Introduction to Machine Learning

Deep Learning Applications

Barnabás Póczos

MACHINE LEARNING DEPARTMENT

Carnegie Mellon.
School of Computer Science

Applications

- **Image Classification**
  (Alexnet, VGG, Resnet) on Cifar 10, Cifar 100, Mnist, Imagenet
- **Art**
  - Neural style transfer on images and videos
  - Inception, deep dream
- **Visual Question Answering**
- **Image and Video Captioning**
- **Text generation from a style**
  - Shakespare, Code, receipts, song lyrics, romantic novels, etc
- **Story based question answering**
- **Image generation, GAN**
- **Games, deep RL**

Deep Learning Software Packages

Collection: http://deeplearning.net/software_links/

- **Torch:** http://torch.ch/
- **Caffe:** http://caffe.berkeleyvision.org/
  - **Caffe Model Zoo**: https://github.com/BVLC/caffe/wiki/Model-Zoo
- **NVIDIA Digits:** https://developer.nvidia.com/digits
- **Tensorflow:** https://www.tensorflow.org/
- **Theano:** http://deeplearning.net/software/theano/
- **Lasagne:** http://lasagne.readthedocs.io/en/latest/
- **Keras:** https://keras.io/
- **MXNet:** http://mxnet.io/
- **Dynet:** https://github.com/clab/dynet
- **Microsoft Cognitive Toolkit (MCNTK)**

  https://www.microsoft.com/en-us/research/product/cognitive-toolkit/

Torch

Torch is a scientific computing framework with wide support for machine learning algorithms that puts GPUs first.

It is easy to use and efficient, thanks to an easy and fast scripting language, LuaJIT, and an underlying C/CUDA implementation.

Torch tutorials:

- https://github.com/bapoczos/TorchTutorial
- https://github.com/bapoczos/TorchTutorial/blob/master/DeepLearningTorchTutorial.ipynb
- https://github.com/bapoczos/TorchTutorial/blob/master/iTorch_Demo.ipynb

- Written in Lua
- Used by Facebook
- Often faster than Tensorflow, Theano

Tensorflow

TensorFlow™ is an open source library for numerical computation using data flow graphs.

- Nodes in the graph represent mathematical operations,
- while the graph edges represent the multidimensional data arrays (tensors) communicated between them.

Tensorflow tutorials: https://www.tensorflow.org/tutorials/

- ❑ Developed by Google Brain and used by Google in many products
- ❑ Well-documented
- ❑ Probably the most popular
- ❑ Easy to use with Python

Image Classification

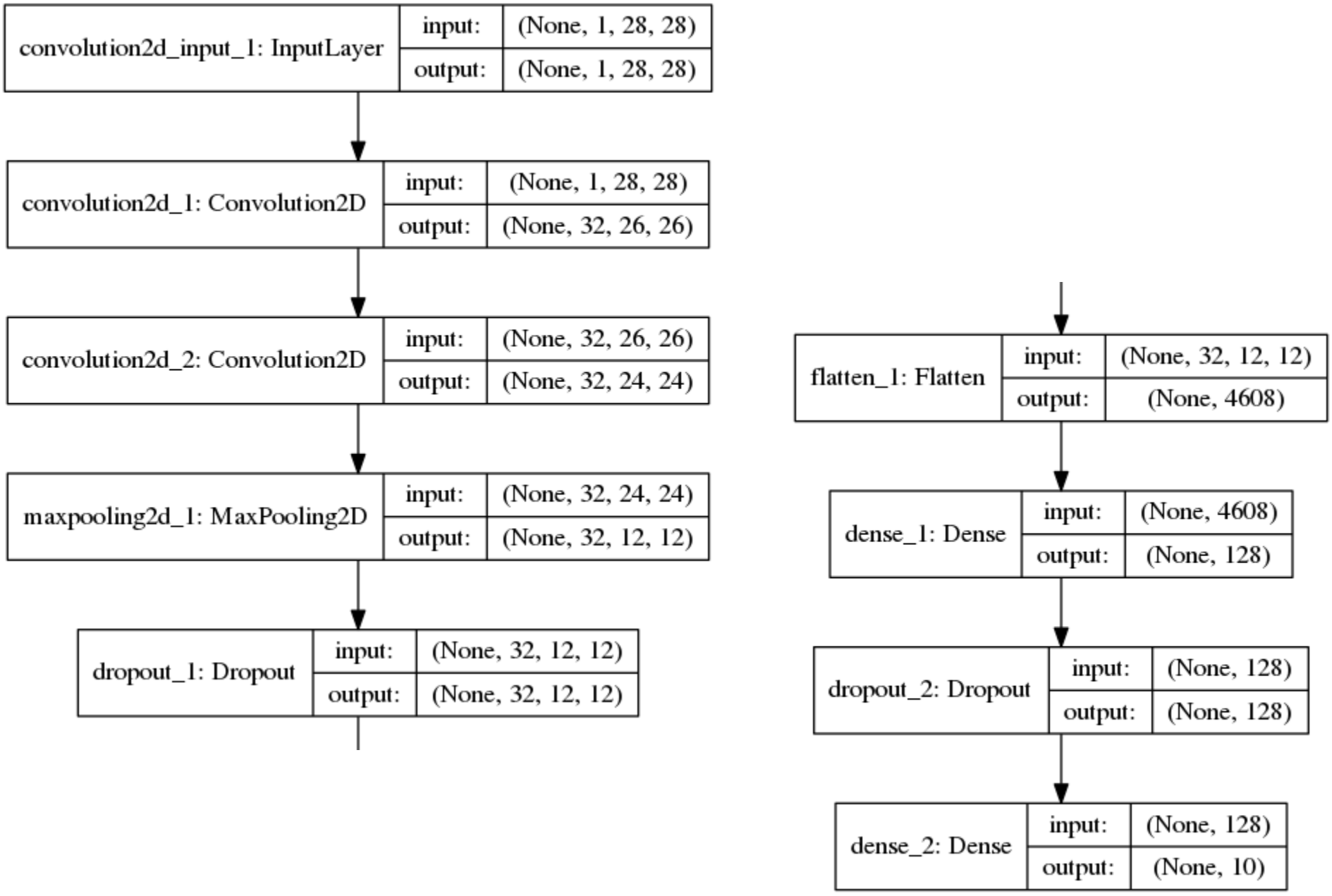
Image Classification with Keras

Keras for building and training a convolutional neural network and using the network for image classification:

Demonstration on MNIST:

https://github.com/bapoczos/keras-mnist-ipython/blob/master/Keras_mnist_tutorial_v1.ipynb

Image Classification with Keras



```
In [32]: model.summary()
```

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
convolution2d_1 (Convolution2D)  (None, 32, 26, 26)    320         convolution2d_input_1[0][0]
____________________________________________________________________________________________________
convolution2d_2 (Convolution2D)  (None, 32, 24, 24)    9248        convolution2d_1[0][0]
____________________________________________________________________________________________________
maxpooling2d_1 (MaxPooling2D)    (None, 32, 12, 12)    0           convolution2d_2[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 32, 12, 12)    0           maxpooling2d_1[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 4608)          0           dropout_1[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 128)           589952      flatten_1[0][0]
____________________________________________________________________________________________________
dropout_2 (Dropout)              (None, 128)           0           dense_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 10)            1290        dropout_2[0][0]
====================================================================================================
Total params: 600,810
Trainable params: 600,810
```

Number or parameters:

320 = 32*(3*3+1)

9248 = 32*(32*3*3+1)

4608 = 32*12*12

589952= (4608+1)*128

1290 = 10*(128+1)

600810= 320+9248+589952+1290

The shape of the weight matrices without the bias parameter:

```
print(model.layers[0].get_weights()[0].shape)
meters
print(model.layers[1].get_weights()[0].shape)
print(model.layers[5].get_weights()[0].shape)
print(model.layers[7].get_weights()[0].shape)
```

```
(32, 1, 3, 3)
(32, 32, 3, 3)
(4608, 128)
(128, 10)
```

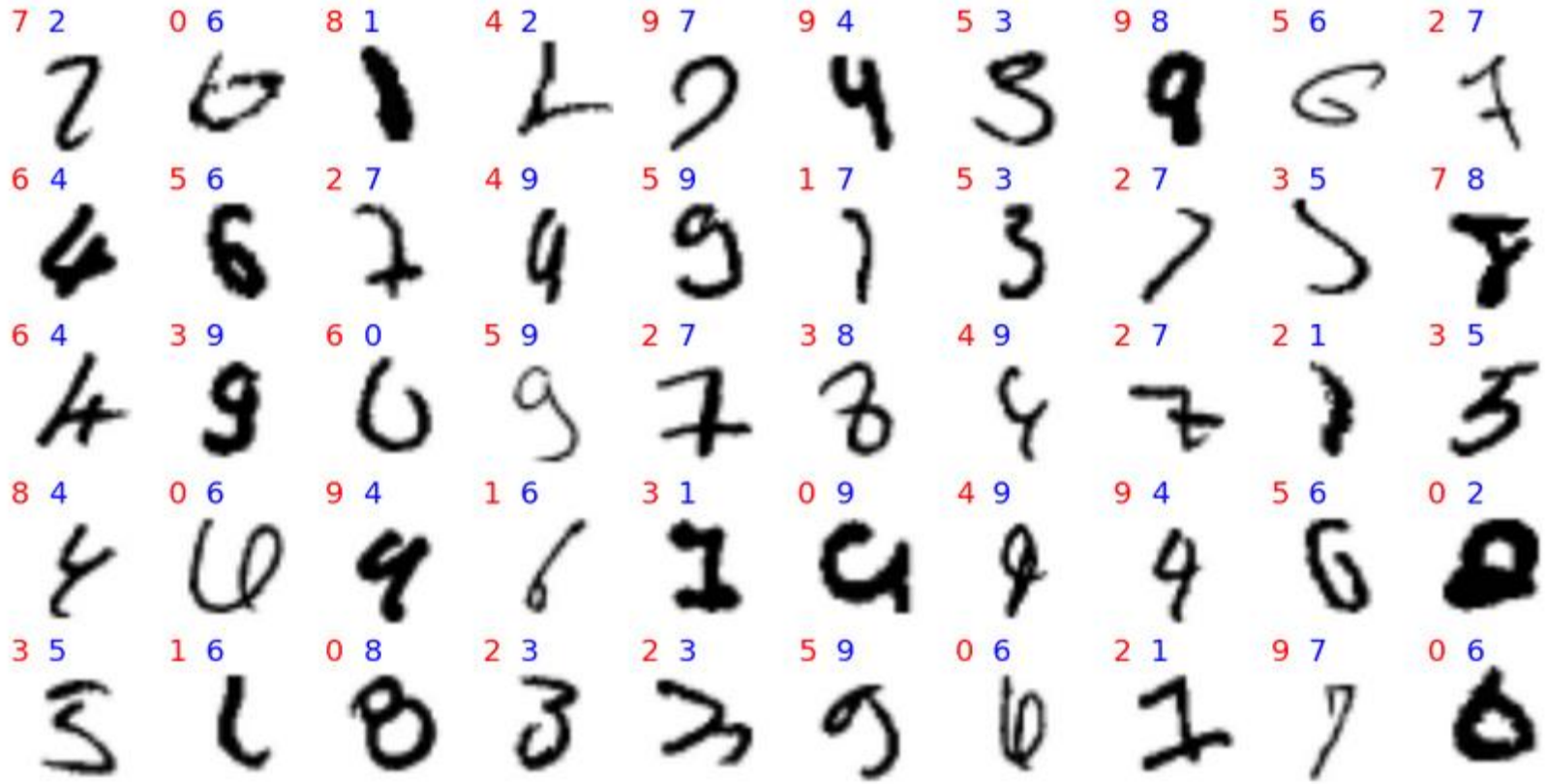
Image Classification with Keras

The confusion matrix:

col_0	0	1	2	3	4	5	6	7	8	9
row_0										
0	975	0	1	0	0	0	4	1	2	3
1	0	1128	0	0	0	0	2	1	0	0
2	0	3	1026	2	0	0	0	11	1	0
3	1	2	0	1004	0	4	0	1	1	3
4	0	0	2	0	973	0	1	0	0	4
5	0	0	0	3	0	887	4	0	0	4
6	3	1	0	0	4	1	946	0	1	0
7	0	0	3	0	0	0	0	1010	2	0
8	1	1	0	1	1	0	1	1	965	2
9	0	0	0	0	4	0	0	3	2	993

Image Classification with Keras

Some misclassified images:



Red = Predicted label, Blue = True label.

Image Classification with Keras using VGG19

Vgg19 network test on Imagenet using keras:

https://github.com/bapoczos/keras-vgg19test-ipython/blob/master/keras_vggtest.ipynb

Image Classification using VGG

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

Karen Simonyan & Andrew Zisserman

ICLR 2015

Visual Geometry Group, University of Oxford

https://arxiv.org/pdf/1409.1556.pdf

- Networks of increasing depth using very small (3 × 3) convolution filters
- Shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19
- ImageNet Challenge 2014: first and the second places in the localization and classification tracks respectively.

VGG16

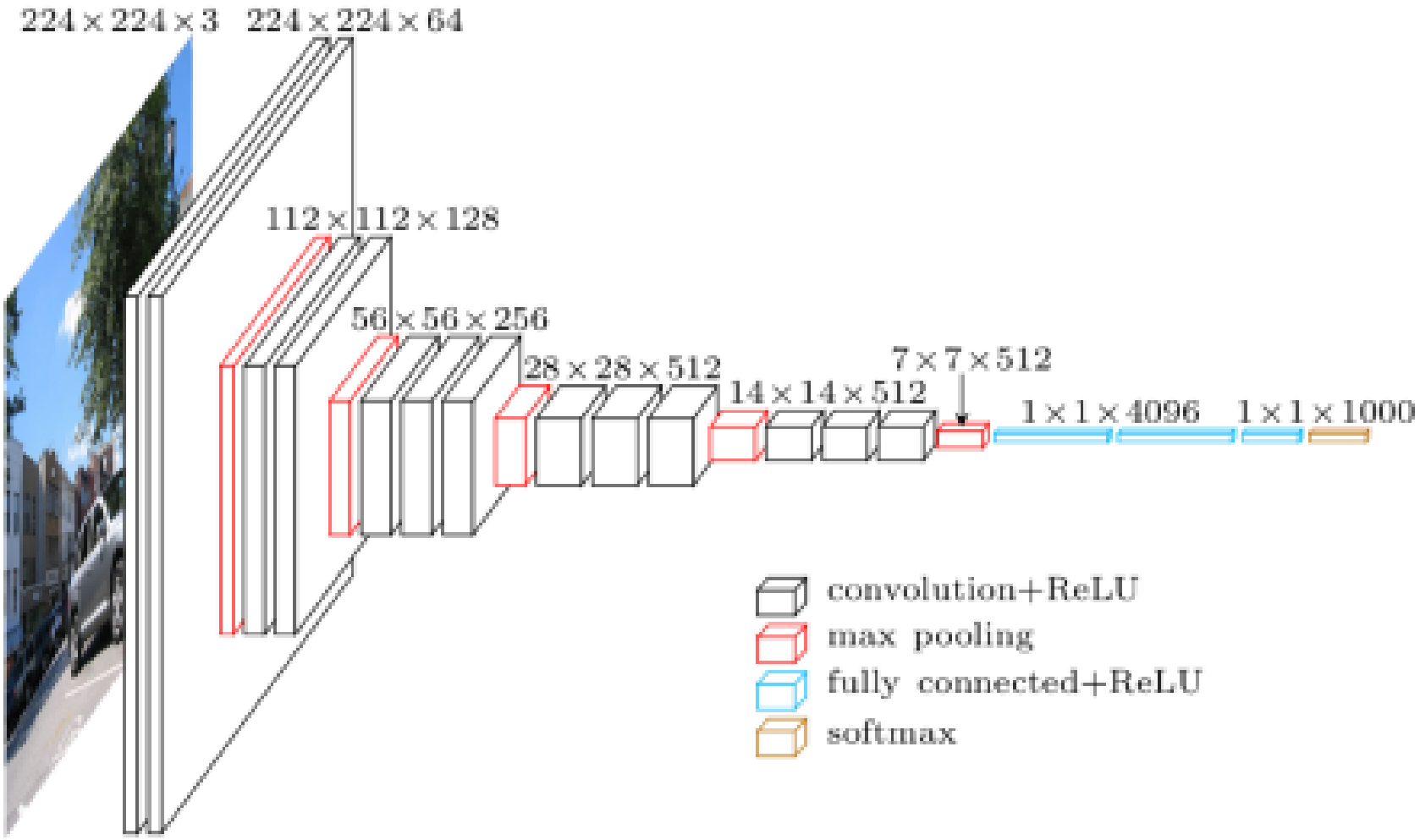


Image credit: https://www.cs.toronto.edu/~frossard/post/vgg16/

VGG19

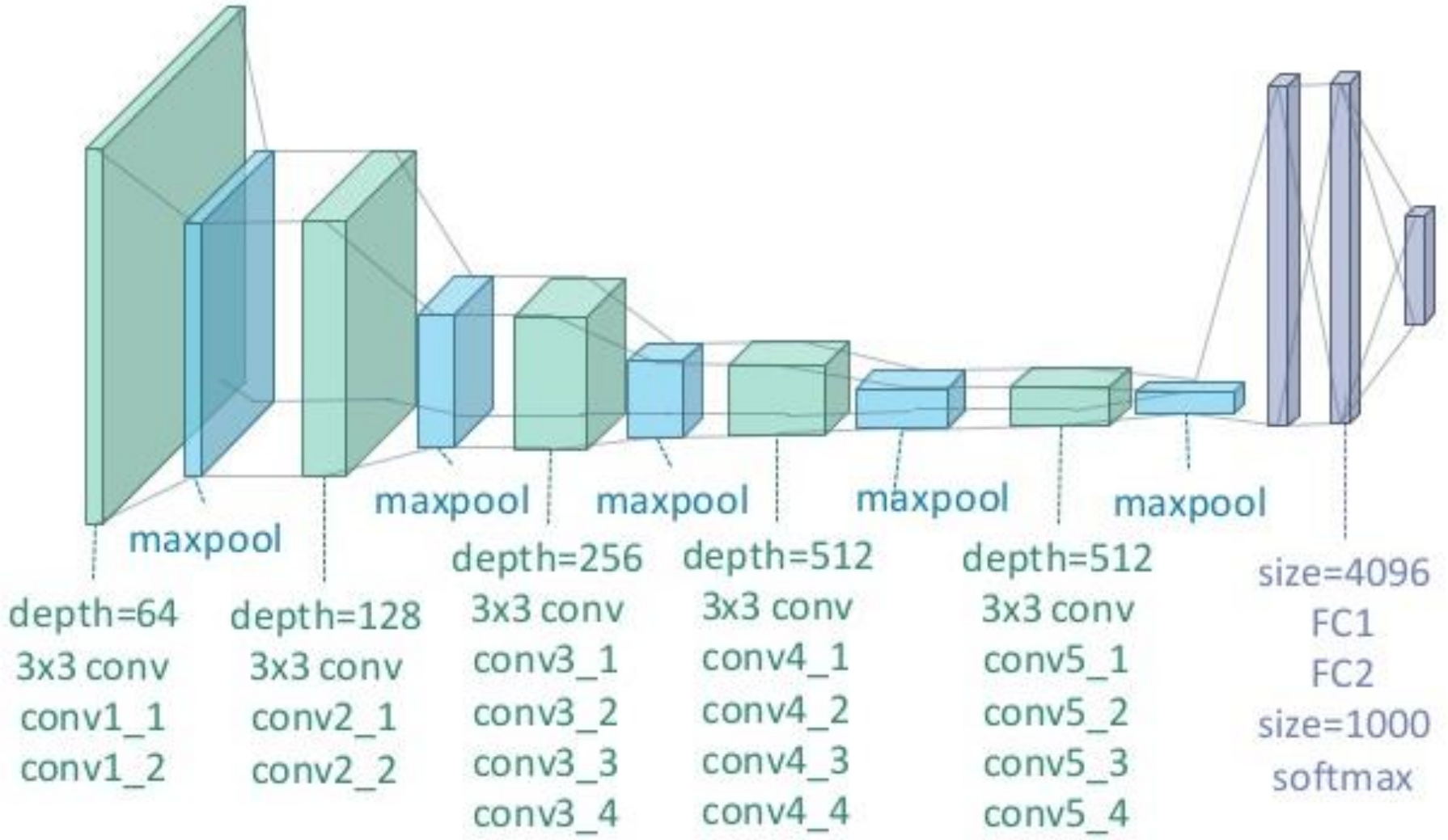


Image credit: https://www.slideshare.net/ckmarkohchang/applied-deep-learning-1103-convolutional-neural-networks

VGG11,13,16,19

LRN = Local Response Normalization

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

- ConvNet configurations (columns). The depth increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold).
- Convolutional layer parameters:"conv - receptive field size-number of channels".
- The ReLU activation function is not shown for brevity.

Image Classification using VGG19

