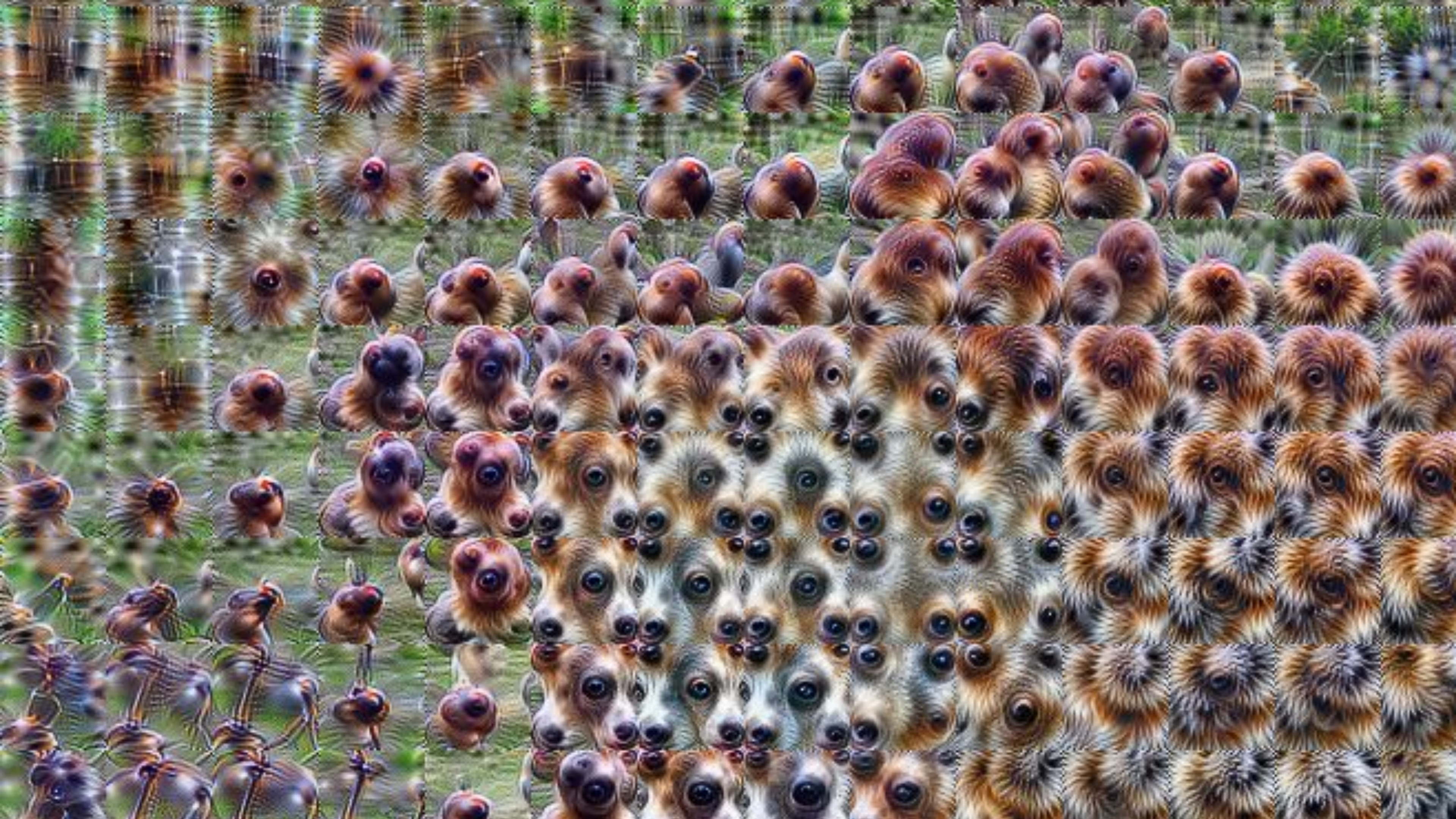
Point your microscope

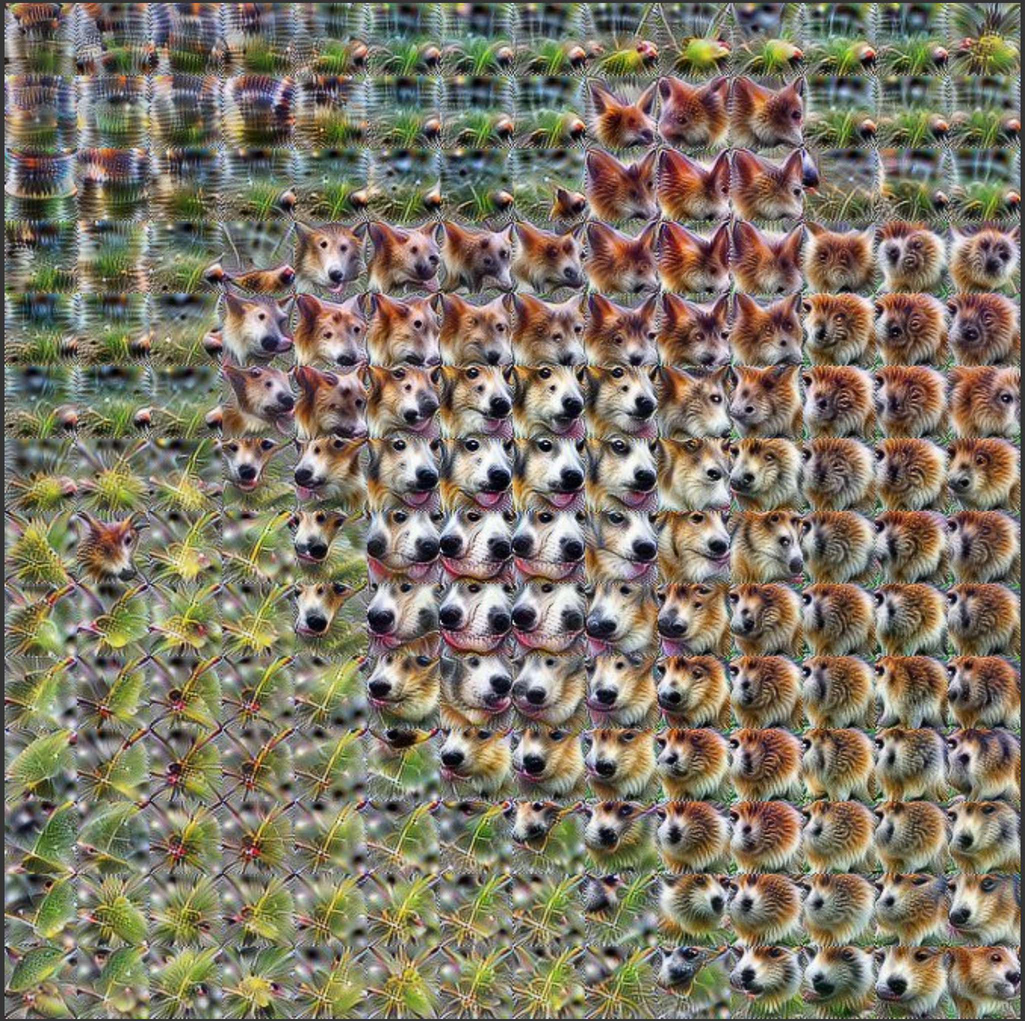
https://distill.pub/2017/feature-visualization/

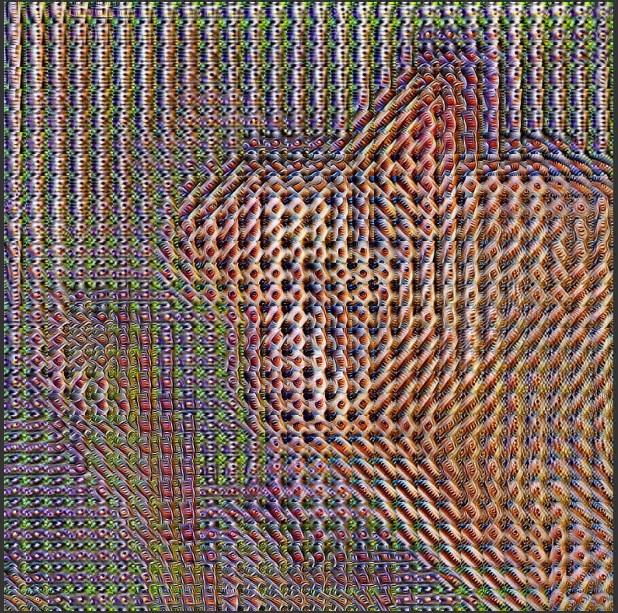
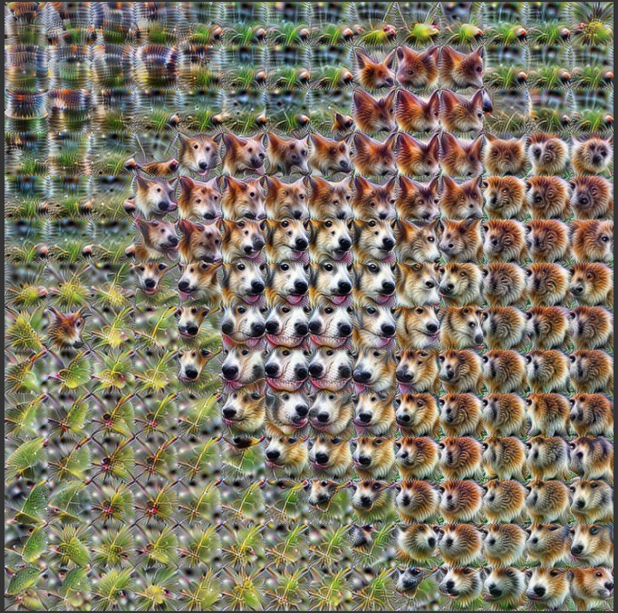