```
model = Sequential()
model.add(ZeroPadding2D((1,1),input_shape=(3,224,224)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))
```

```
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1000, activation='softmax'))
```

```
model.load_weights("./vgg19_weights.h5")
```

VGG19 Parameters (Part 1)

	Layer (type)	Output Shape	Param #	
	zeropadding2d_1 (ZeroPadding2D)	(None, 3, 226, 226)	0	
'conv1_1', 'relu1_1'	convolution2d_1 (Convolution2D)	(None, 64, 224, 224)	1792	1792=(3*3*3+1)*64
	zeropadding2d_2 (ZeroPadding2D)	(None, 64, 226, 226)	0	
'conv1_2', 'relu1_2'	convolution2d_2 (Convolution2D)	(None, 64, 224, 224)	36928	36928=(64*3*3+1)*64
'pool1'	maxpooling2d_1 (MaxPooling2D)	(None, 64, 112, 112)	0	
	zeropadding2d_3 (ZeroPadding2D)	(None, 64, 114, 114)	0	
'conv2_1', 'relu2_1'	convolution2d_3 (Convolution2D)	(None, 128, 112, 112)	73856	73856=(64*3*3+1)*128
	zeropadding2d_4 (ZeroPadding2D)	(None, 128, 114, 114)	0	
'conv2_2', 'relu2_2'	convolution2d_4 (Convolution2D)	(None, 128, 112, 112)	147584	147584=(128*3*3+1)*128
'pool2'	maxpooling2d_2 (MaxPooling2D)	(None, 128, 56, 56)	0	
	zeropadding2d_5 (ZeroPadding2D)	(None, 128, 58, 58)	0	
'conv3_1', 'relu3_1'	convolution2d_5 (Convolution2D)	(None, 256, 56, 56)	295168	295168=(128*3*3+1)*256
	zeropadding2d_6 (ZeroPadding2D)	(None, 256, 58, 58)	0	
'conv3_2', 'relu3_2'	convolution2d_6 (Convolution2D)	(None, 256, 56, 56)	590080	590080=(256*3*3+1)*256
	zeropadding2d_7 (ZeroPadding2D)	(None, 256, 58, 58)	0	
'conv3_3', 'relu3_3'	convolution2d_7 (Convolution2D)	(None, 256, 56, 56)	590080	590080=(256*3*3+1)*256
	zeropadding2d_8 (ZeroPadding2D)	(None, 256, 58, 58)	0	
'conv3_4', 'relu3_4'	convolution2d_8 (Convolution2D)	(None, 256, 56, 56)	590080	590080=(256*3*3+1)*256

VGG19 (Part 2)

‘pool3'	maxpooling2d_3 (MaxPooling2D)	(None, 256, 28, 28)	0	
	zeropadding2d_9 (ZeroPadding2D)	(None, 256, 30, 30)	0	
'conv4_1', 'relu4_1'	convolution2d_9 (Convolution2D)	(None, 512, 28, 28)	1180160	1180160=(256*3*3+1)*512
	zeropadding2d_10 (ZeroPadding2D)	(None, 512, 30, 30)	0	
'conv4_2', 'relu4_2'	convolution2d_10 (Convolution2D)	(None, 512, 28, 28)	2359808	2359808=(512*3*3+1)*512
	zeropadding2d_11 (ZeroPadding2D)	(None, 512, 30, 30)	0	
'conv4_3', 'relu4_3'	convolution2d_11 (Convolution2D)	(None, 512, 28, 28)	2359808	2359808=(512*3*3+1)*512
	zeropadding2d_12 (ZeroPadding2D)	(None, 512, 30, 30)	0	
'conv4_4', 'relu4_4'	convolution2d_12 (Convolution2D)	(None, 512, 28, 28)	2359808	2359808=(512*3*3+1)*512
‘pool4'	maxpooling2d_4 (MaxPooling2D)	(None, 512, 14, 14)	0	
	zeropadding2d_13 (ZeroPadding2D)	(None, 512, 16, 16)	0	
'conv5_1', 'relu5_1'	convolution2d_13 (Convolution2D)	(None, 512, 14, 14)	2359808	2359808=(512*3*3+1)*512
	zeropadding2d_14 (ZeroPadding2D)	(None, 512, 16, 16)	0	