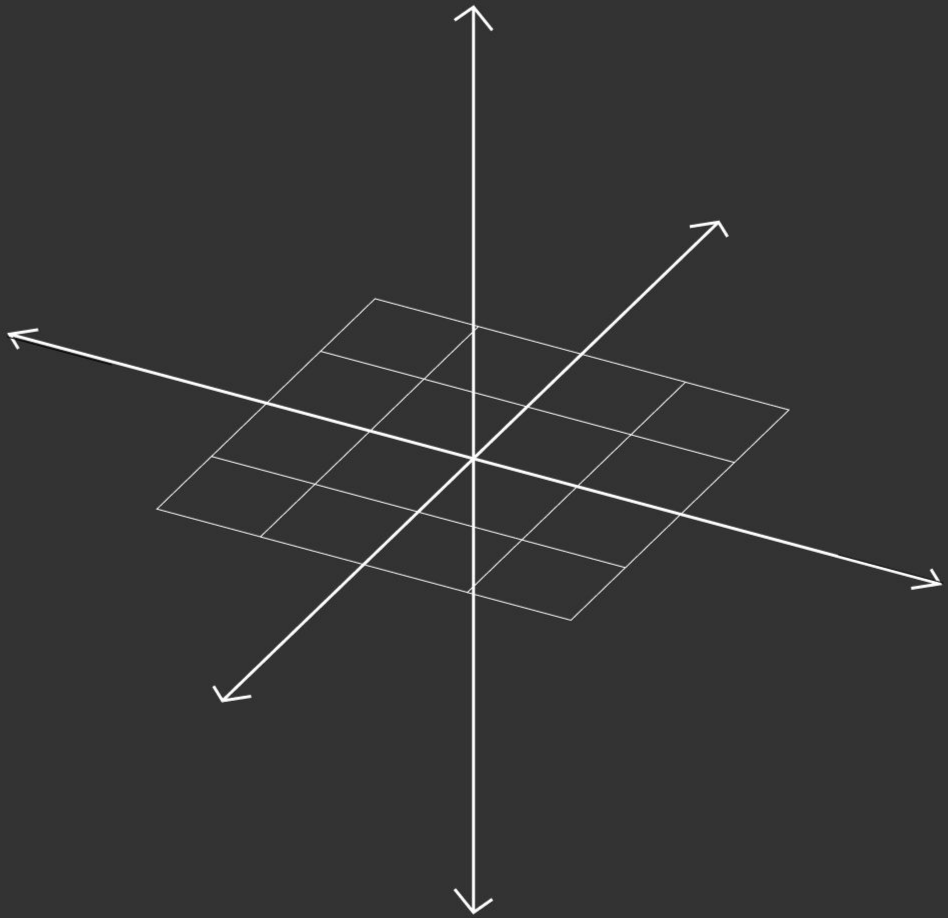


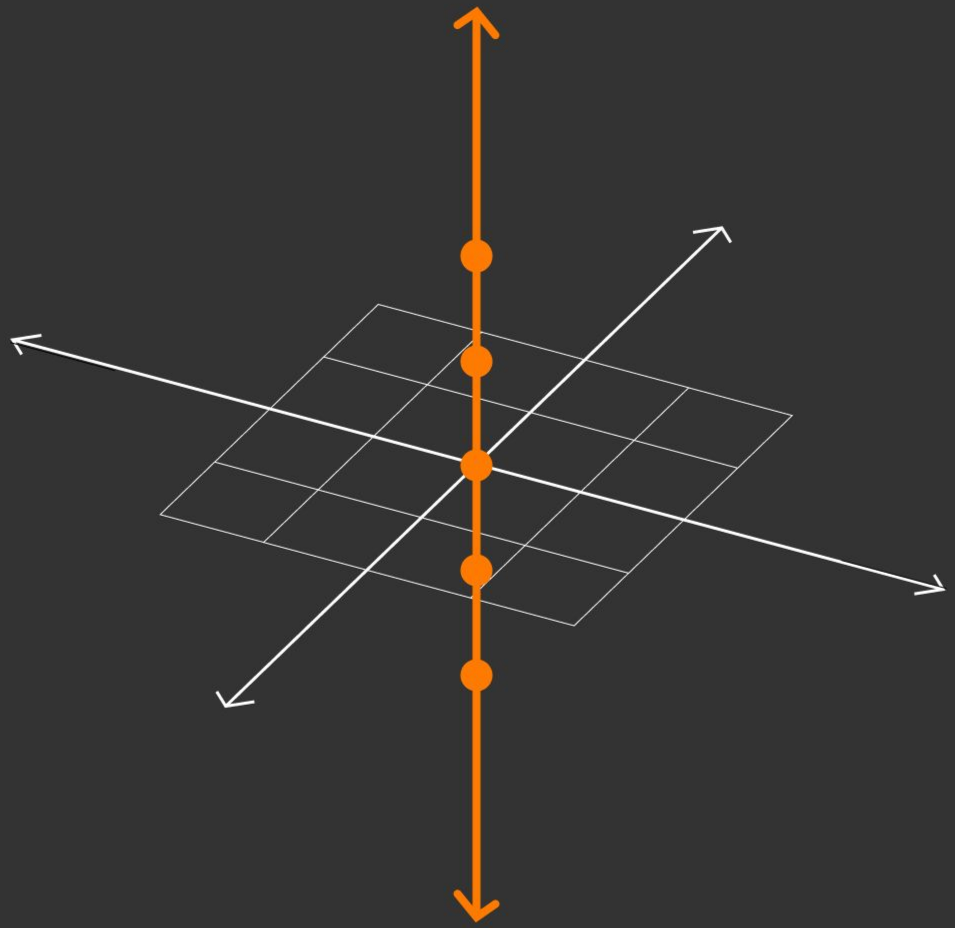


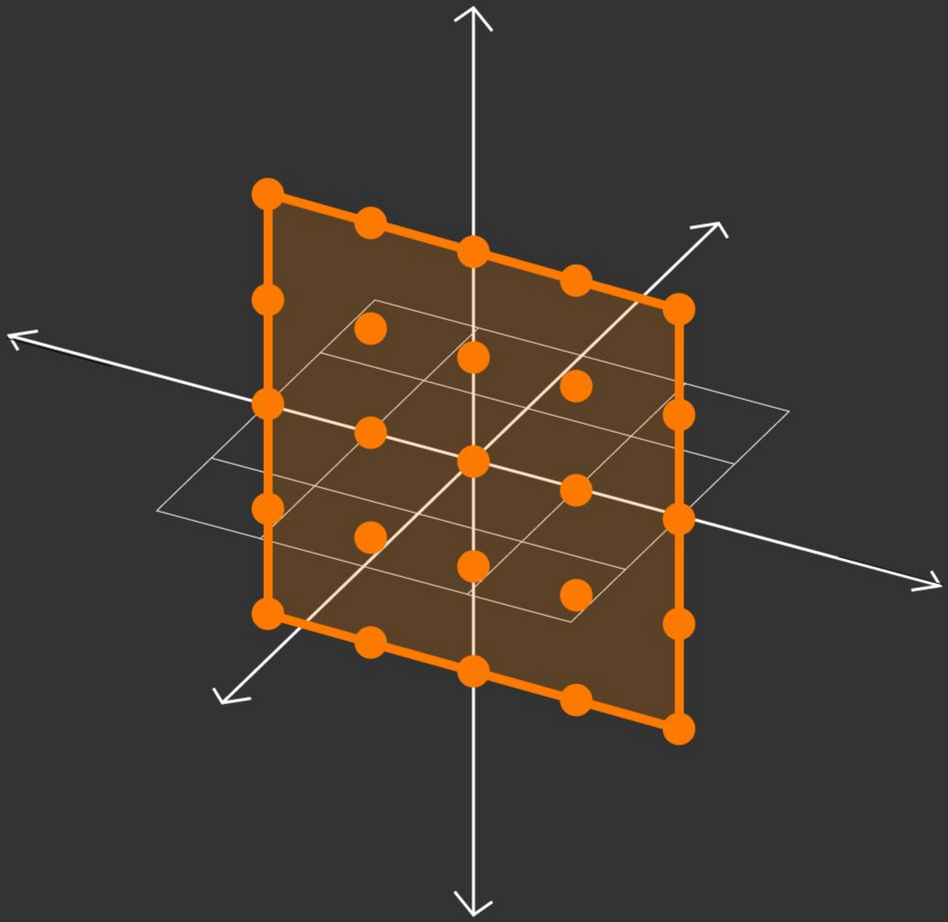


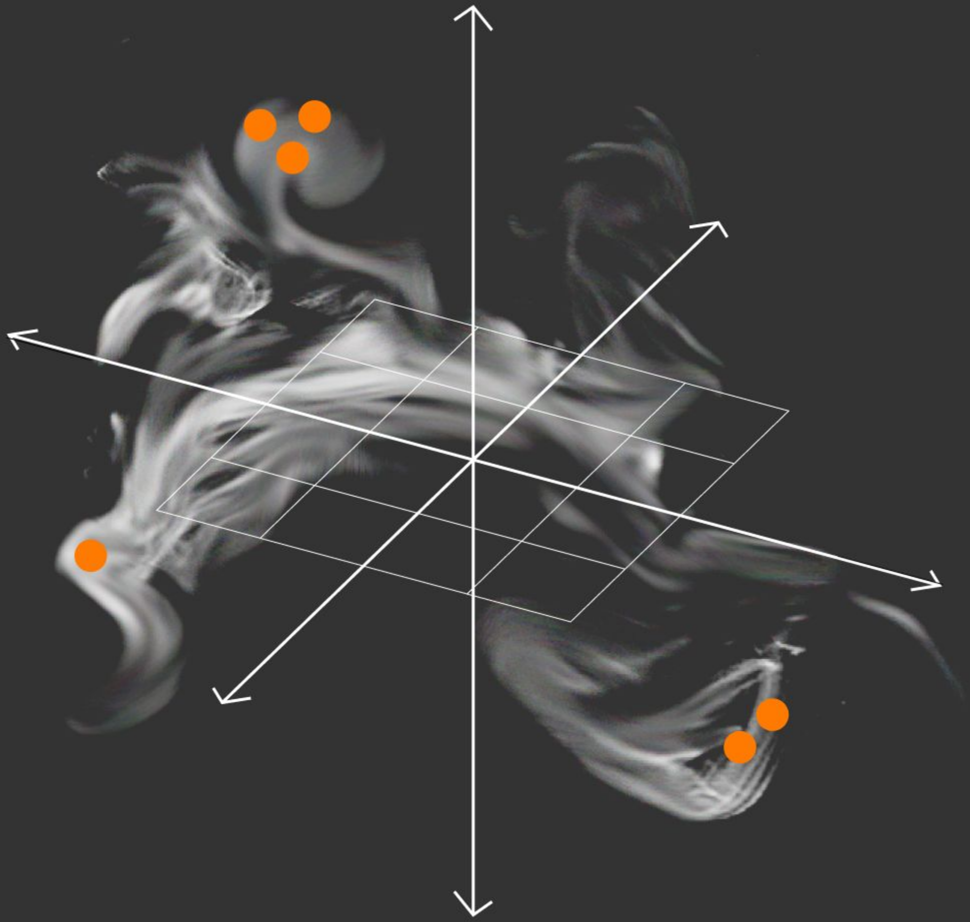


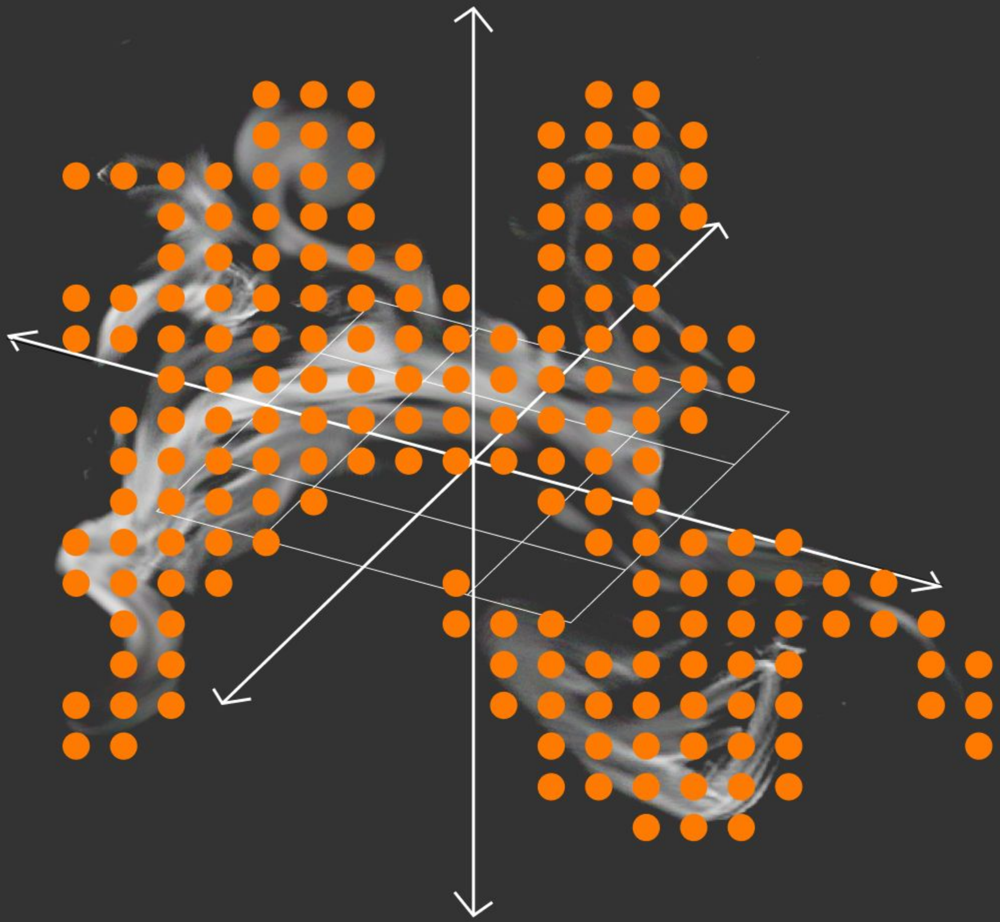




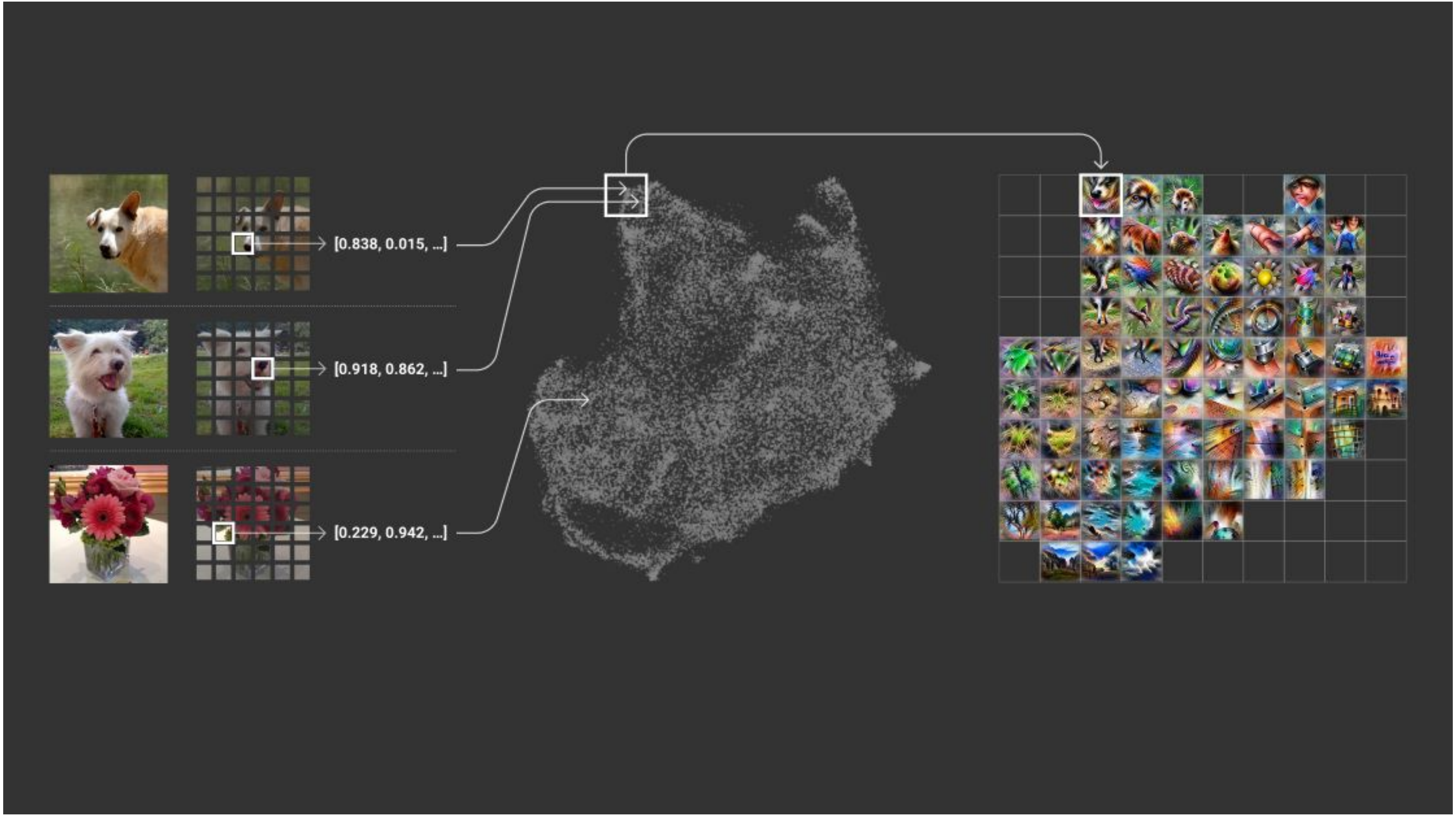


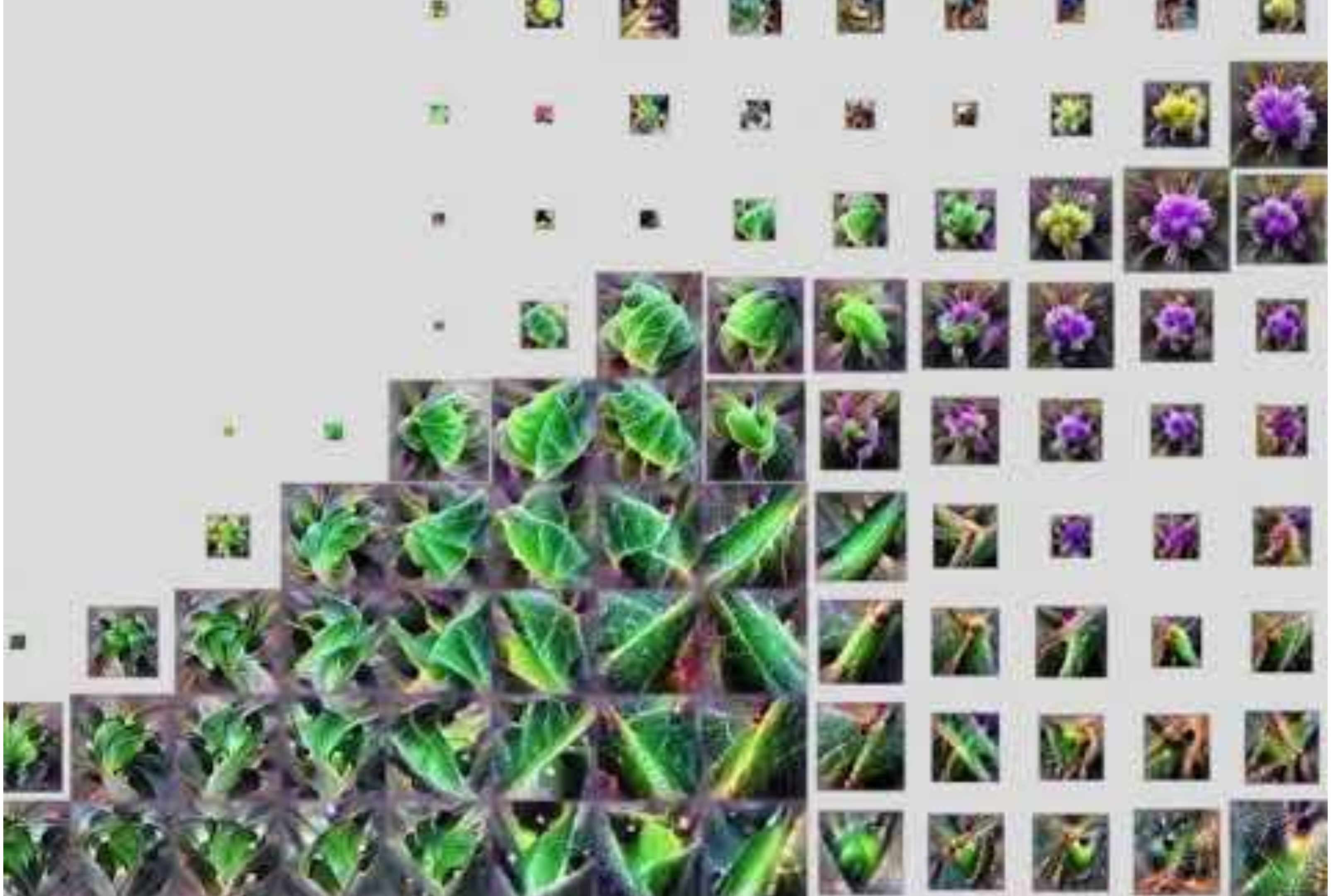


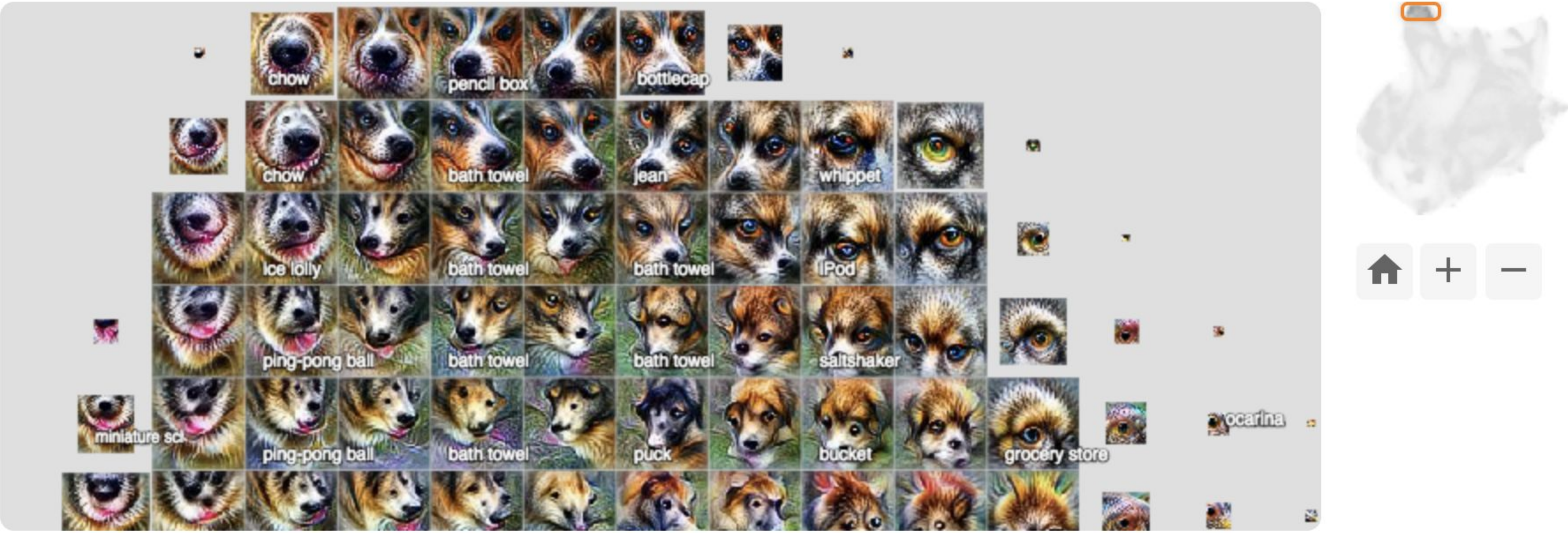