'conv5_2', 'relu5_2'	convolution2d_14 (Convolution2D)	(None, 512, 14, 14)	2359808	2359808=(512*3*3+1)*512
	zeropadding2d_15 (ZeroPadding2D)	(None, 512, 16, 16)	0	
'conv5_3', 'relu5_3'	convolution2d_15 (Convolution2D)	(None, 512, 14, 14)	2359808	2359808=(512*3*3+1)*512
	zeropadding2d_16 (ZeroPadding2D)	(None, 512, 16, 16)	0	
'conv5_4', 'relu5_4'	convolution2d_16 (Convolution2D)	(None, 512, 14, 14)	2359808	2359808=(512*3*3+1)*512
‘pool5'	maxpooling2d_5 (MaxPooling2D)	(None, 512, 7, 7)	0	

VGG19 (Part 3)

	flatten_1 (Flatten)	(None, 25088)	0	25088=512*7*7
'FC1'	dense_1 (Dense)	(None, 4096)	102764544	102764544=(25088+1)*4096
	dropout_1 (Dropout)	(None, 4096)	0	
'FC2'	dense_2 (Dense)	(None, 4096)	16781312	16781312=(4096+1)*4096
	dropout_2 (Dropout)	(None, 4096)	0	
'softmax'	dense_3 (Dense)	(None, 1000)	4097000	4097000=(4096+1)*1000

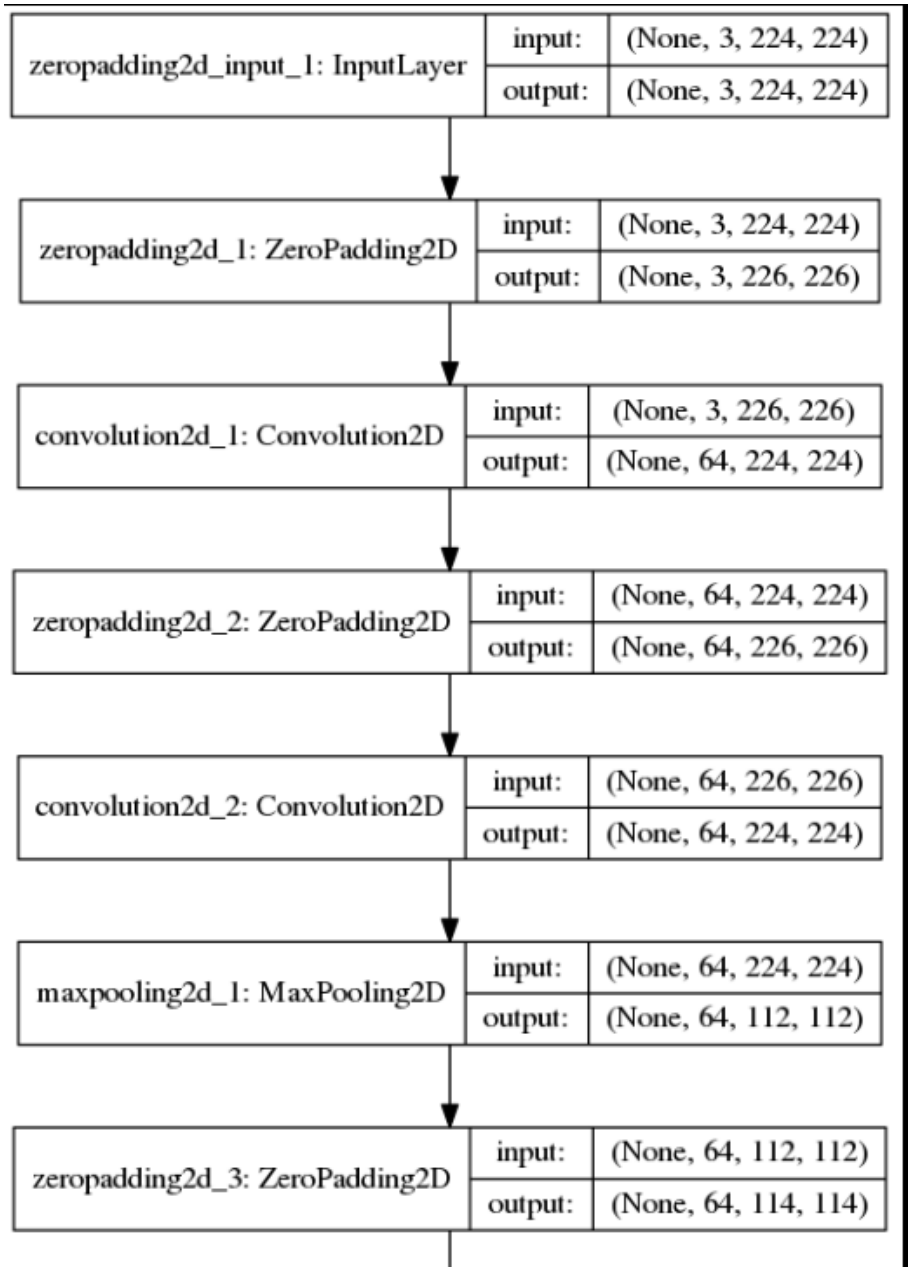
Total params: 143,667,240
Trainable params: 143,667,240
Non-trainable params: 0

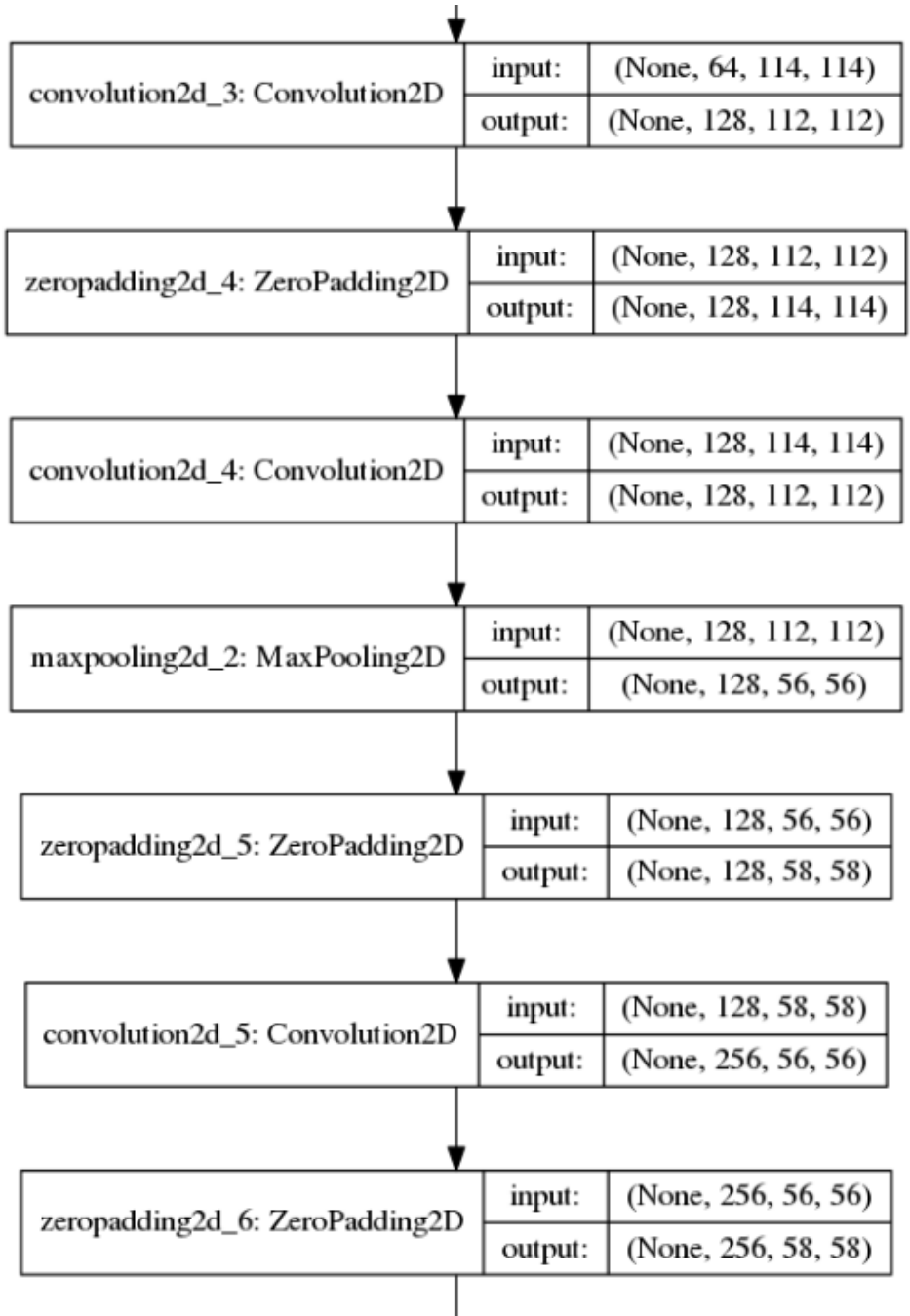
Softmax:

$$K = 1000,\ \mathbf{w}_j, \mathbf{x} \in \mathbb{R}^{(4096+1)},\ j \in 1, \ldots, K$$

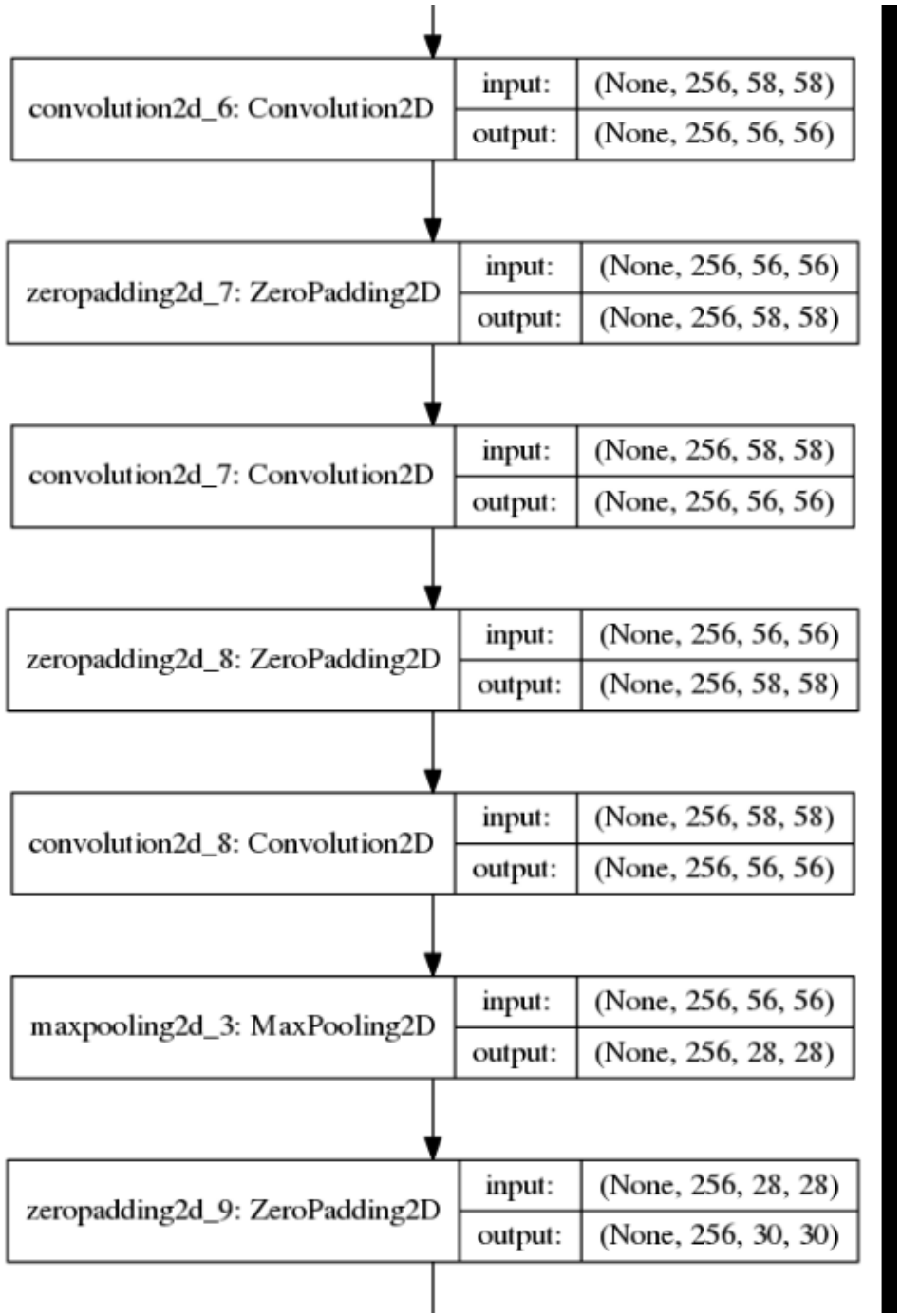
$$P(y = j|\mathbf{x}) = \frac{e^{\mathbf{x}^\top \mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\top \mathbf{w}_k}}$$

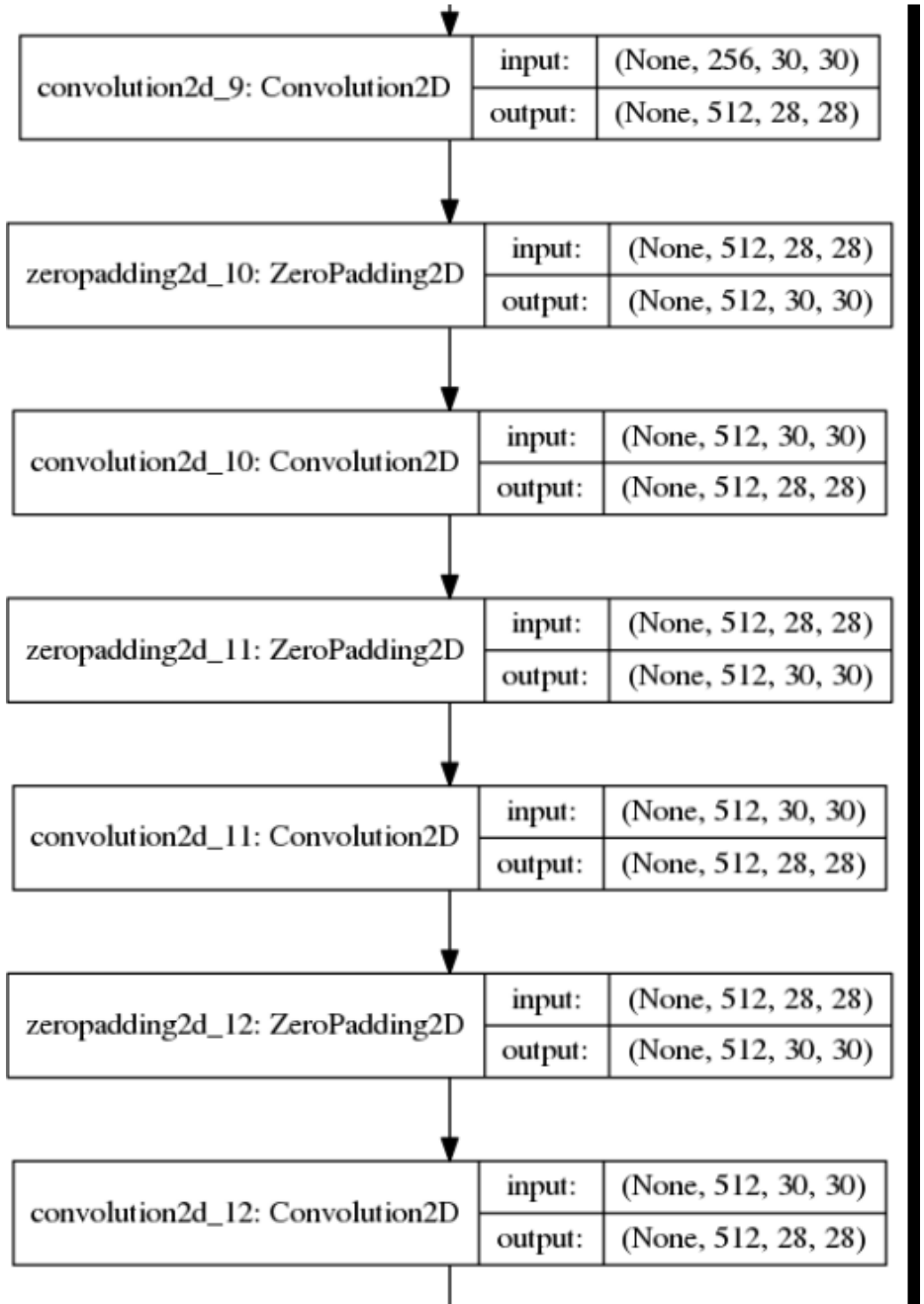
VGG19 (Part 1)


zeropadding2d_input_1: InputLayer
input: (None, 3, 224, 224)
output: (None, 3, 224, 224)
zeropadding2d_1: ZeroPadding2D
input: (None, 3, 224, 224)
output: (None, 3, 226, 226)
convolution2d_1: Convolution2D
input: (None, 3, 226, 226)
output: (None, 64, 224, 224)
zeropadding2d_2: ZeroPadding2D
input: (None, 64, 224, 224)
output: (None, 64, 226, 226)
convolution2d_2: Convolution2D
input: (None, 64, 226, 226)
output: (None, 64, 224, 224)
maxpooling2d_1: MaxPooling2D
input: (None, 64, 224, 224)
output: (None, 64, 112, 112)
zeropadding2d_3: ZeroPadding2D
input: (None, 64, 112, 112)
output: (None, 64, 114, 114)

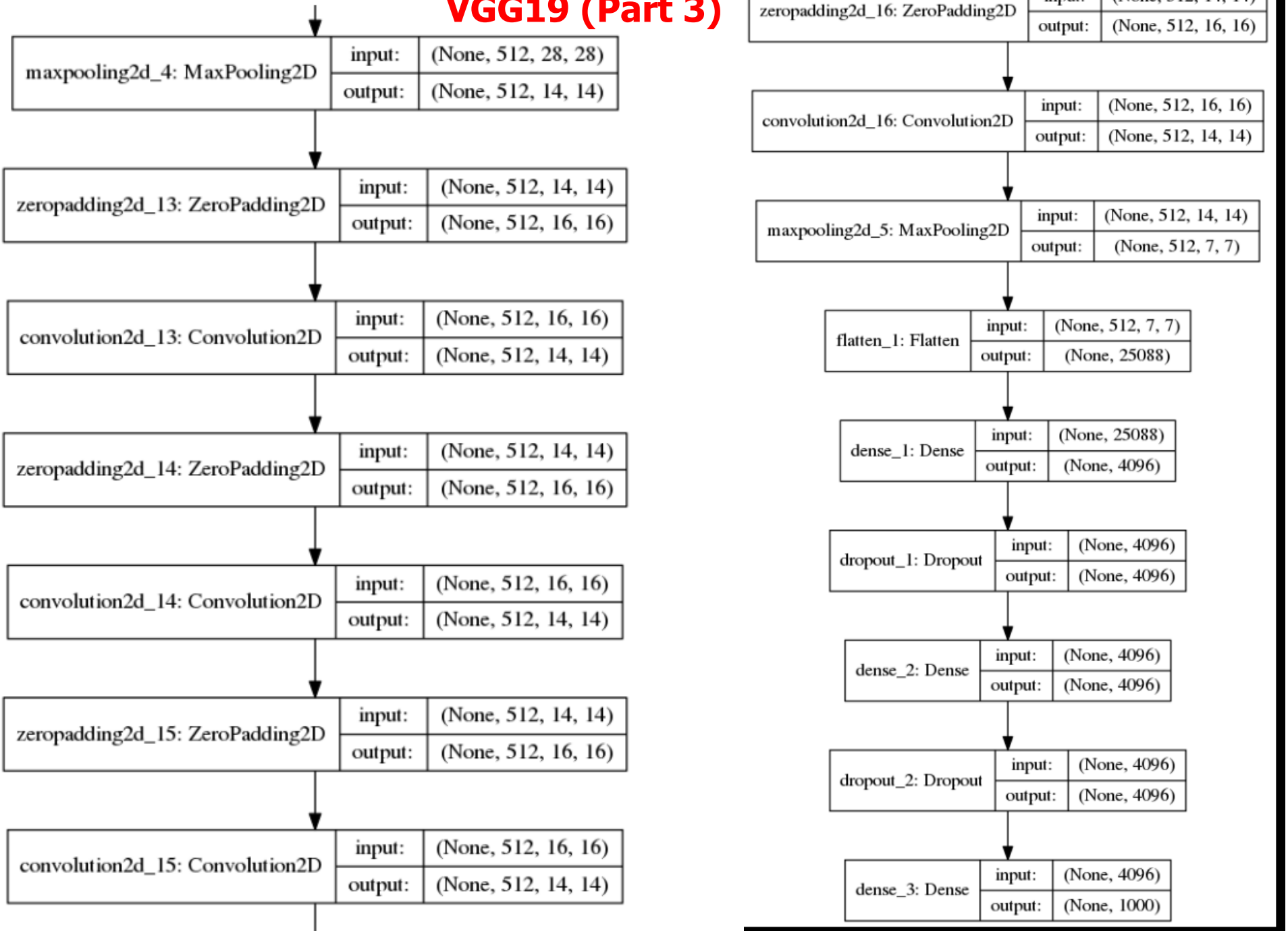

convolution2d_3: Convolution2D
input: (None, 64, 114, 114)
output: (None, 128, 112, 112)
zeropadding2d_4: ZeroPadding2D
input: (None, 128, 112, 112)
output: (None, 128, 114, 114)
convolution2d_4: Convolution2D
input: (None, 128, 114, 114)
output: (None, 128, 112, 112)
maxpooling2d_2: MaxPooling2D
input: (None, 128, 112, 112)
output: (None, 128, 56, 56)
zeropadding2d_5: ZeroPadding2D
input: (None, 128, 56, 56)
output: (None, 128, 58, 58)
convolution2d_5: Convolution2D
input: (None, 128, 58, 58)
output: (None, 256, 56, 56)
zeropadding2d_6: ZeroPadding2D
input: (None, 256, 56, 56)
output: (None, 256, 58, 58)

VGG19 (Part 2)


convolution2d_6: Convolution2D
input: (None, 256, 58, 58)
output: (None, 256, 56, 56)
zeropadding2d_7: ZeroPadding2D
input: (None, 256, 56, 56)
output: (None, 256, 58, 58)
convolution2d_7: Convolution2D
input: (None, 256, 58, 58)
output: (None, 256, 56, 56)
zeropadding2d_8: ZeroPadding2D
input: (None, 256, 56, 56)
output: (None, 256, 58, 58)
convolution2d_8: Convolution2D
input: (None, 256, 58, 58)
output: (None, 256, 56, 56)
maxpooling2d_3: MaxPooling2D
input: (None, 256, 56, 56)
output: (None, 256, 28, 28)
zeropadding2d_9: ZeroPadding2D
input: (None, 256, 28, 28)
output: (None, 256, 30, 30)


convolution2d_9: Convolution2D
input: (None, 256, 30, 30)
output: (None, 512, 28, 28)
zeropadding2d_10: ZeroPadding2D
input: (None, 512, 28, 28)
output: (None, 512, 30, 30)
convolution2d_10: Convolution2D
input: (None, 512, 30, 30)
output: (None, 512, 28, 28)
zeropadding2d_11: ZeroPadding2D
input: (None, 512, 28, 28)
output: (None, 512, 30, 30)
convolution2d_11: Convolution2D
input: (None, 512, 30, 30)
output: (None, 512, 28, 28)
zeropadding2d_12: ZeroPadding2D
input: (None, 512, 28, 28)
output: (None, 512, 30, 30)
convolution2d_12: Convolution2D
input: (None, 512, 30, 30)
output: (None, 512, 28, 28)

VGG19 (Part 3)


maxpooling2d_4: MaxPooling2D
input: (None, 512, 28, 28)
output: (None, 512, 14, 14)
zeropadding2d_13: ZeroPadding2D
input: (None, 512, 14, 14)
output: (None, 512, 16, 16)
convolution2d_13: Convolution2D
input: (None, 512, 16, 16)
output: (None, 512, 14, 14)
zeropadding2d_14: ZeroPadding2D
input: (None, 512, 14, 14)
output: (None, 512, 16, 16)
convolution2d_14: Convolution2D
input: (None, 512, 16, 16)
output: (None, 512, 14, 14)
zeropadding2d_15: ZeroPadding2D
input: (None, 512, 14, 14)
output: (None, 512, 16, 16)
convolution2d_15: Convolution2D
input: (None, 512, 16, 16)
output: (None, 512, 14, 14)
zeropadding2d_16: ZeroPadding2D
input: (None, 512, 14, 14)
output: (None, 512, 16, 16)
convolution2d_16: Convolution2D
input: (None, 512, 16, 16)
output: (None, 512, 14, 14)
maxpooling2d_5: MaxPooling2D
input: (None, 512, 14, 14)
output: (None, 512, 7, 7)
flatten_1: Flatten
input: (None, 512, 7, 7)
output: (None, 25088)
dense_1: Dense
input: (None, 25088)
output: (None, 4096)
dropout_1: Dropout
input: (None, 4096)
output: (None, 4096)
dense_2: Dense
input: (None, 4096)
output: (None, 4096)
dropout_2: Dropout
input: (None, 4096)
output: (None, 4096)
dense_3: Dense
input: (None, 4096)
output: (None, 1000)

VGG Results

ILSVRC-2012 dataset (which was used for ILSVRC 2012–2014 challenges). The dataset includes images of 1000 classes, and is split into three sets: training (1.3M images), validation (50K images), and testing (100K images with held-out class labels).

Table 3: **ConvNet performance at a single test scale.**