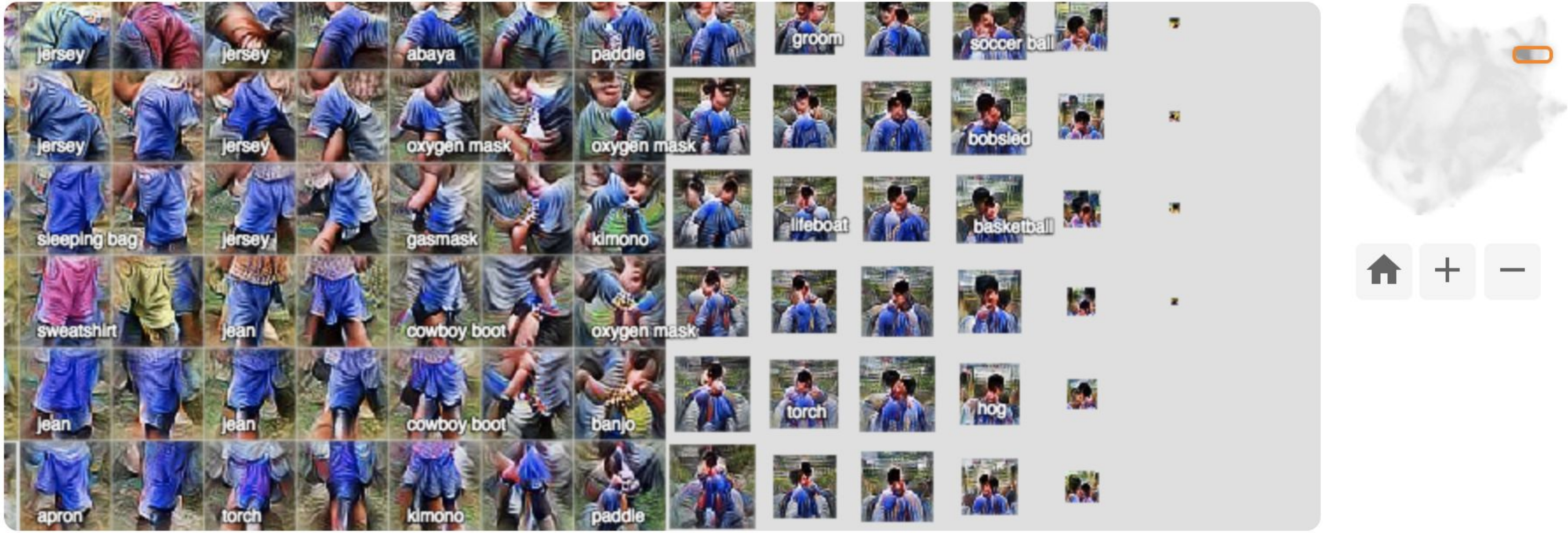


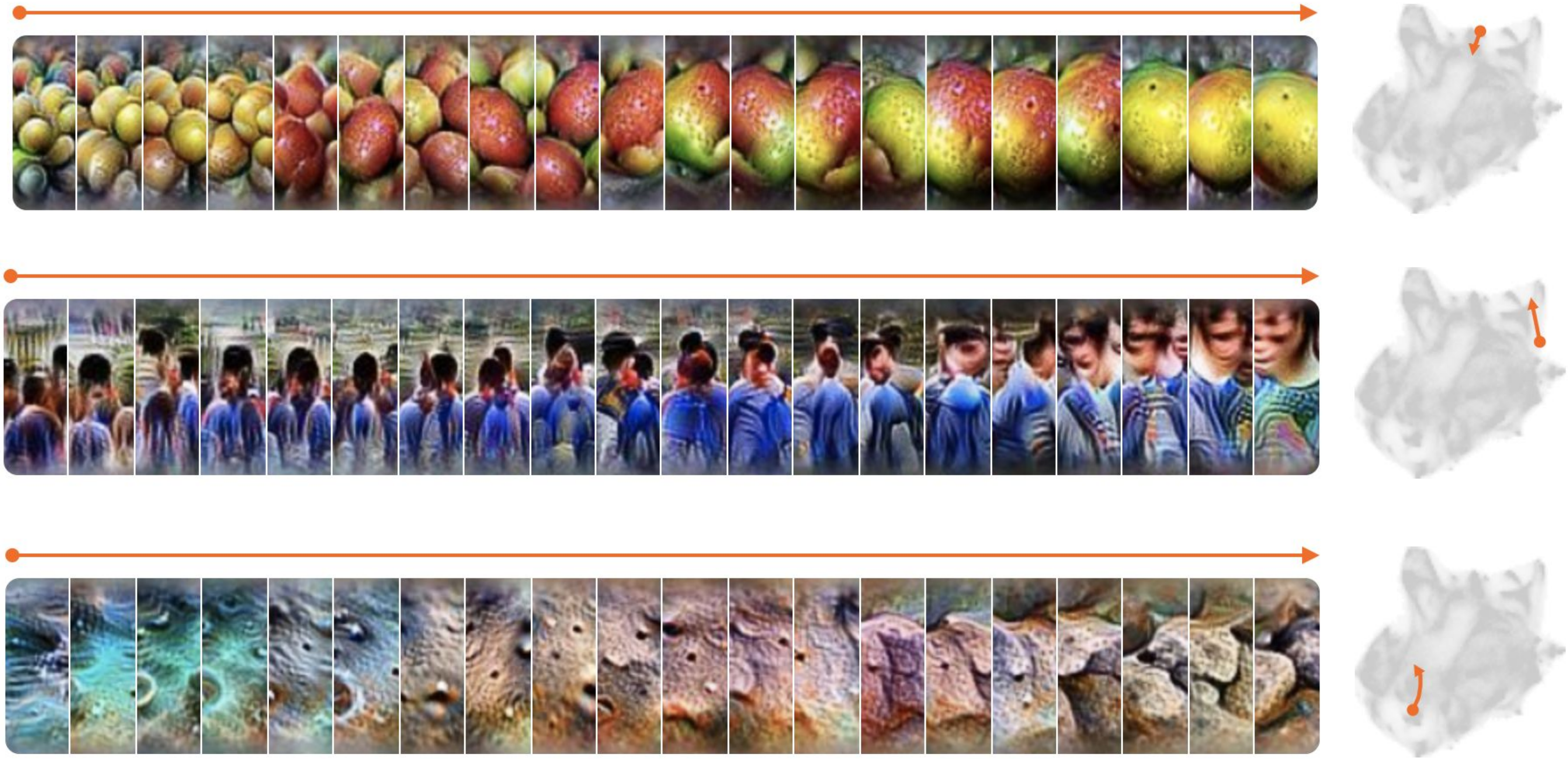

[0.838, 0.015, ...]
[0.918, 0.862, ...]
[0.229, 0.942, ...]



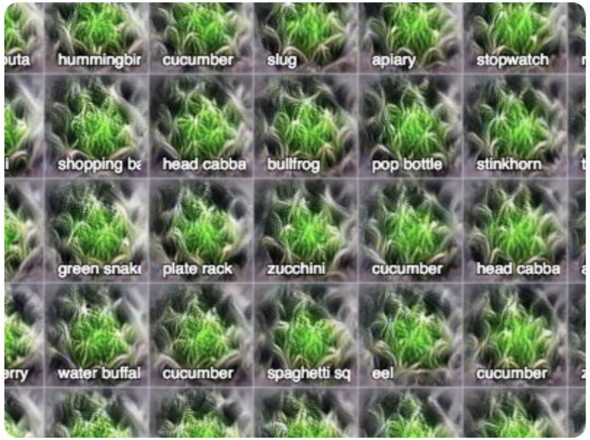


chow
pencil box
bottlecap
chow
bath towel
jean
whippet
ice lolly
bath towel
bath towel
iPod
ping-pong ball
bath towel
bath towel
saltshaker
miniature sc
ping-pong ball
bath towel
puck
bucket
grocery store
ocarina


jersey
jersey
abaya
paddle
groom
soccer ball
jersey
jersey
oxygen mask
oxygen mask
bobsled
sleeping bag
jersey
gasmask
kimono
lifeboat
basketball
sweatshirt
jean
cowboy boot
oxygen mask
jean
jean
cowboy boot
banjo
torch
hog
apron
torch
kimono
paddle