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	**25.5**	**8.0**

VGG Results

0.4170 - n01871265 tusker
0.2178 - n02504458 African elephant, Loxodonta africana
0.1055 - n01704323 triceratops
0.0496 - n02504013 Indian elephant, Elephas maximus
0.0374 - n01768244 trilobite
0.0187 - n01817953 African grey, African gray, Psittacus erithacus
0.0108 - n02398521 hippopotamus, hippo, river horse, Hippopotamus amphibius
0.0095 - n02056570 king penguin, Aptenodytes patagonica
0.0090 - n02071294 killer whale, killer, orca, grampus, sea wolf, Orcinus orca
0.0068 - n01855672 goose

VGG Results

0.7931 - n04335435 streetcar, tram, tramcar, trolley, trolley car
0.1298 - n04487081 trolleybus, trolley coach, trackless trolley
0.0321 - n03895866 passenger car, coach, carriage
0.0135 - n03769881 minibus
0.0103 - n03902125 pay-phone, pay-station
0.0054 - n03272562 electric locomotive
0.0012 - n03496892 harvester, reaper
0.0011 - n03126707 crane
0.0010 - n04465501 tractor
0.0010 - n03417042 garbage truck, dustcart

Video Classification

https://www.youtube.com/watch?v=qrzQ_AB1DZk

Andrej Karpathy, CVPR 2014

Style Transfer

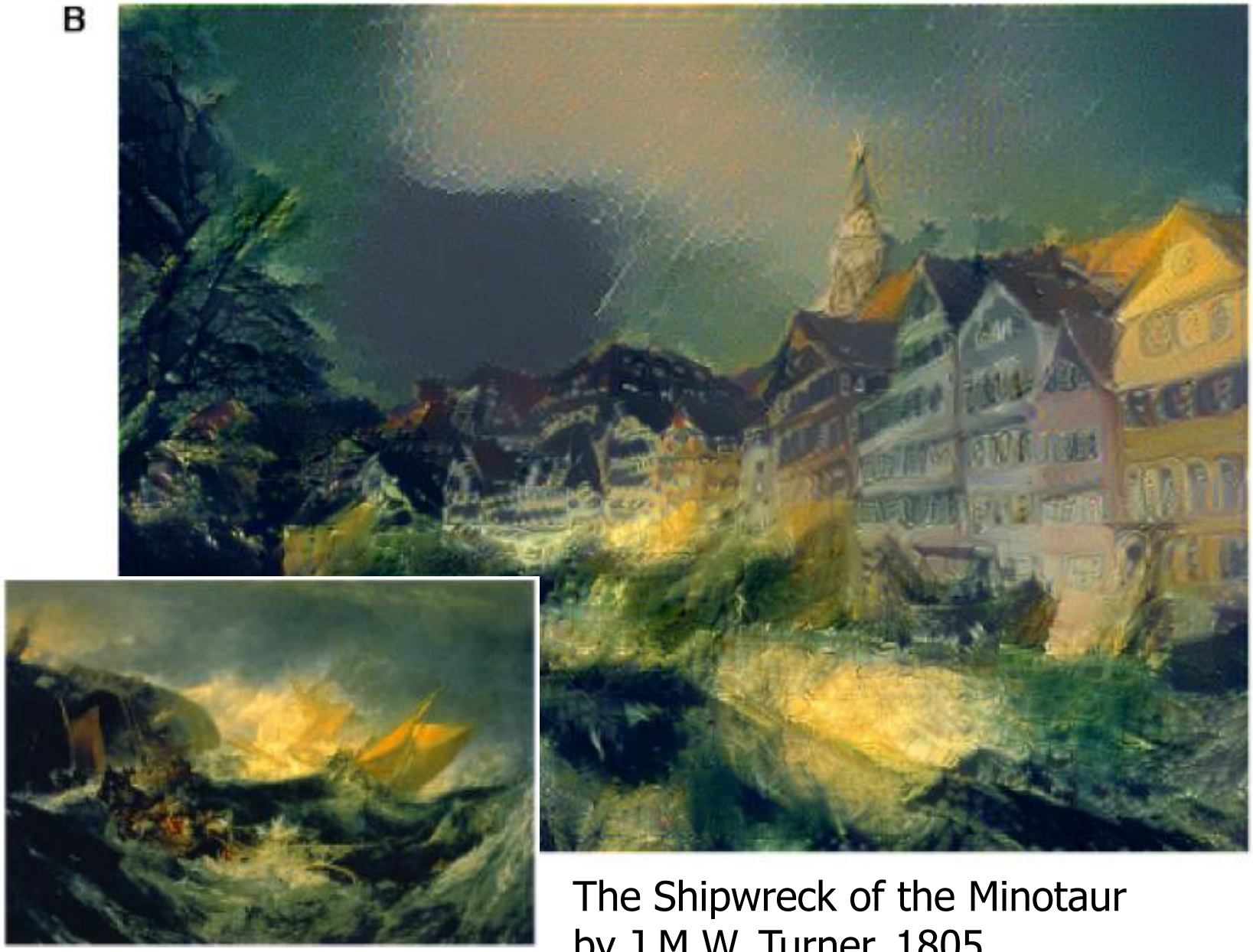
Style Transfer

Gatys, Ecker, Bethge: A Neural Algorithm of Artistic Style

Style Transfer, Relevant Papers

- Image Style Transfer Using Convolutional Neural Networks Leon A. Gatys, Alexander S. Ecker, Matthias Bethge
- Combining Markov Random Fields and Convolutional Neural Networks for Image Synthesis, Chuan Li, Michael Wand

Style Transfer



The Shipwreck of the Minotaur
by J.M.W. Turner, 1805.

Style Transfer

The Starry Night by Vincent van Gogh, 1889.

Style Transfer

Der Schrei by Edvard Munch, 1893

Style Transfer with Keras and Tensorflow

https://github.com/bapoczos/StyleTransfer/blob/master/style_transfer_keras_tensorflow.ipynb

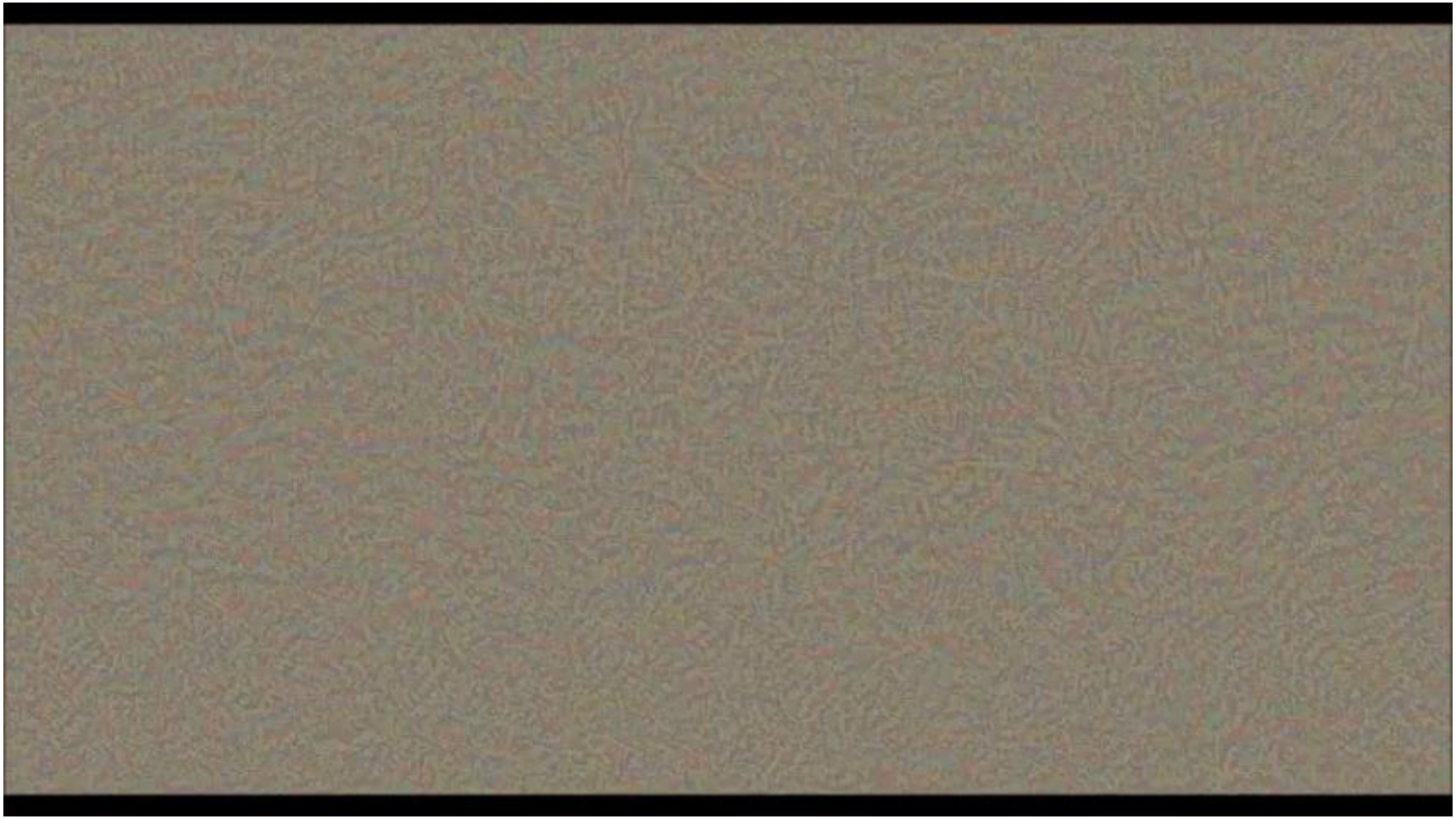
Content Image

Content image size: (1, 450, 845, 3)

Style Image

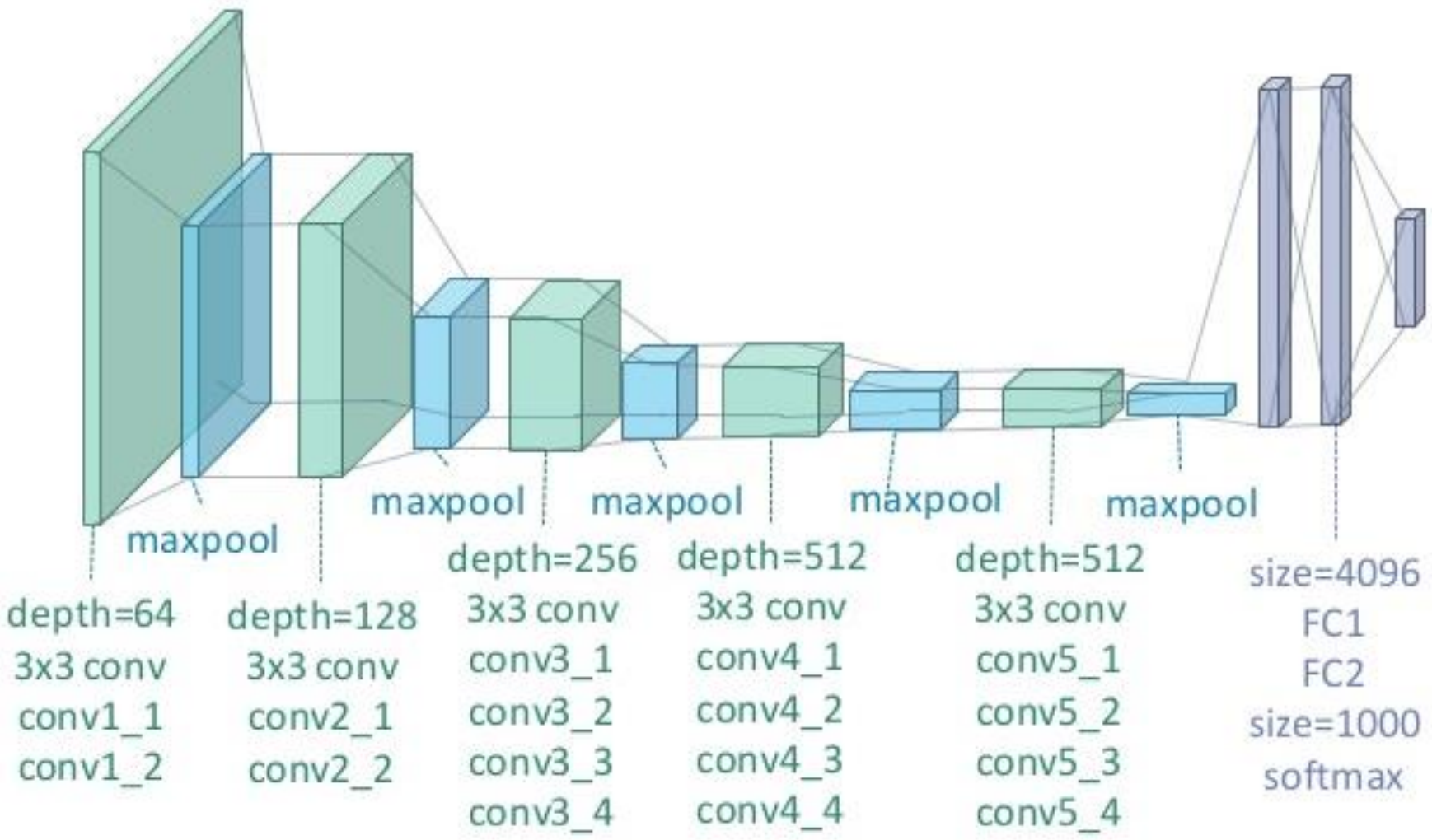
Style image size: (1, 507, 640, 3)

Style Transfer



Style Transformed Image

Style Transform with VGG 19

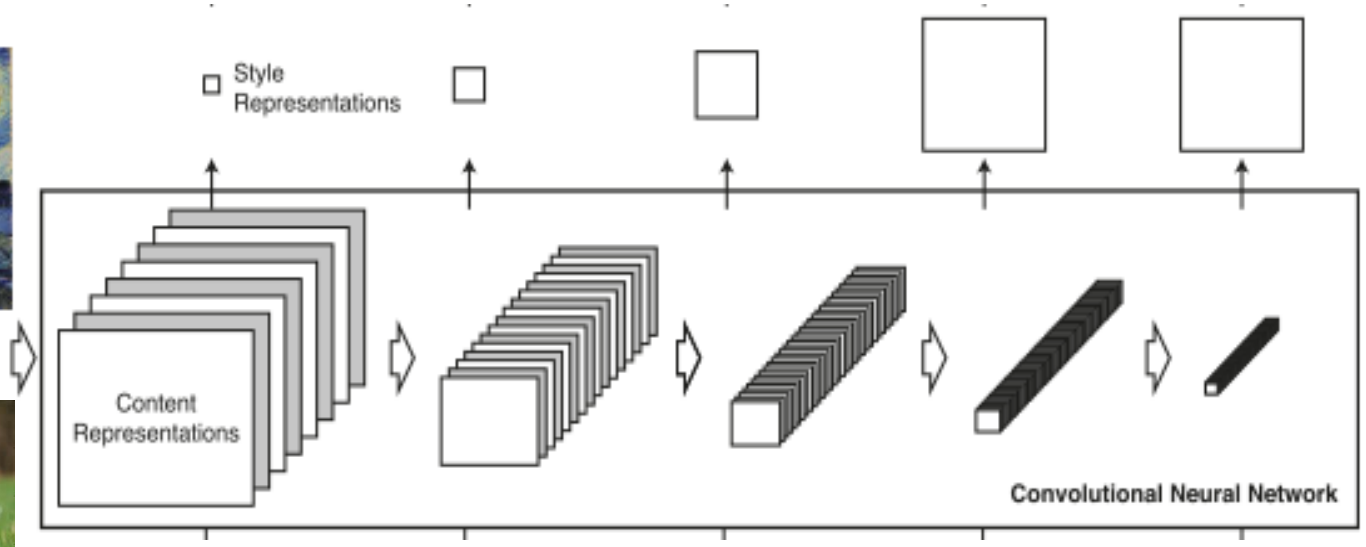


Style Transfer

Algorithm:

1) Calculate content features (set of tensors which are neuron activities in the hidden layers)
2) Calculate style features (set of Gram matrices which are correlations between neuron activities in the hidden layers)
3) Create a new image that matches both the content activities and the style Gram matrices

Input image



Style Transform: Content features

We will use VGG19 without the final maxpool, Flat, Dense, Dropout, and Softmax Layers

Select CONTENT_LAYERS

For example:

{'conv1_1', 'conv2_1', 'conv4_1', 'conv4_2'}
or just simply {'relu4_2'}

Size of relu4_2', (1, 57, 106, 512)