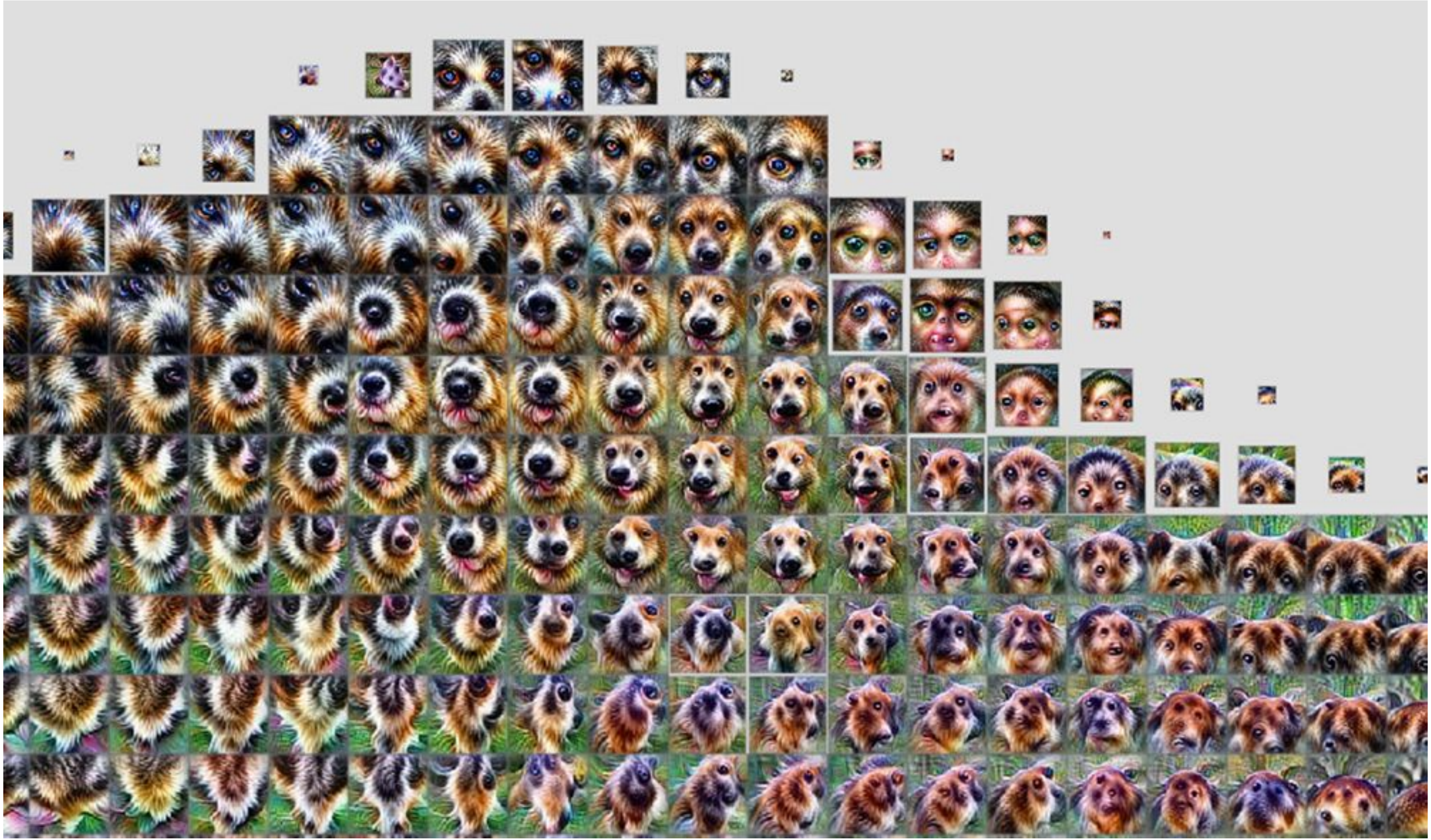


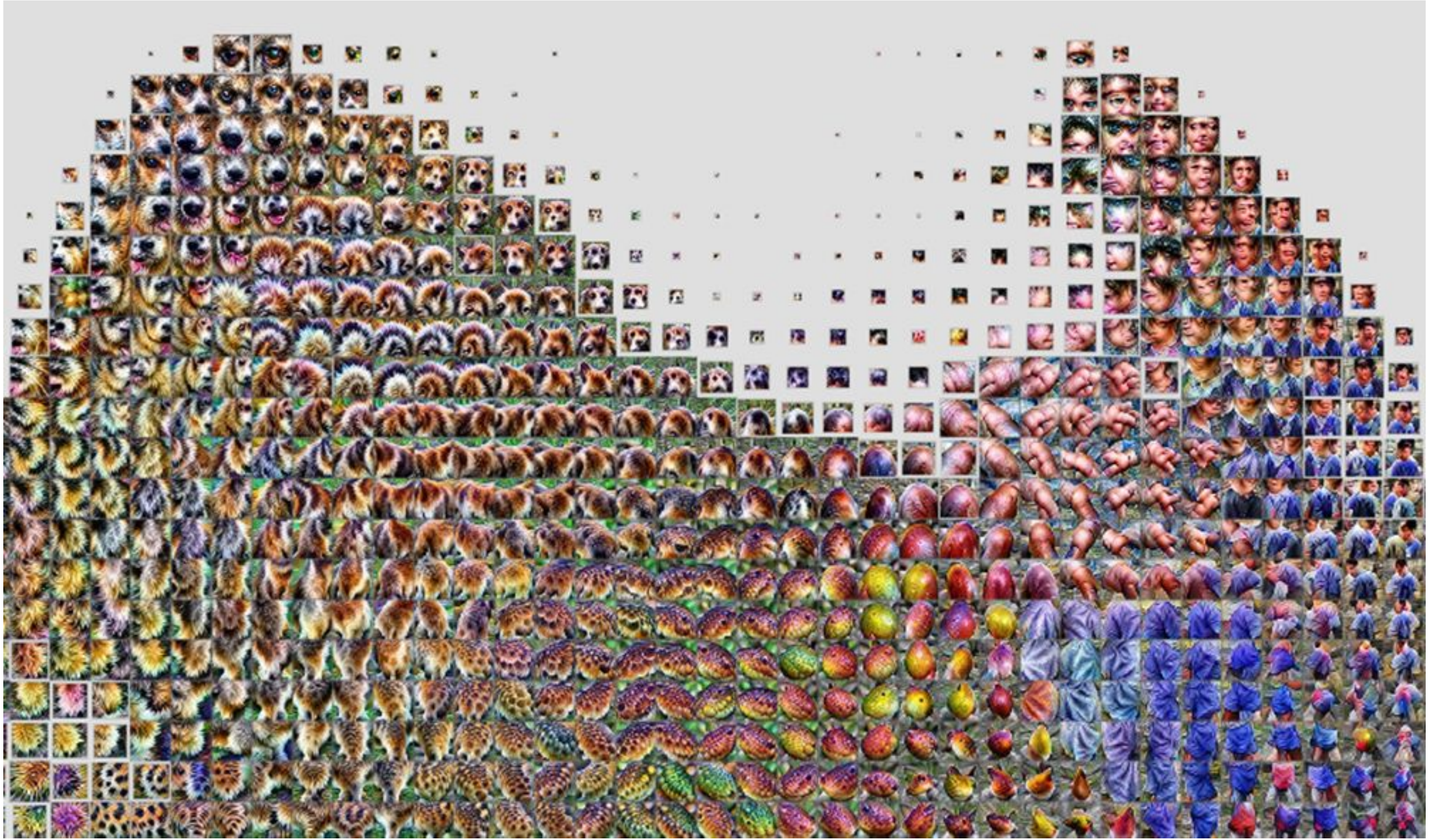
MIXED3B

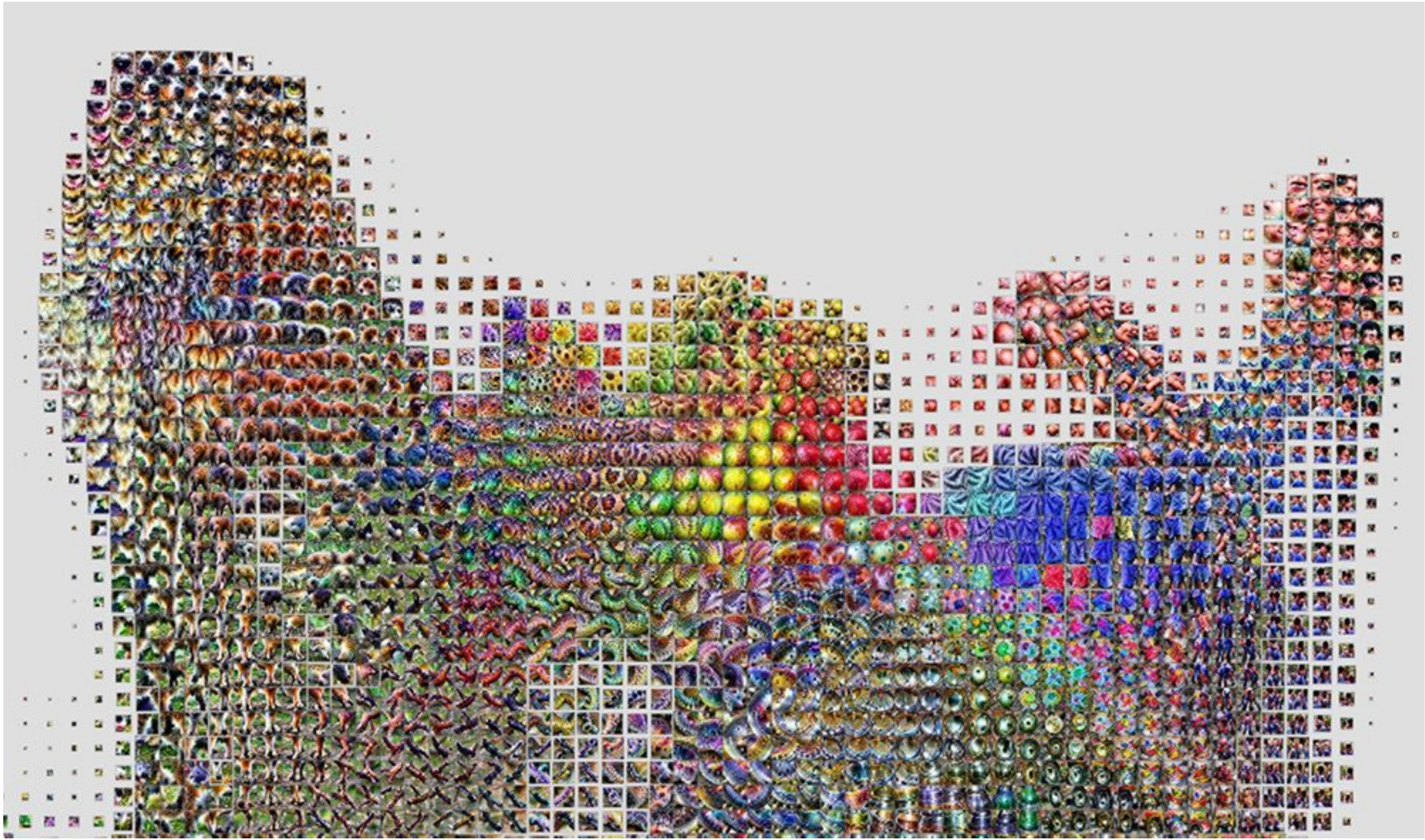


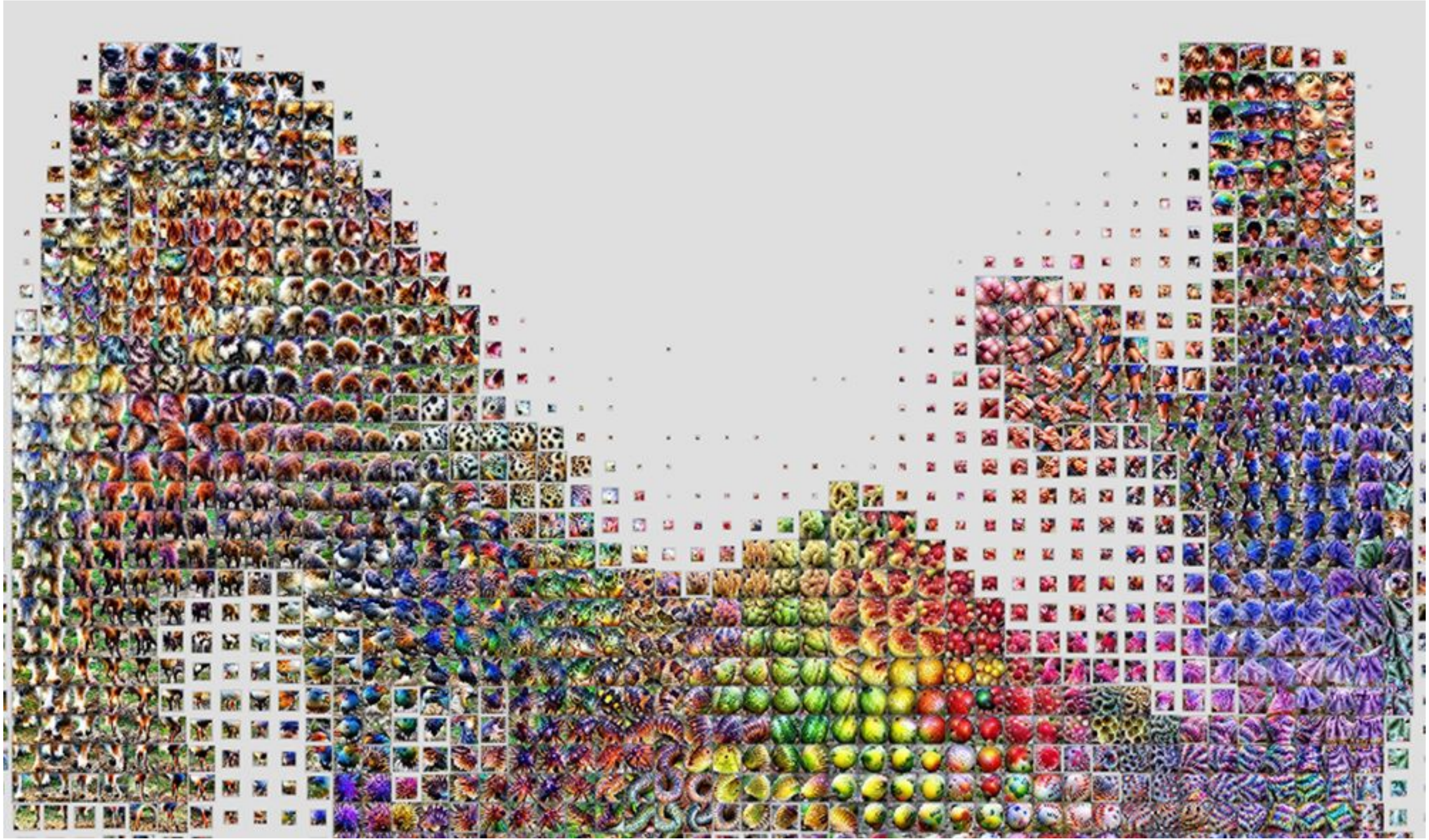
MIXED4C

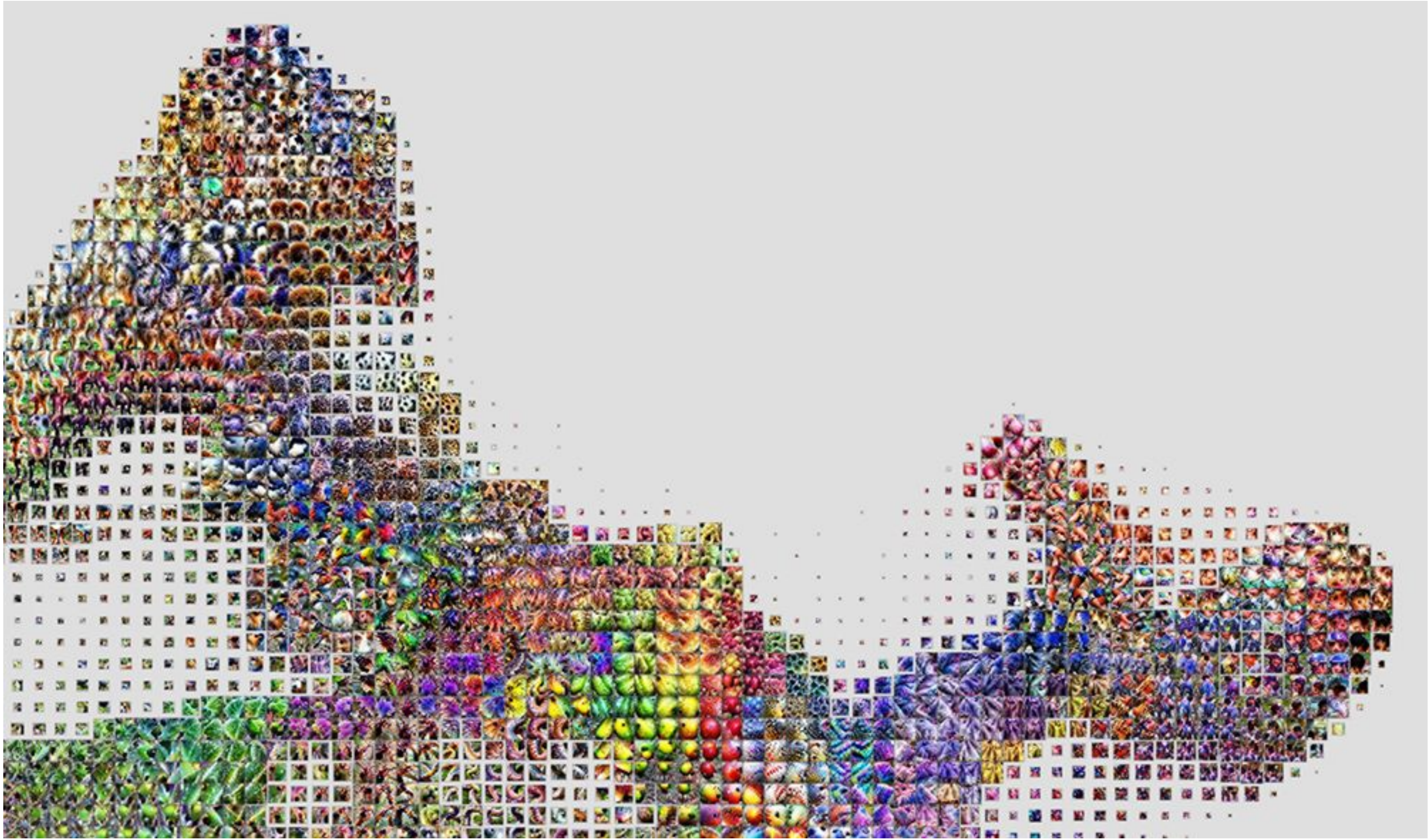
MIXED5B



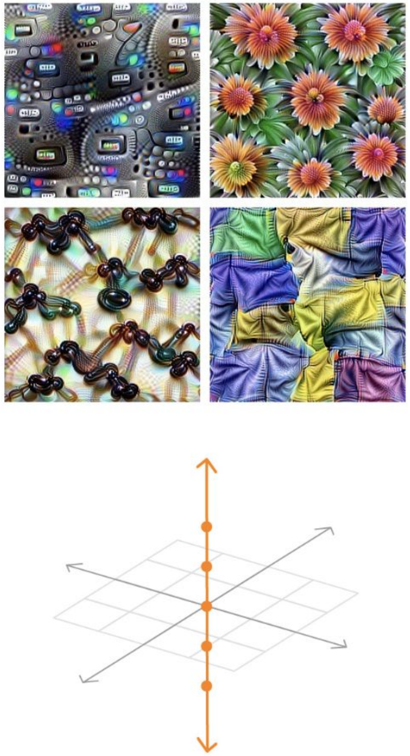




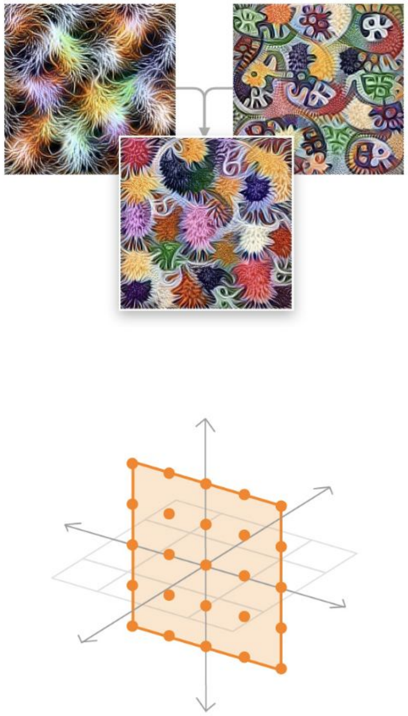




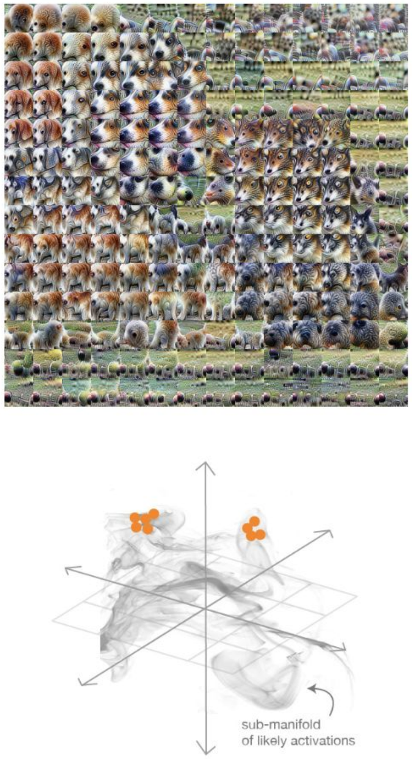
INDIVIDUAL NEURONS



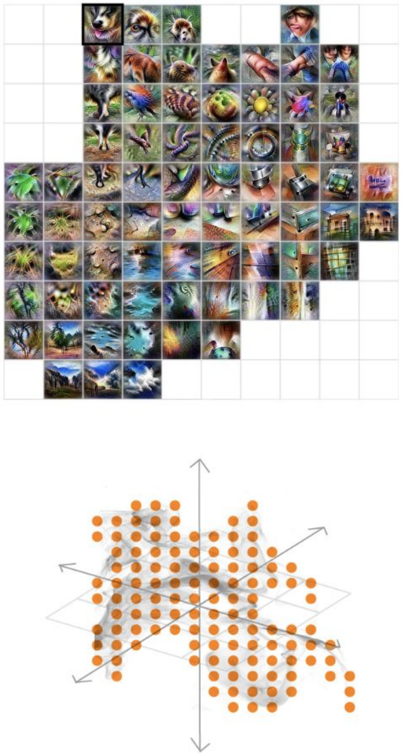
PAIRWISE INTERACTIONS

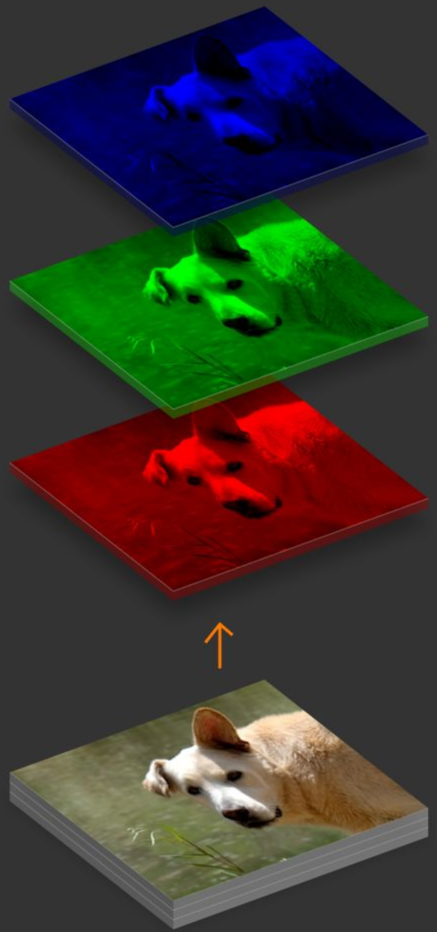


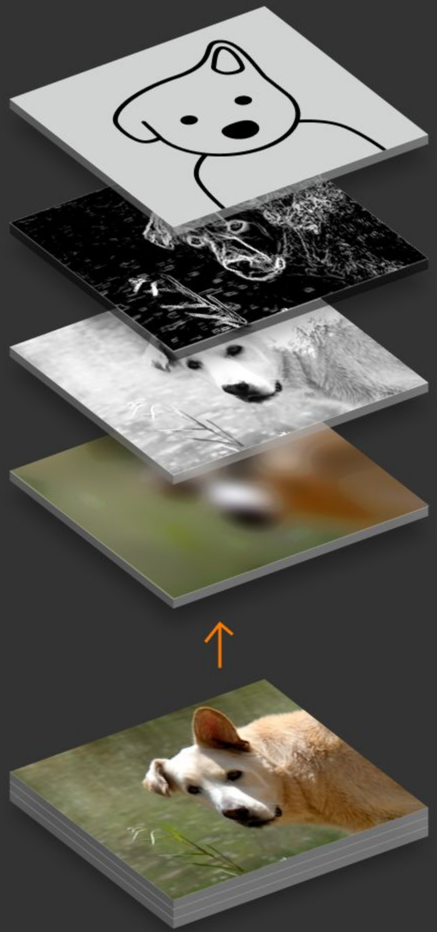
SPATIAL ACTIVATIONS

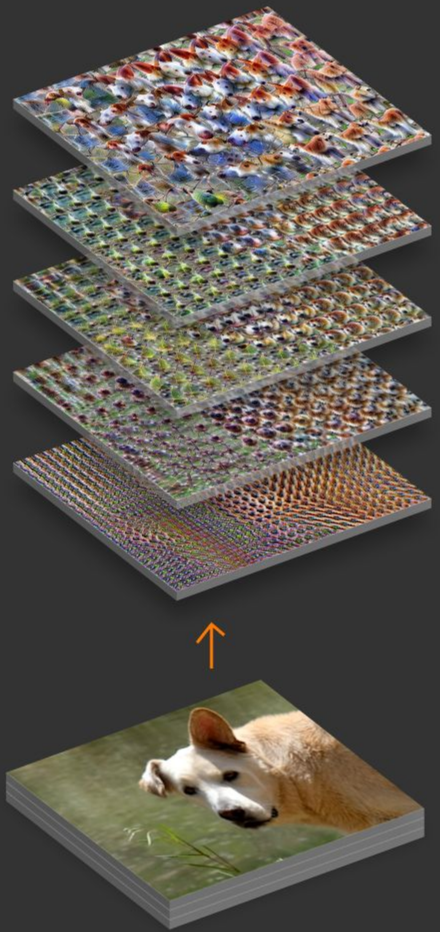


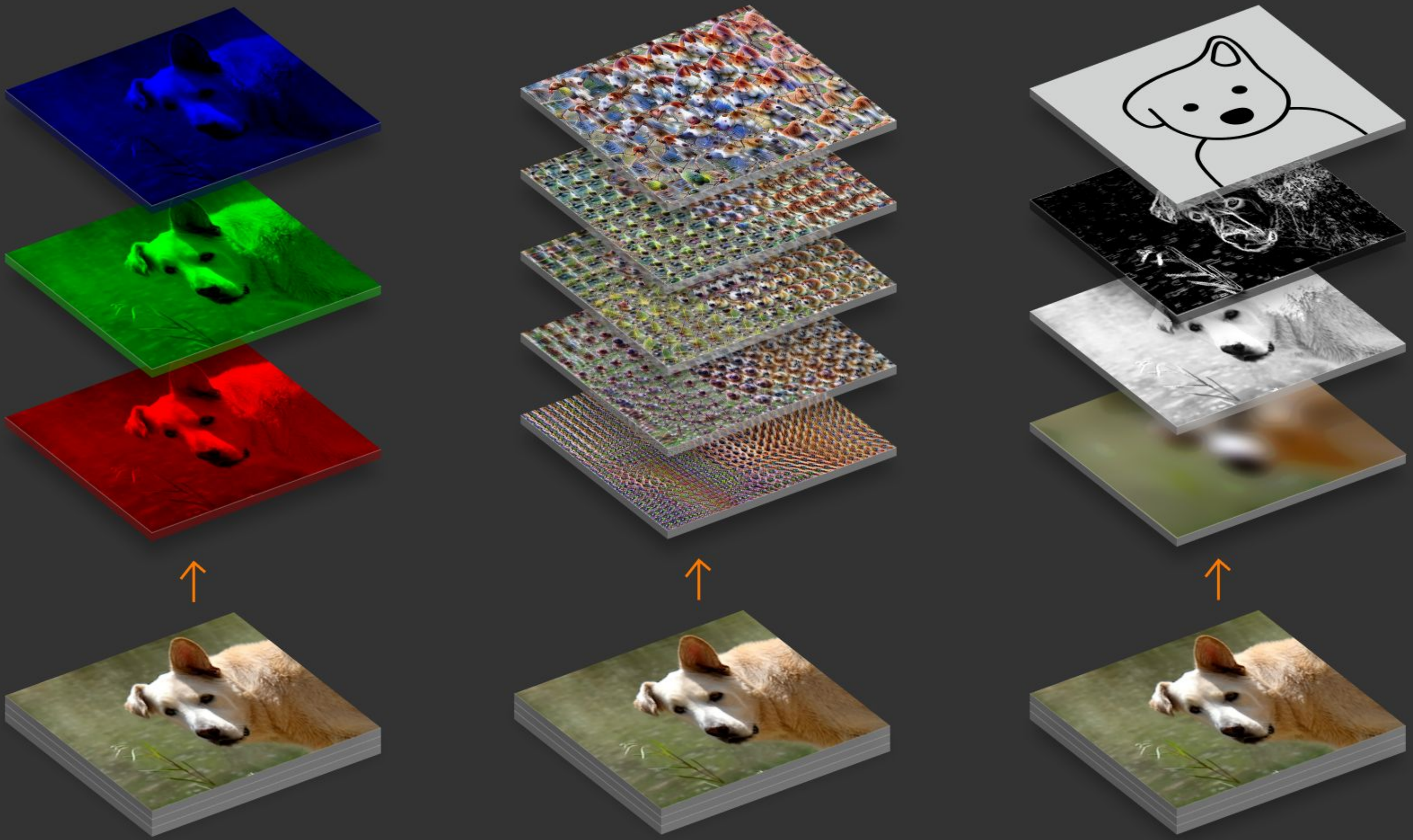
ACTIVATION ATLAS





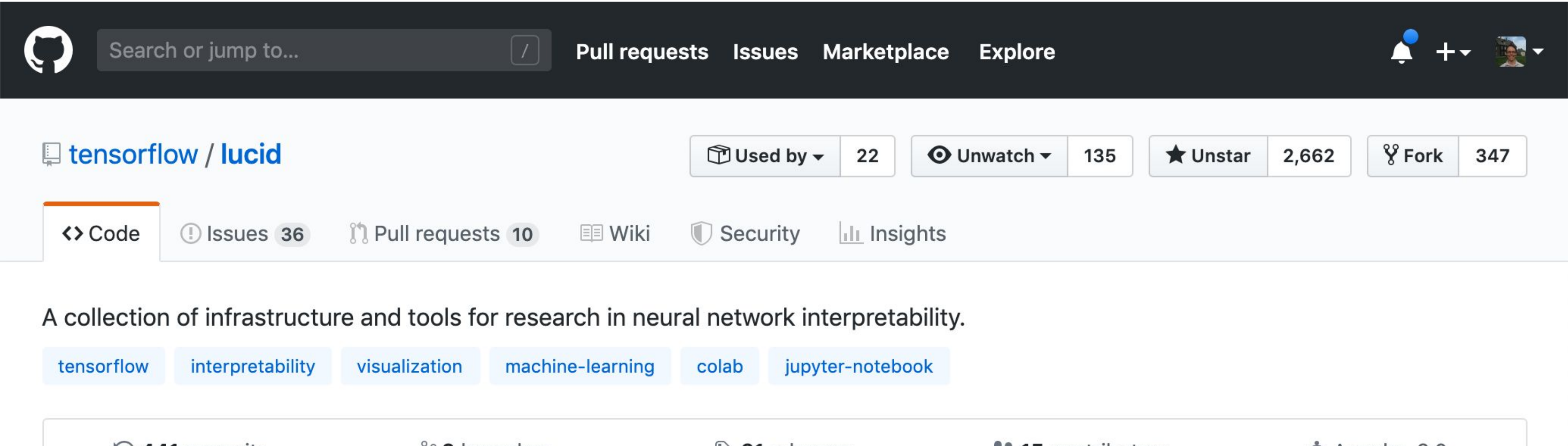






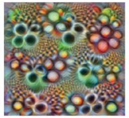
TensorFlow

Lucid
github.com/tensorflow/lucid



Notebooks

Tutorial Notebooks



Lucid Tutorial [colab]

Quickly get started using Lucid. Become familiar with changing **objectives**, **transformations**, and **paramaterization**.

Feature Visualization Notebooks

Notebooks corresponding to the Feature Visualization article

Negative Neurons [colab]

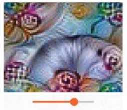
What is the *opposite* of what a neuron is looking for? This can reveal interesting things about the representation.