[57 =450 /8,106 = 845/8
8 =2^3 Size decrease after 3 maxpool]

The elements of the (1, 57, 106, 512) tensor are the content features

19 weight layers
conv3-64
conv3-64
conv3-128
conv3-128
conv3-256
conv3-256
conv3-256
conv3-256
conv3-512
conv3-512
conv3-512
conv3-512
conv3-512
conv3-512
conv3-512
conv3-512

Layers:

1) 'conv1_1', 'relu1_1',
2) 'conv1_2', 'relu1_2', 'pool1',
3) 'conv2_1', 'relu2_1',
4) 'conv2_2', 'relu2_2', 'pool2',
5) 'conv3_1', 'relu3_1',
6) 'conv3_2', 'relu3_2',
7) 'conv3_3', 'relu3_3',
8) 'conv3_4', 'relu3_4', 'pool3',
9) 'conv4_1', 'relu4_1',
10) 'conv4_2', 'relu4_2',
11) 'conv4_3', 'relu4_3',
12) 'conv4_4', 'relu4_4', 'pool4',
13) 'conv5_1', 'relu5_1',
14) 'conv5_2', 'relu5_2',
15) 'conv5_3', 'relu5_3',
16) 'conv5_4', 'relu5_4'

Style Transform: Calculating Style Gram matrices

Select STYLE_LAYERS

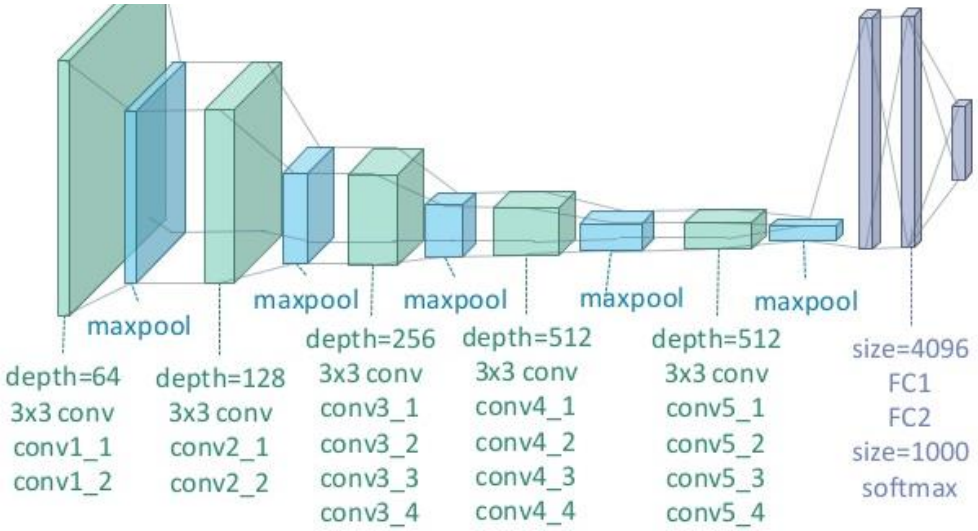
For example:

{'conv3_1','conv5_1'}

Or

{'relu1_1', 'relu2_1', 'relu3_1', 'relu4_1', 'relu5_1'}

Style image size: (1, 507, 640, 3)



'relu1_1' shape: (1, 507, 640, 64)
reshaped: (324480, 64)
gram matrix shape: (64, 64)

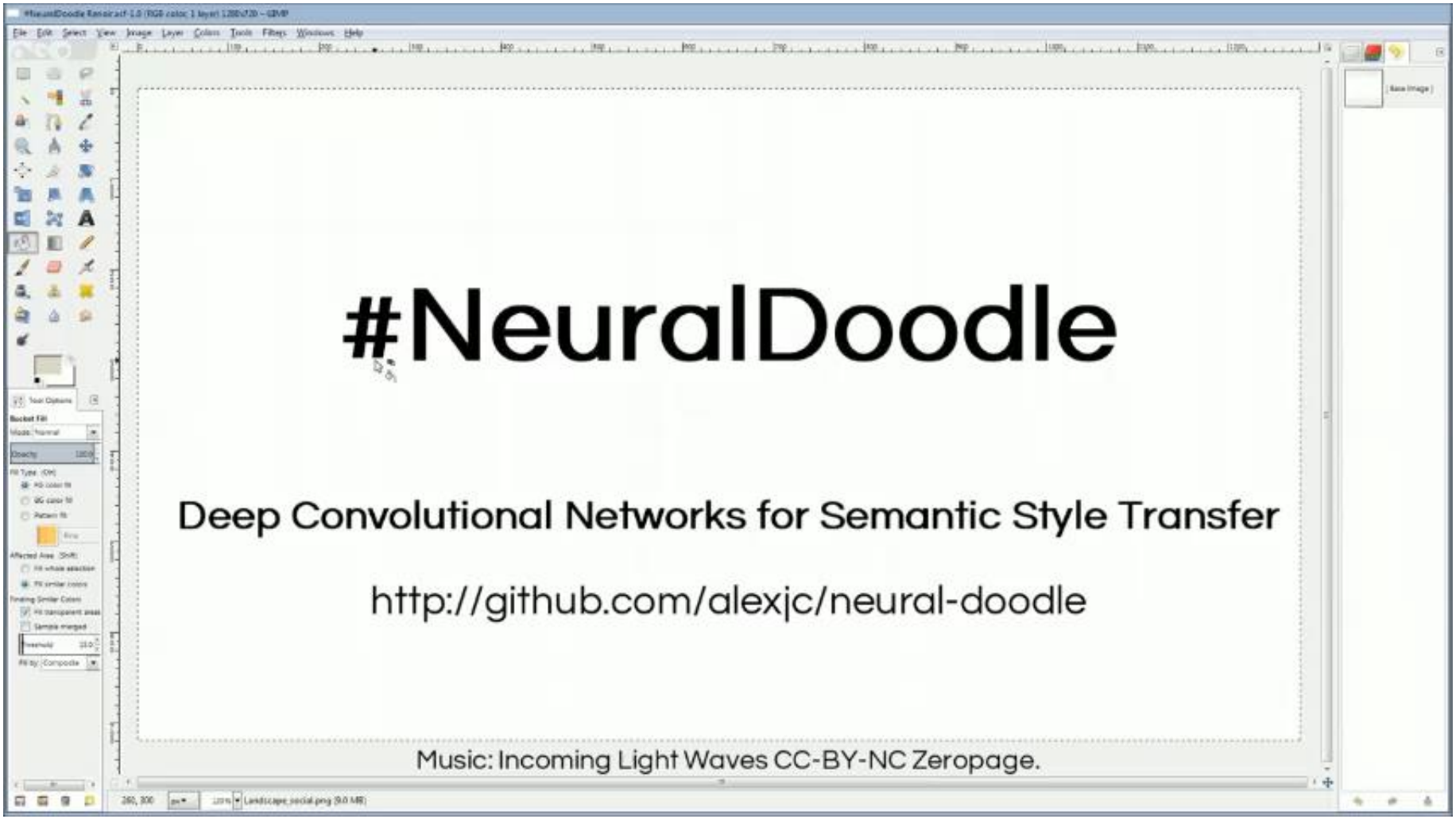
'relu2_1' shape: (1, 254, 320, 128)
'reshaped: (81280, 128)
gram matrix shape: (128, 128)

'relu3_1' shape: (1, 127, 160, 256)
reshaped: (20320, 256)
gram matrix shape: (256, 256)

'relu4_1' shape: (1, 64, 80, 512)
reshaped: (5120, 512)
gram matrix shape: (512, 512)

'relu5_1' shape: (1, 32, 40, 512)
reshaped: (1280, 512)
gram matrix shape: (512, 512)

Style Transform: Neural Doodle



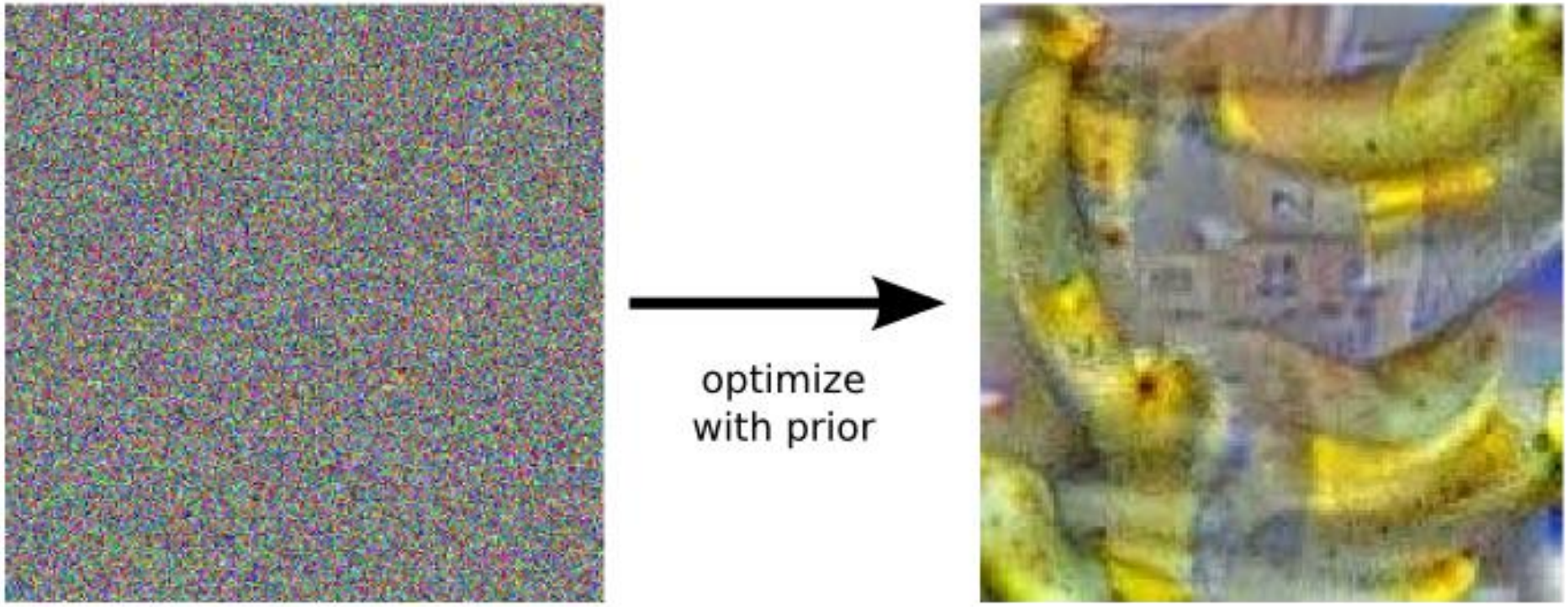
Style Transfer for Videos

https://www.youtube.com/watch?v=Khuj4ASldmU

Inception / Deep Dream

Tune the Inputs

Instead of tuning the neural network weights, keep them fixed (egVGG19 weights) and tune the input image of the network.



Starting from random noise, find the image that will maximize the probability of being classified as banana

Image credit: https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html

Tune the Inputs

Hartebeest

Measuring Cup

Ant

Starfish

Anemone Fish

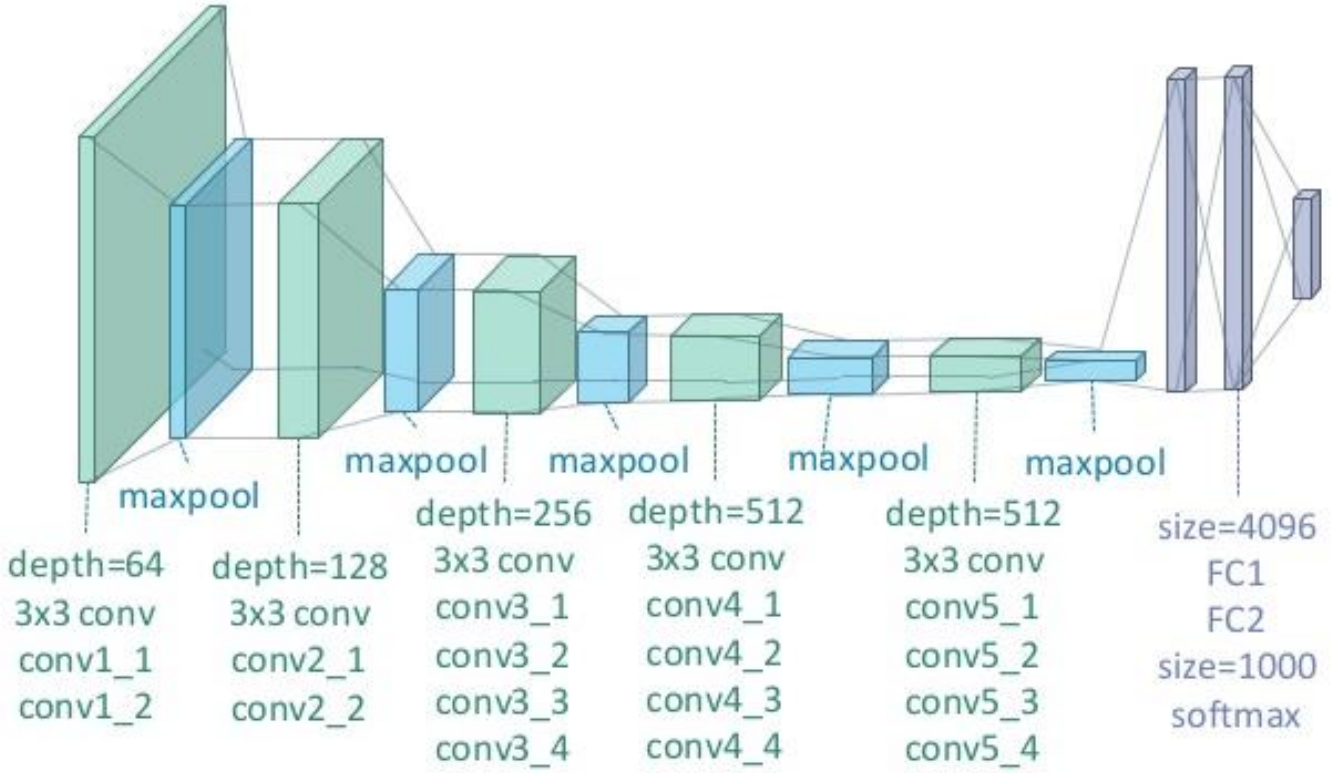
Banana

Parachute

Screw

Deep Dream

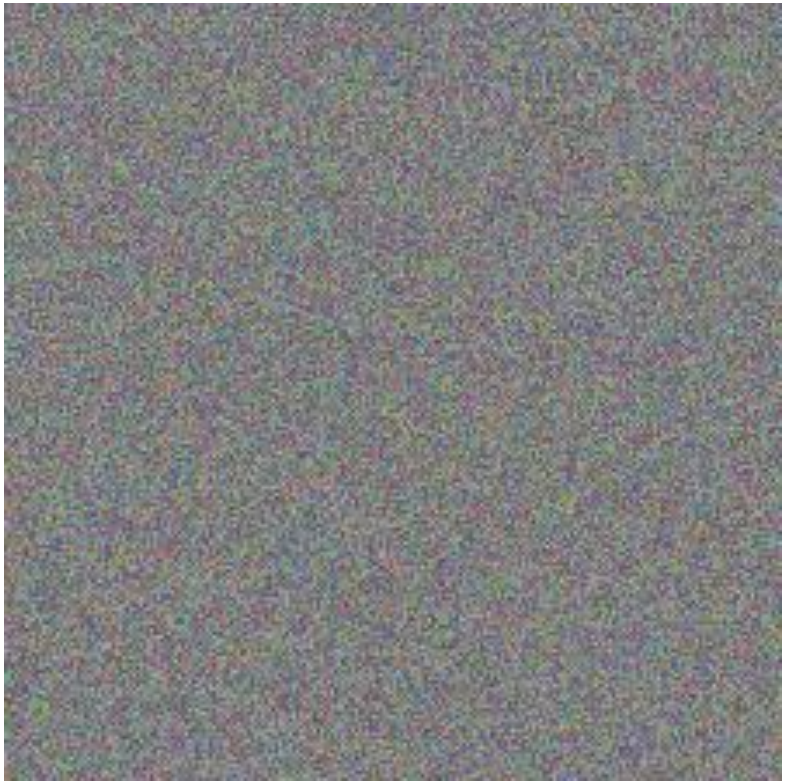
Deep Dream

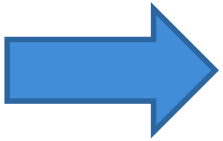


Goal: Find the image the maximizes the sum of the neuron activities on some selected channels of some selected layers

Deep Dream

layer = 'mixed4d_3x3_bottleneck_pre_relu' channel = 139





Deep Dream

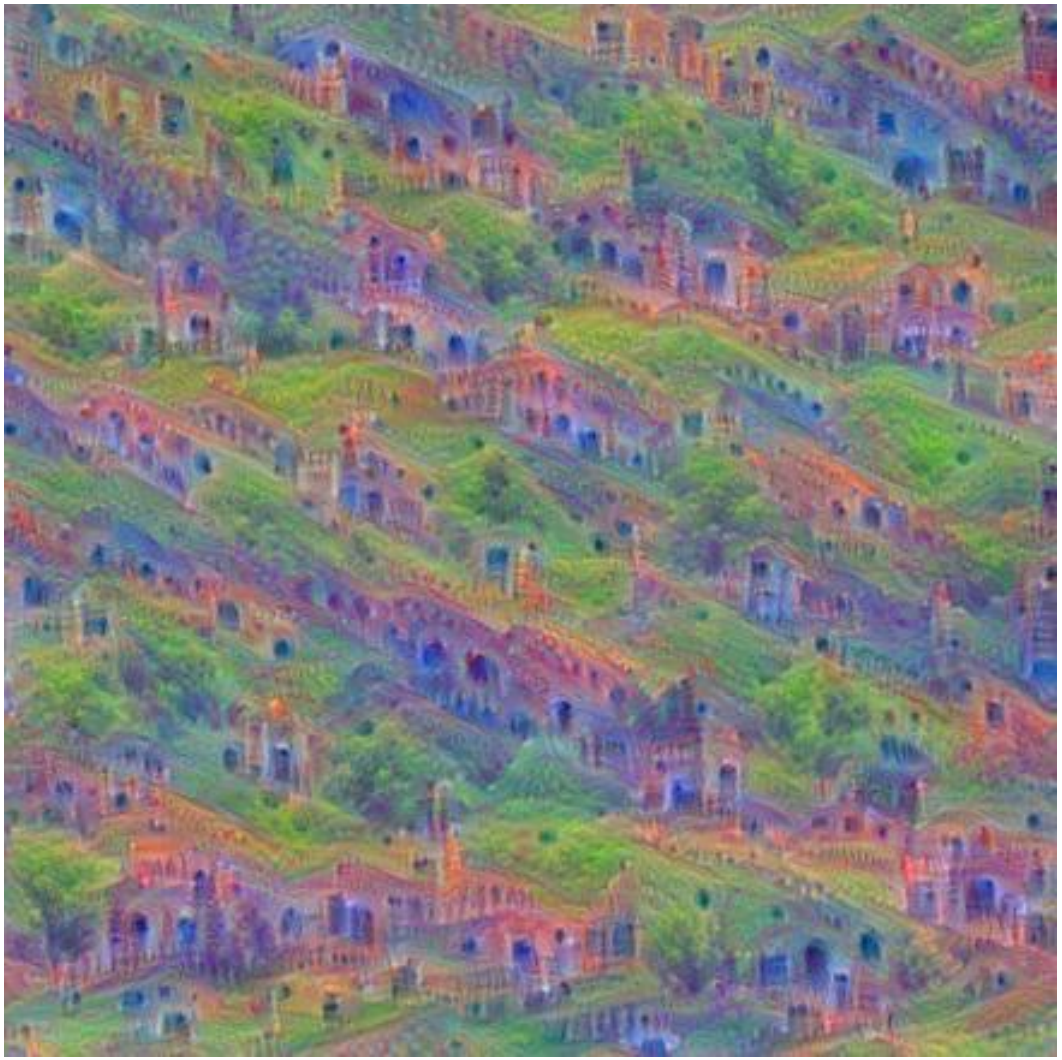
After multiscale + smoothing

Blur the image a little every iteration by suppressing the higher frequencies, so that the lower frequencies can catch up

Deep Dream

Let's try to visualize another channel from the same layer

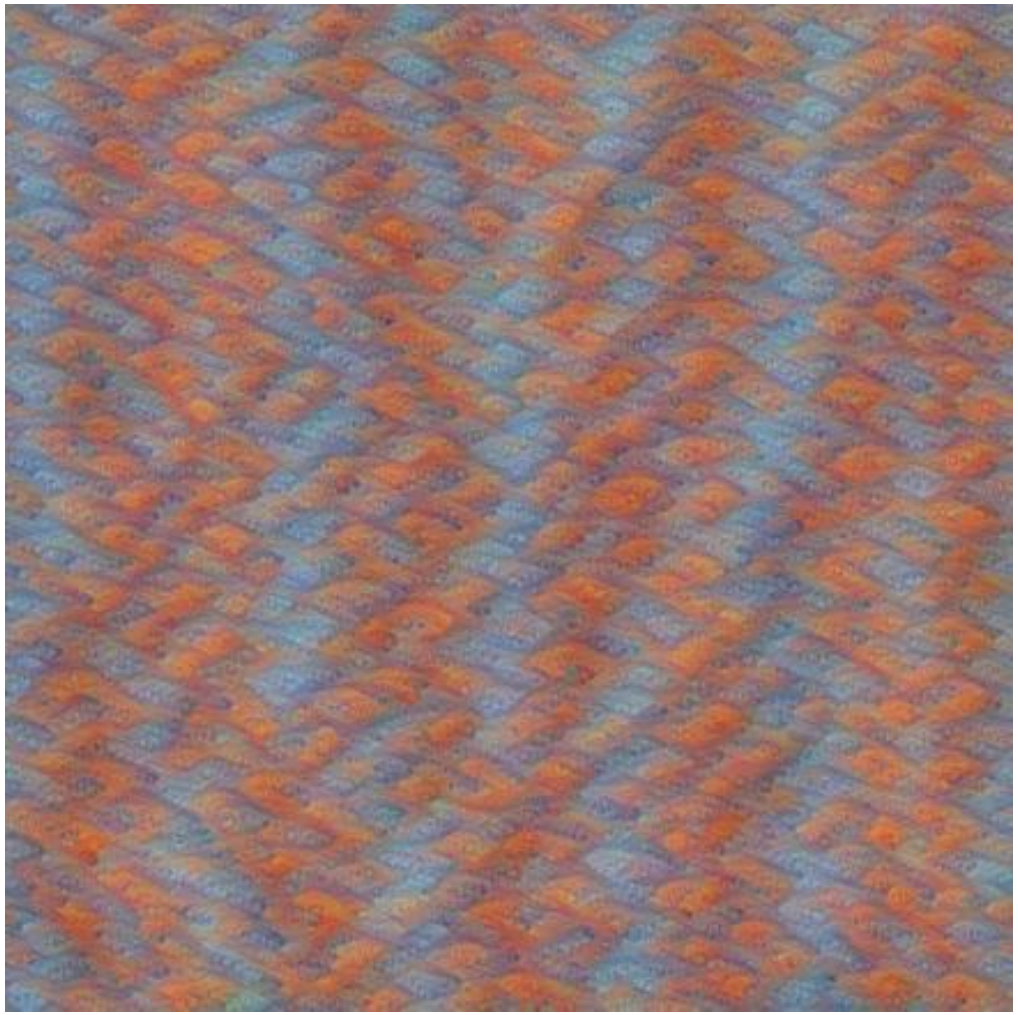
layer = 'mixed4d_3x3_bottleneck_pre_relu' channel = 65



Deep Dream

Lower layers produce features of lower complexity.

layer = 'mixed3b_1x1_pre_relu' channel = 121



Deep Dream

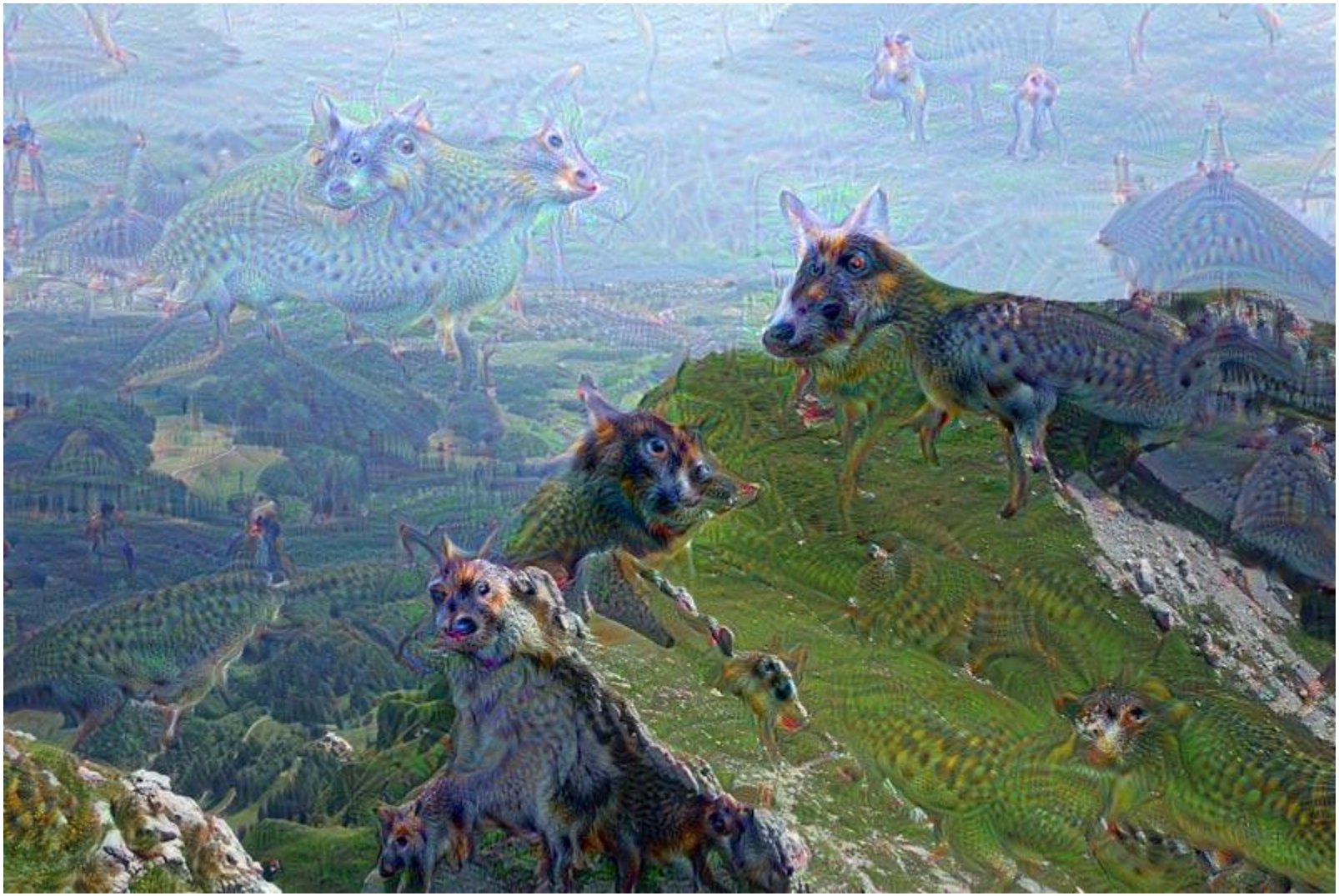
Optimizing a linear combination of features often gives a "mixture" pattern. (Channels 139 + 65)

https://github.com/bapoczos/deep-dream-tensorflow/blob/master/deepdream.ipynb

Deep Dream

Starting from an image instead of noise

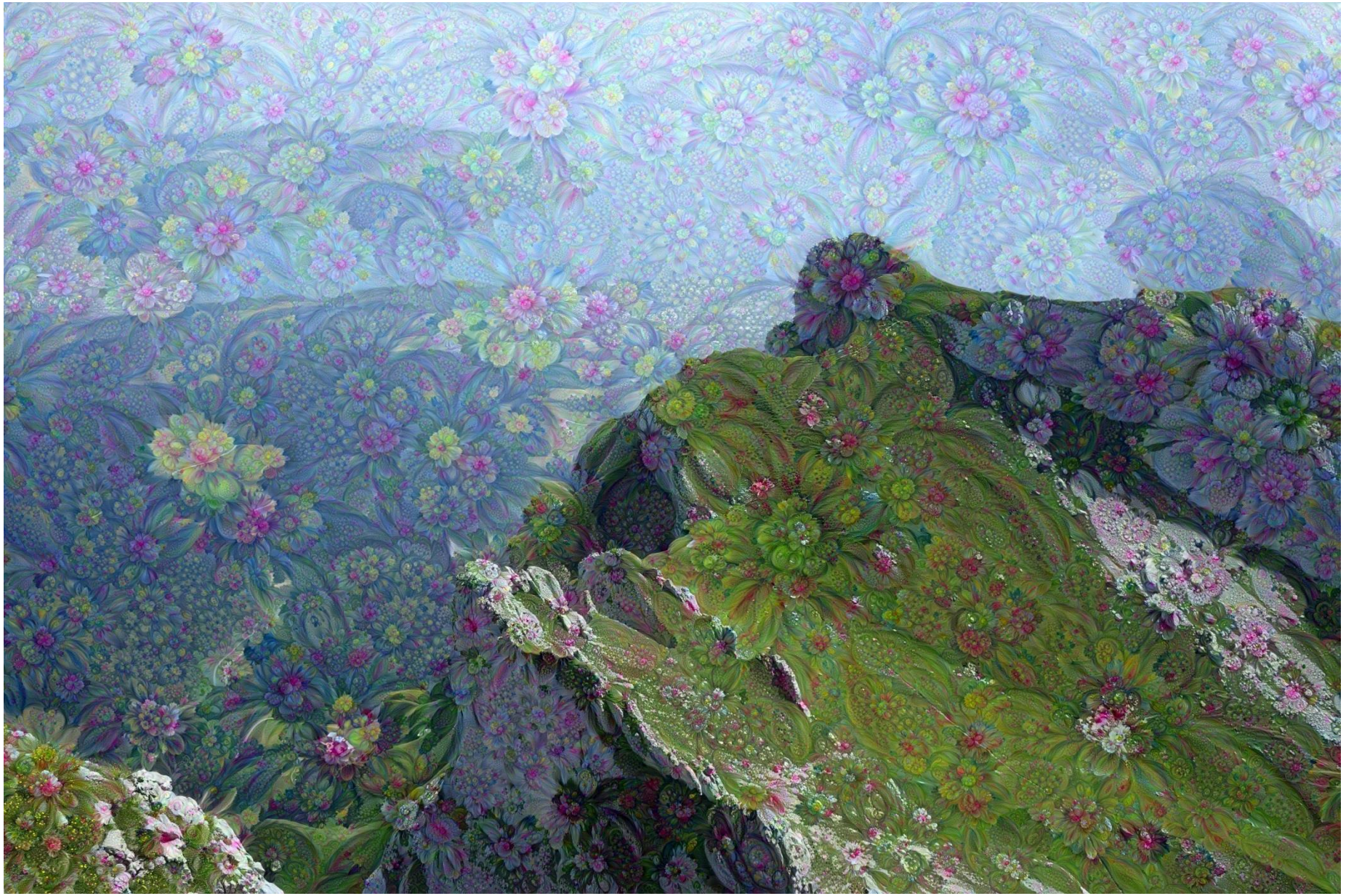
Deep Dream

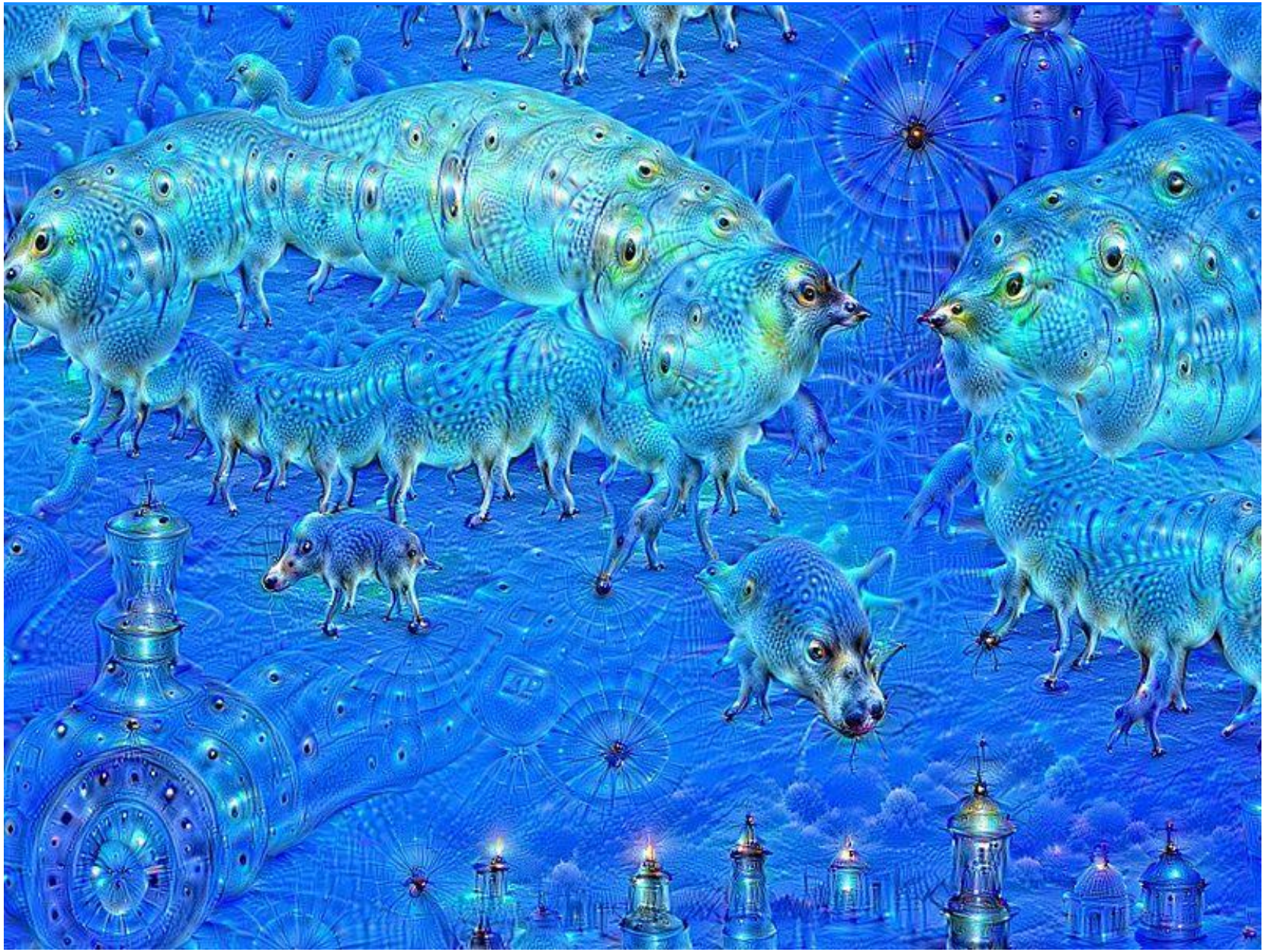


Maximizing the sum of squared activities on the ‘mixed4c’ layer

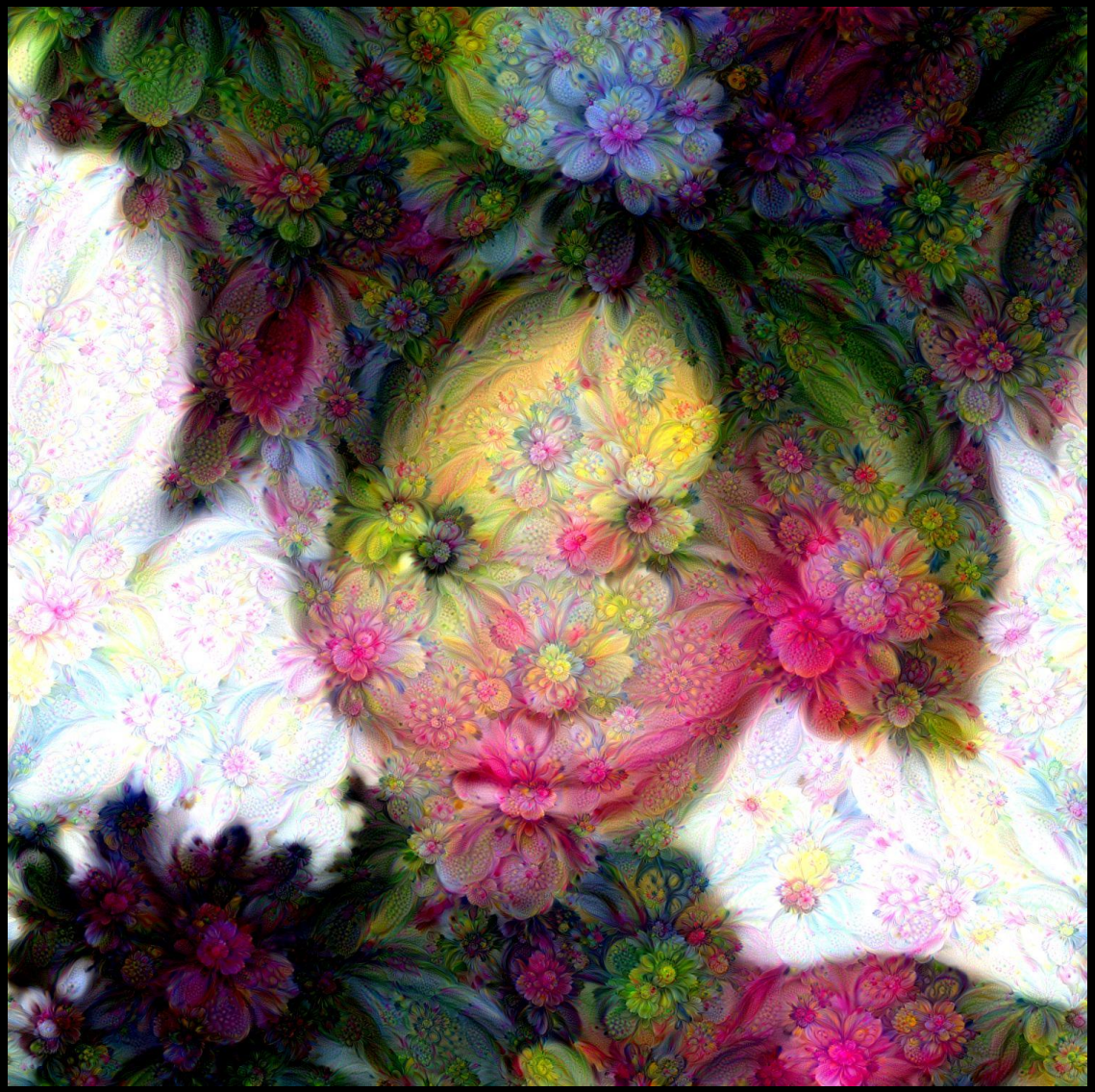
Channel 139:

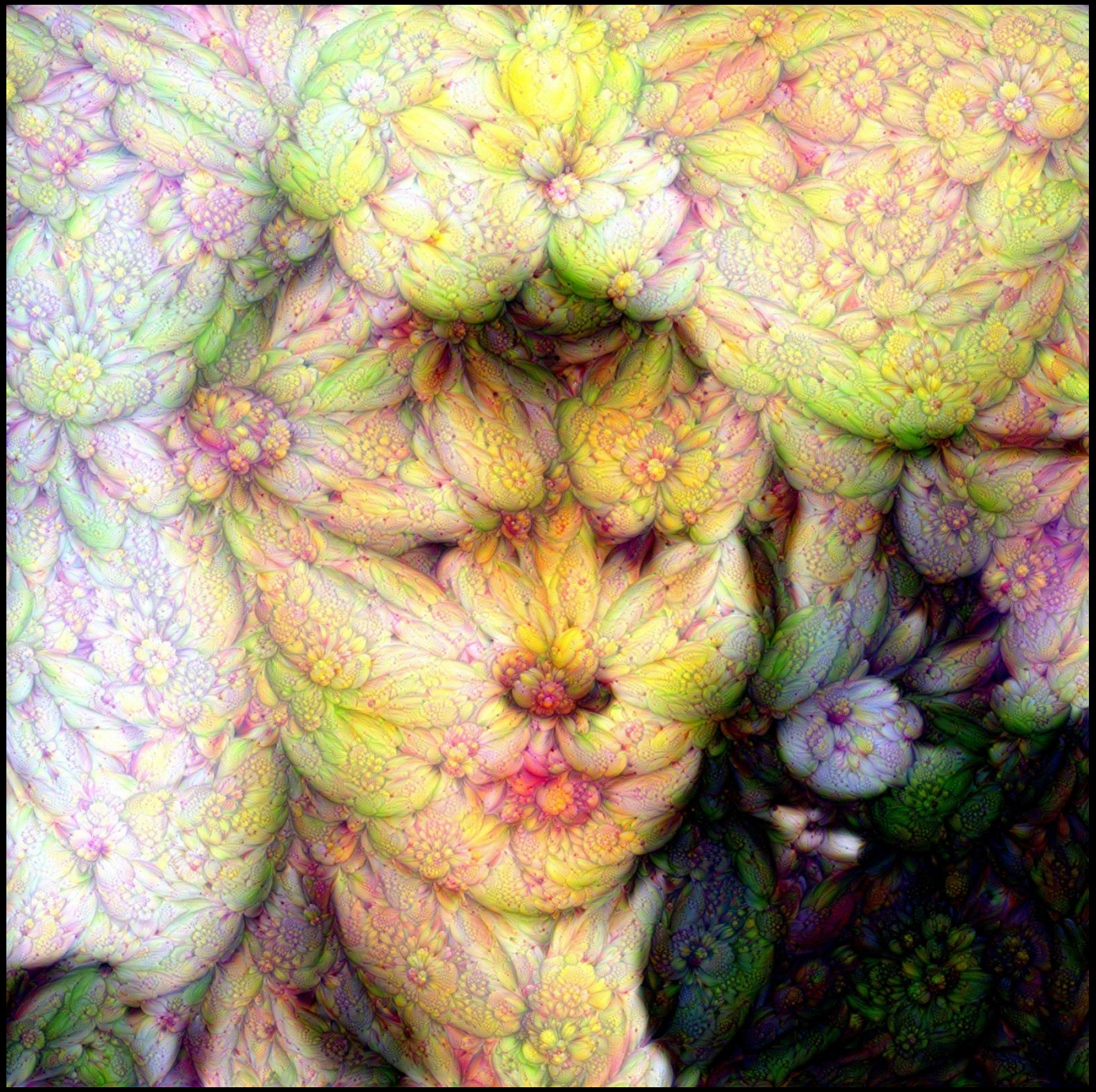
Deep Dream





Machine Learning and Art





Caption Generation

Caption Generation

Implementations:

- Google's tensorflow: Im2txt
  https://github.com/tensorflow/models/tree/master/im2txt

  "**Show and Tell: Lessons learned from the 2015 MSCOCO Image Captioning Challenge**." Oriol Vinyals, Alexander Toshev, Samy Bengio, Dumitru Erhan. *IEEE transactions on pattern analysis and machine intelligence (2016).* http://arxiv.org/abs/1609.06647

- Karpathy's Neuraltalk2 Torch:
  https://github.com/karpathy/neuraltalk2

Examples

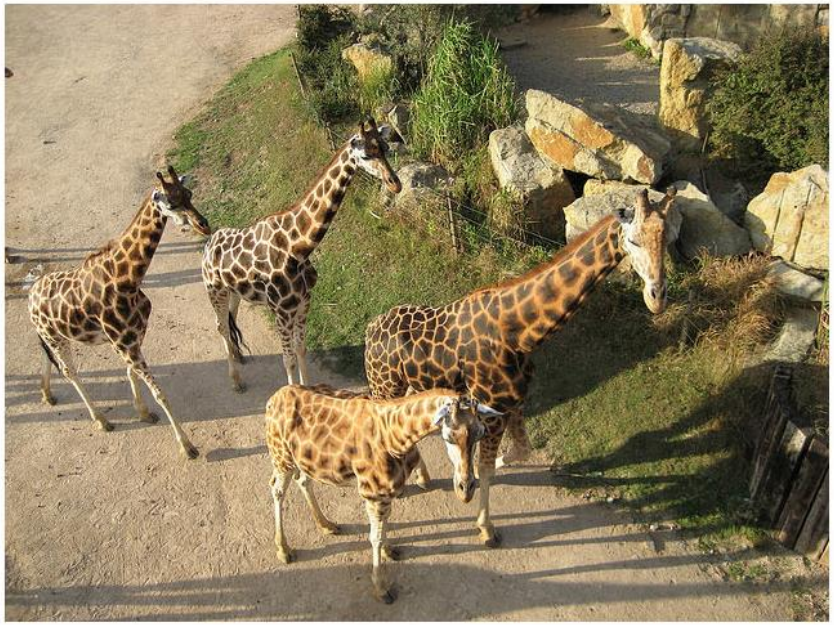
A person on a beach flying a kite.

A black and white photo of a train on a train track.

Examples

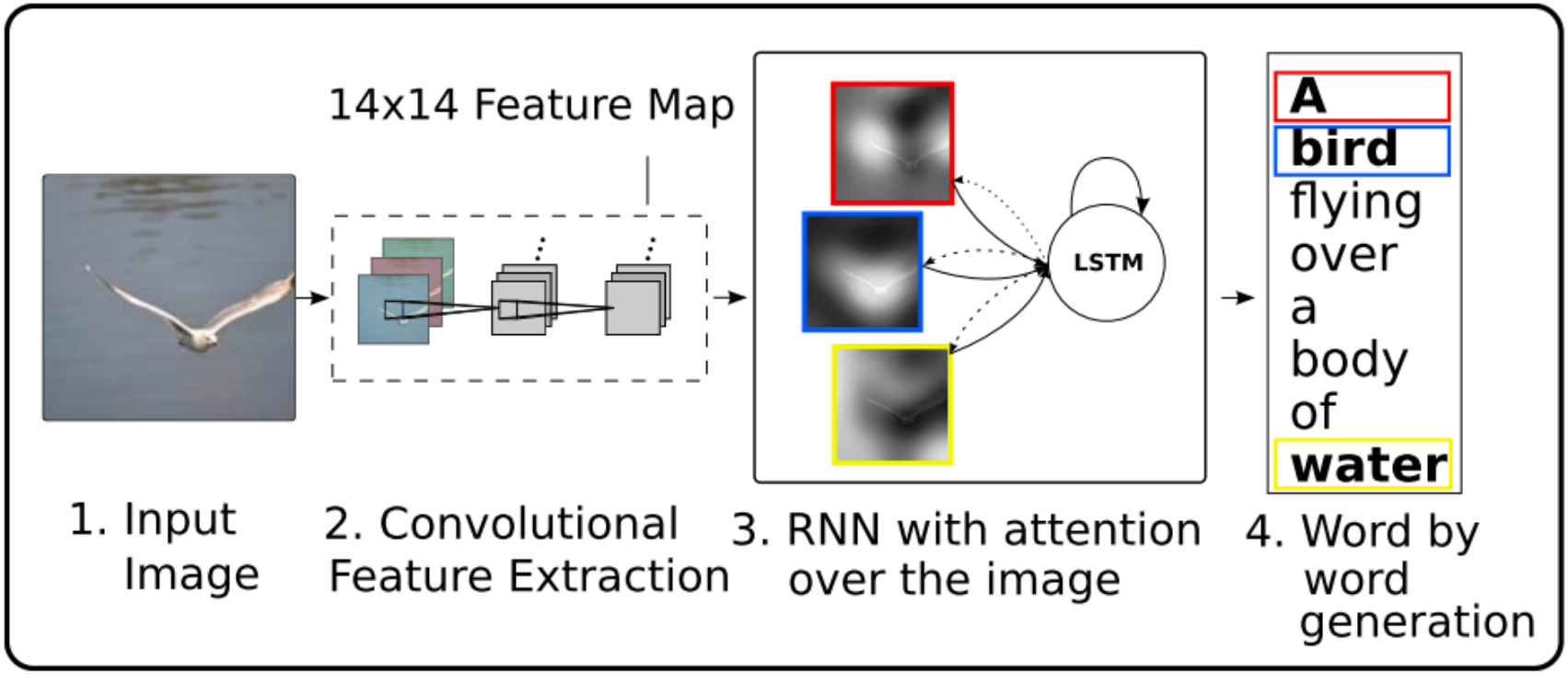
A person skiing down a snow covered slope.

A group of giraffe standing next to each other.



Computer Vision + Natural Language Processing

Xu et al, 2015



http://kelvinxu.github.io/projects/capgen.html

A woman is throwing a frisbee in a park.

A little girl sitting on a bed with a teddy bear.

A stop sign is on a road with a mountain in the background.

A woman is sitting at a table with a large pizza.

Word Embedding, Word2Vec

- http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/

- Mikolov, Tomas; Sutskever, Ilya; Chen, Kai; Corrado, Greg S.; Dean, Jeff (2013). *Distributed representations of words and phrases and their compositionality*. Advances in Neural Information Processing Systems. arXiv:1310.4546

- Efficient Estimation of Word Representations in Vector Space Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean https://arxiv.org/abs/1301.3781

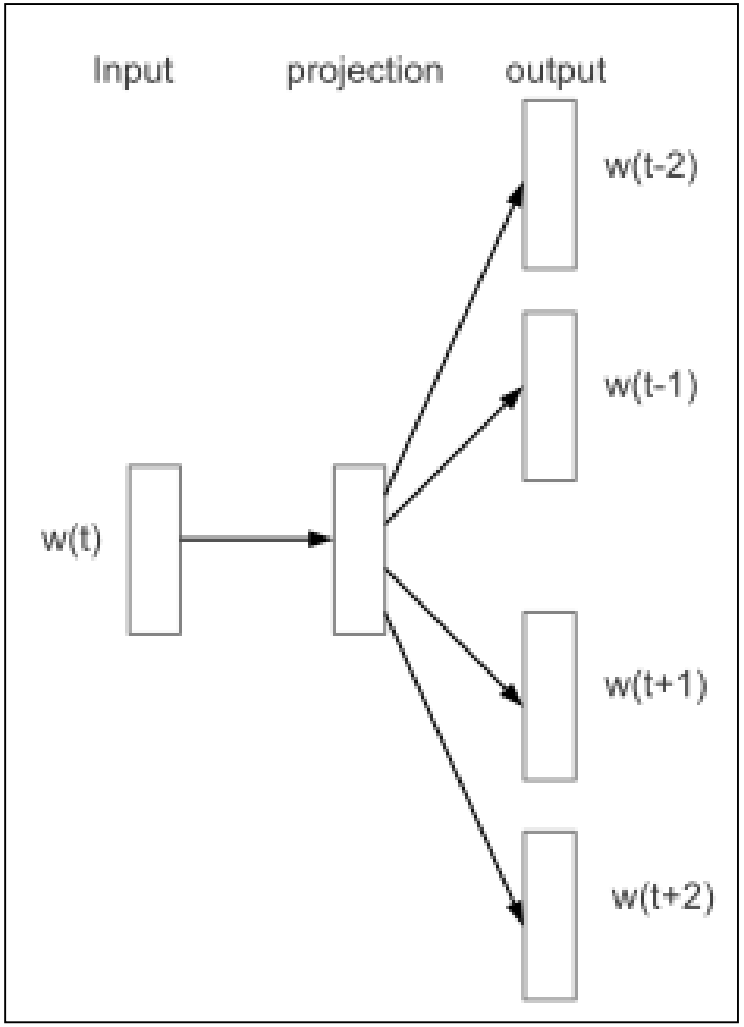
The Skip-Gram model

The goal is to maximize

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

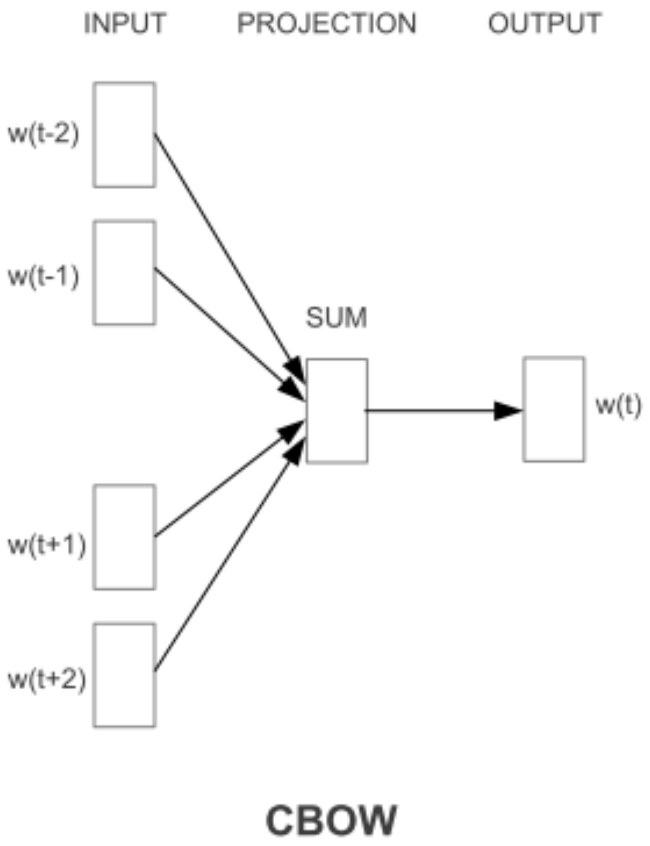
where

$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^{\top} v_{w_I}\right)}{\sum_{w=1}^{W}\exp\left({v'_w}^{\top} v_{w_I}\right)}$$

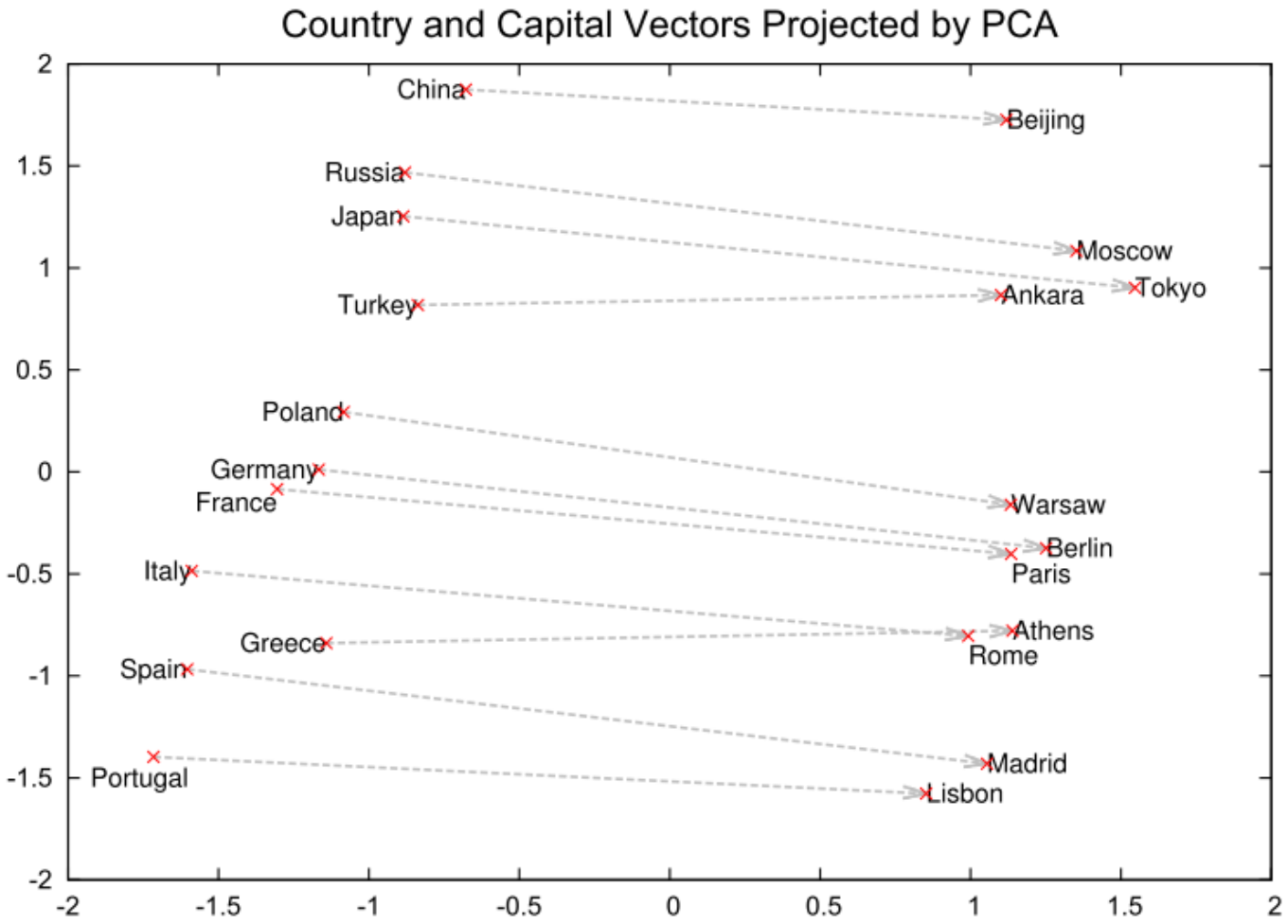