Diversity Visualization [colab]

Neurons generally respond to multiple things -- sometimes similar and sometimes wildly different. How can we visaulize this diversity?

Neuron Interactions [colab]

Explore how neurons combine and interact. Linear combinations, random directions in neuron space, and interpolation.



Regularizing Visualizations [colab]

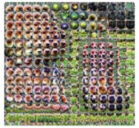
One of the main challenges to visualizing features is regularizing the feature visualizations. Try different techniques and fiddle with hyperparameters.

Building Blocks Notebooks

Notebooks corresponding to the Building Blocks of Interpretability article

Semantic Dictionaries [colab]

Saying "neuron 312 fired" isn't very meaningful to humans. Combining neuron activations with feature visualization can make things much more meaningful.



Activation Grids [colab]

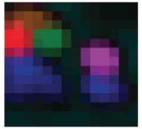
Activation grids can help us see how the network understood each spatial position.

Spatial Attribution [colab]

Do attribution to spatial positions in hidden layers -- either from the output or other hidden layers. This is similar to traditional saliency maps.

Channel Attribution [colab]

How did different features effect the output? We can use attribution between channels in hidden layers and the output, along with feature visualization, to explore this.

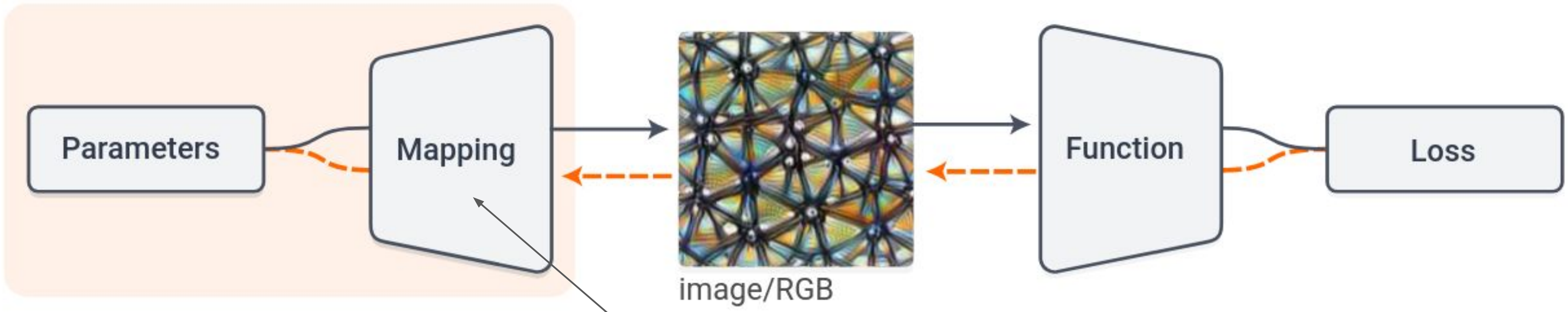


Neuron Groups [colab]

Explore how groups of neurons work together to represent objects in an image. Automatically extract neuron groups and then visualize them.

```
model = lucid.modelzoo.vision_models.InceptionV1()
model.load_graphdef()
_ = lucid.optvis.render.render_vis(model, "mixed4a:476")
```

Differential Image Parameterizations

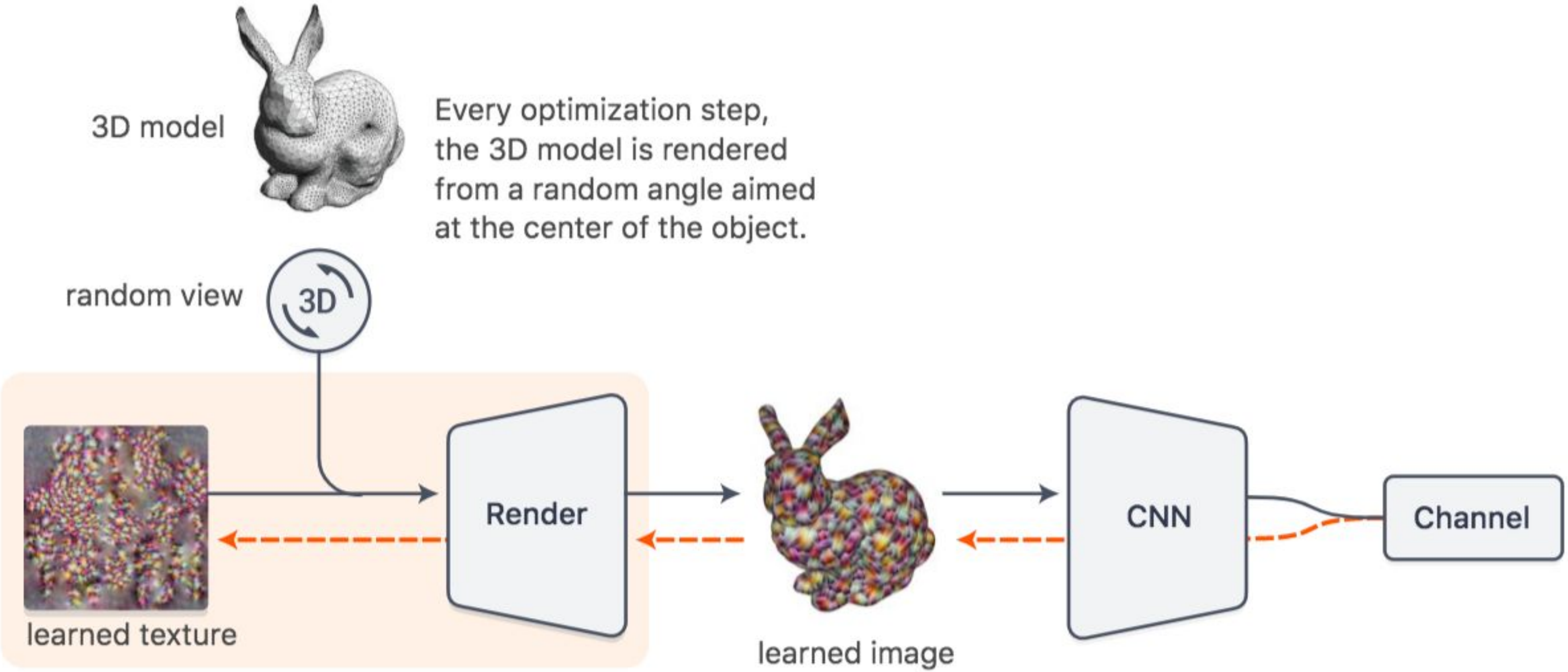


Anything Differentiable

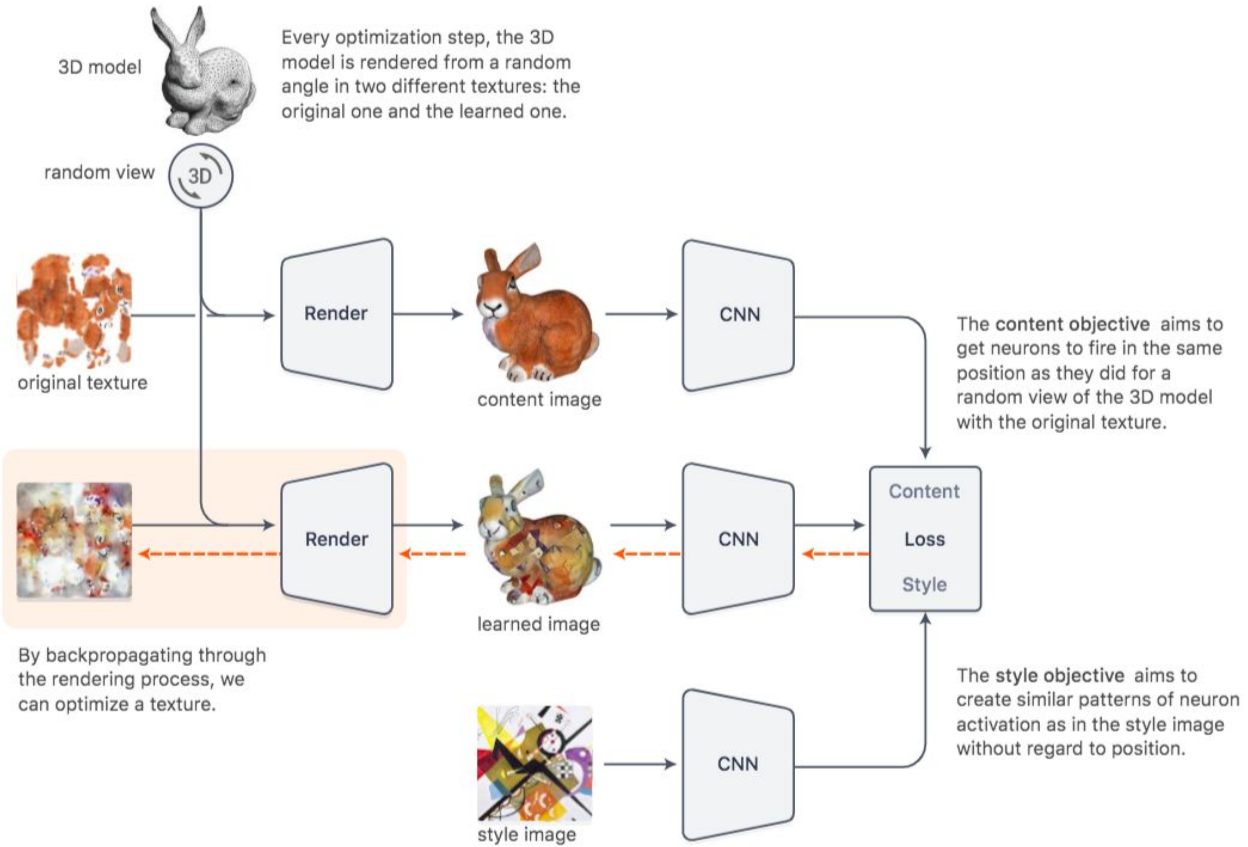
Can be stochastic!

Anything Differentiable

Case study: 3D Mesh texturing



Application: 3D texture style transfer



TensorFlow

Q&A

Shan Carter

TensorFlow

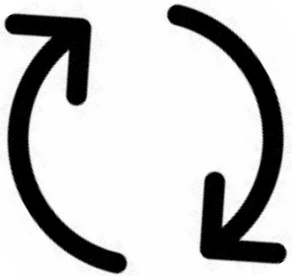
Conclusion

Sofien Bouaziz (@_sofien_)


CARS

“Tennis for Two” - *William Higinbotham, 1958*

Deep Learning for Graphics



Graphics for Deep Learning

TensorFlow

Thank You!

Paige Bailey, Sofien Bouaziz,
Shan Carter, Josh Gordon,
Christian Häne, Alexander Mordvintsev,
Julien Valentin, Martin Wicke

Latest version of the materials hosted

[here]

SIGGRAPH '19 Courses, July 28 - August 01, 2019, Los Angeles, CA, USA
ACM 978-1-4503-6307-5/19/07.
10.1145/3305366.3328041