The Skip-gram model architecture. The training objective is to learn word vector representations that are good at predicting the nearby words

The Continuous Bag-of-Words (CBOW) Model



The CBOW architecture predicts the current word based on the context

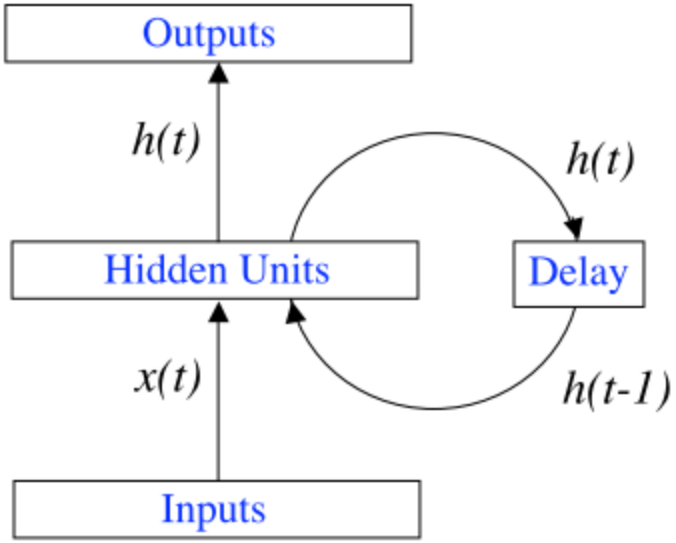


Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

Recurrent Neural Networks

$$h(t) = f_H\left(W_{IH}x(t) + W_{HH}h(t-1)\right)$$

$$y(t) = f_O\left(W_{HO}h(t)\right)$$

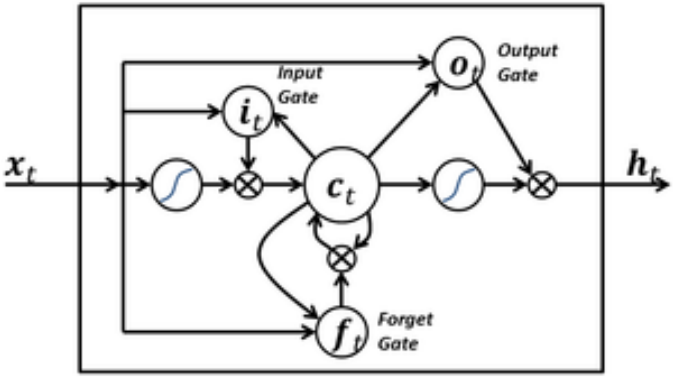


Long Short Term Memory (LSTM)

A LSTM block contains gates that determine when the input is significant enough to remember, when it should continue to remember or forget the value, and when it should output the value.

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

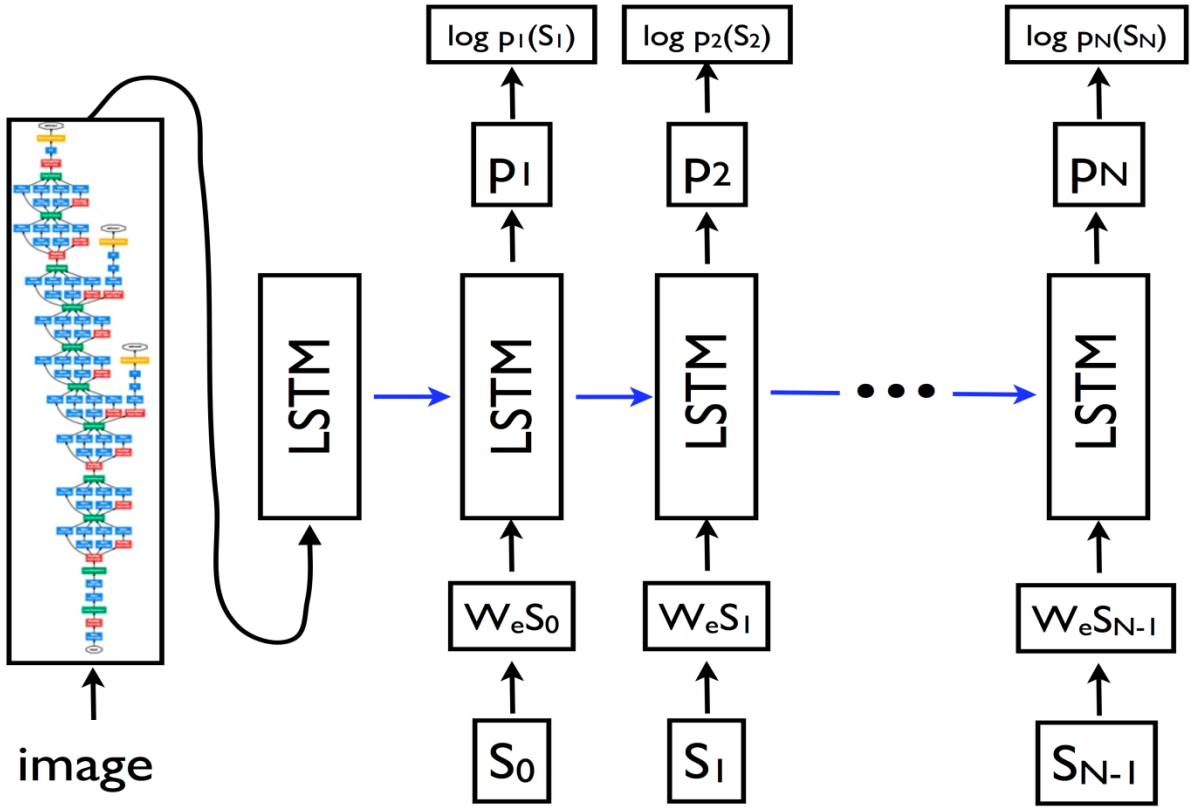
$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$
$$h_t = o_t \circ \sigma_h(c_t)$$



Variables

x_t: input vector
h_t: output vector
c_t: cell state vector
W, U and b: parameter matrices and vector
f_t, i_t and o_t: gate vectors
f_t: Forget gate vector. Weight of remembering old information.
i_t: Input gate vector. Weight of acquiring new information.
o_t: Output gate vector.

Caption Generation



$\{s_0, s_1, ..., s_{N-1}\}$ are the words of the caption and $\{w_e s_0, w_e s_1, ..., w_e s_{N-1}\}$ are their corresponding word embedding vectors.

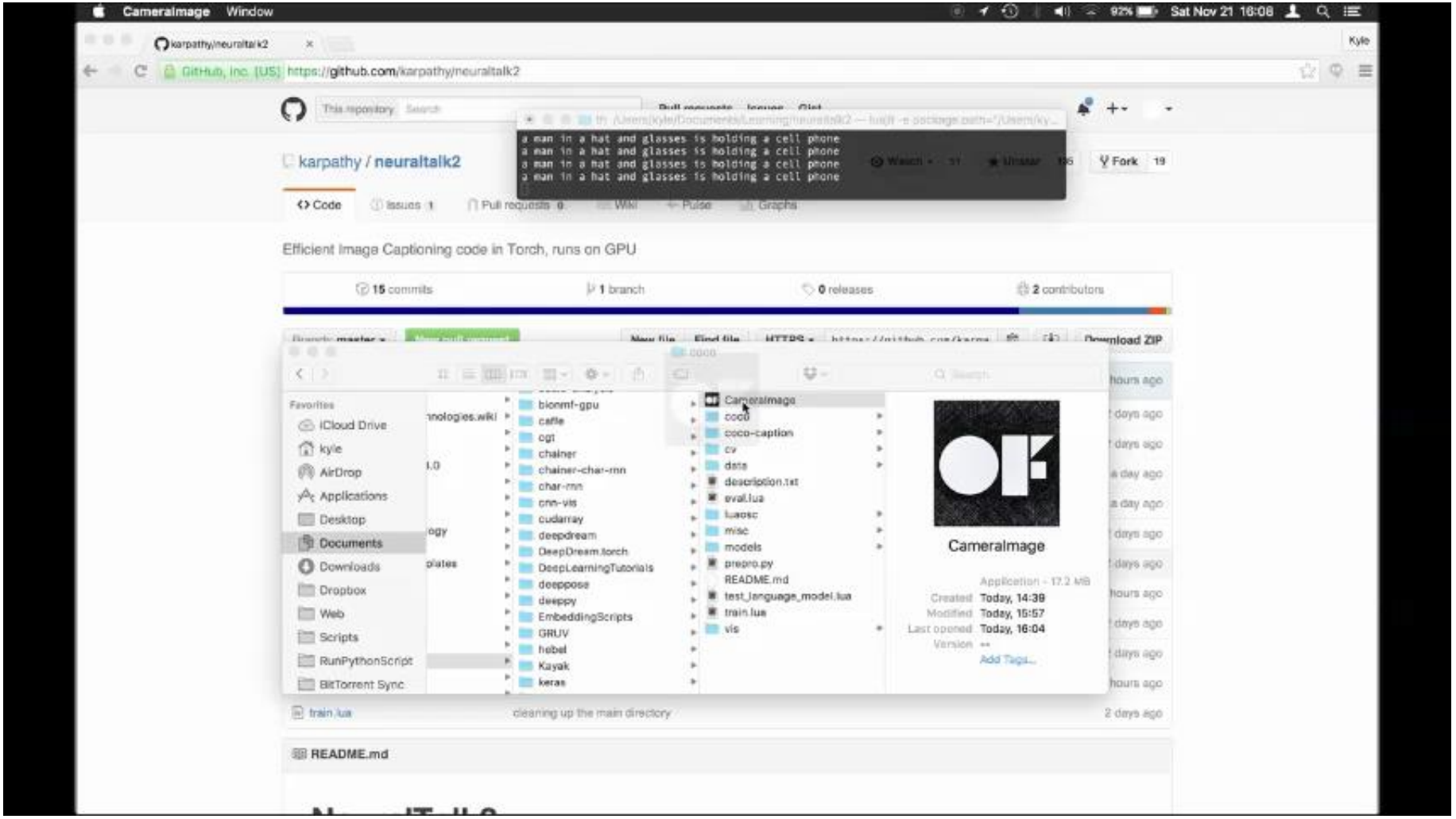
The outputs $\{p_1, p_2, ..., p_N\}$ of the LSTM are probability distributions generated by the model for the next word in the sentence. The terms $\{\log p_1(s_1), \log p_2(s_2), ..., \log p_N(s_N)\}$ are the log-likelihoods of the correct word at each step

Video Caption Generation

- Andrej Karpathy's "NeuralTalk2" code slightly modified to run from a webcam feed [github.com/karpathy/neuraltalk2]
- NeuralTalk is trained on the MS COCO dataset [mscoco.org/dataset/#captions-challenge2015]
- MS COCO contains 100k image-caption pairs
- All processing is done on a 2013 MacBook Pro with the NVIDIA 750M and only 2GB of GPU memory.
- Video recording: Walking around with the laptop open
- The openFrameworks code for streaming the webcam and reading from disk is available at [gist.github.com/kylemcdonald/b02edbc33942a85856c8]
- While the captions run at about four captions per second on the laptop, in this video one caption per second was generated to make it more reasonable.

Video Caption Generation

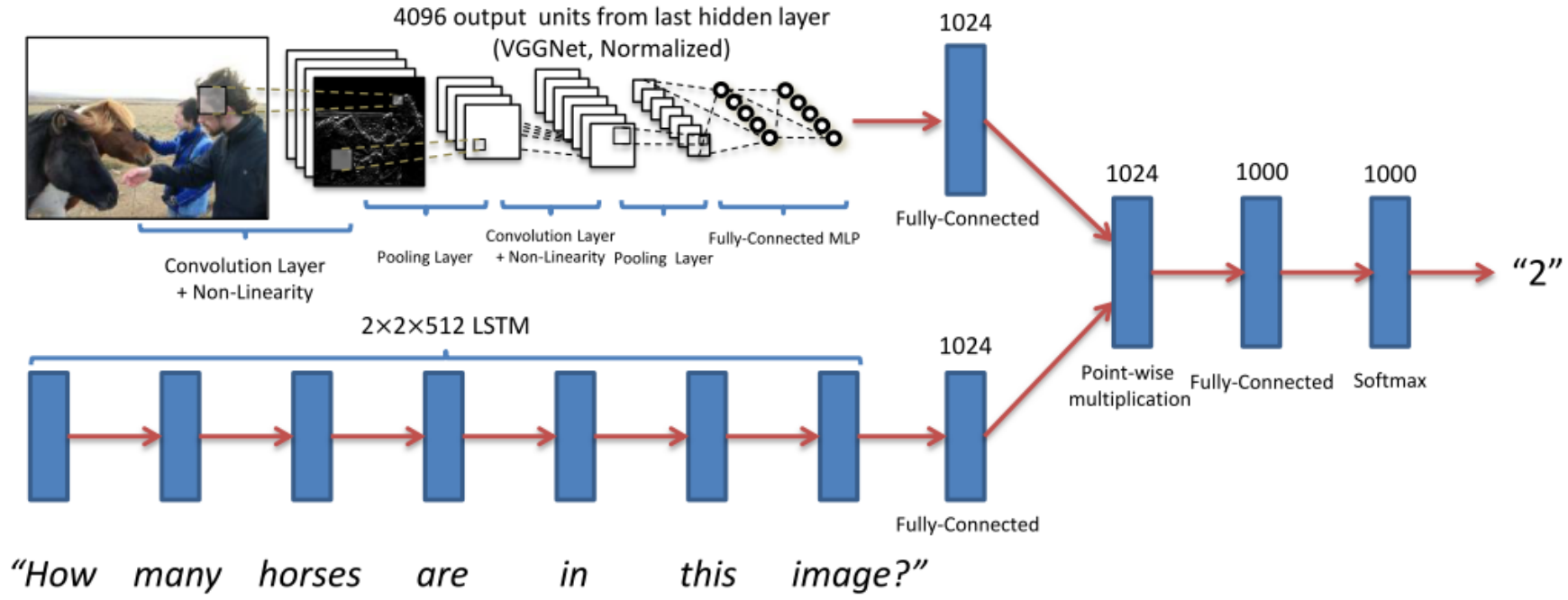
https://vimeo.com/146492001 from Kyle McDonald



Visual Question Answering

Visual Question Answering

https://arxiv.org/pdf/1505.00468.pdf



Demo: https://cloudcv.org/vqa/

What is he doing?

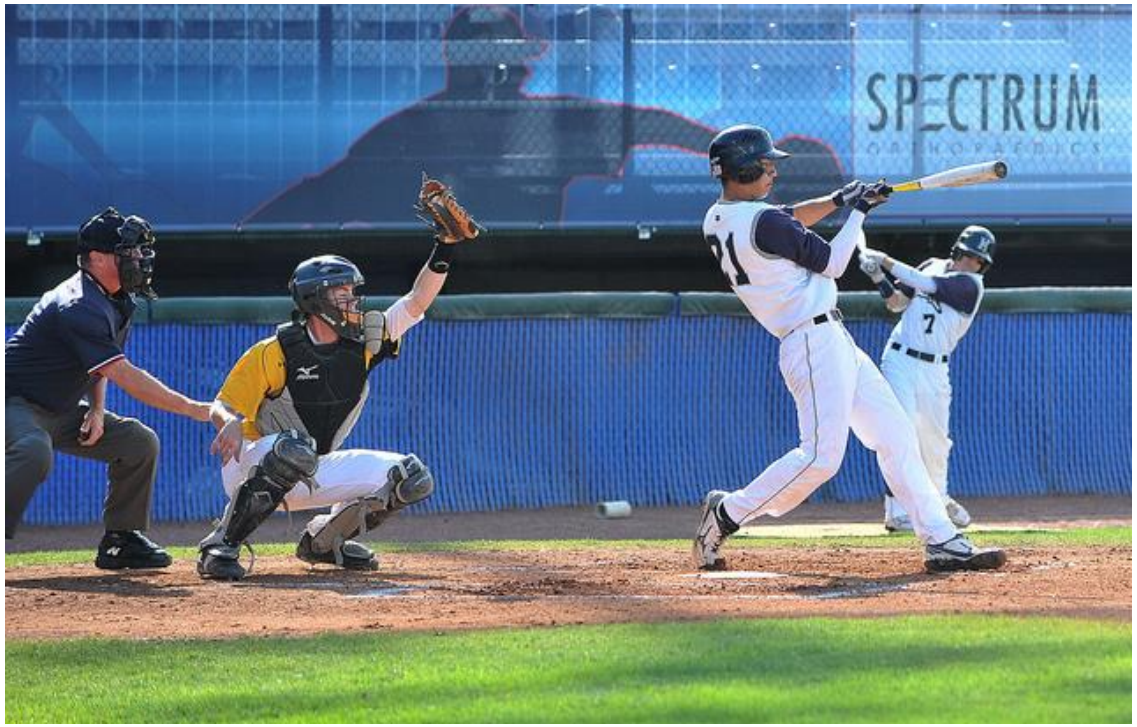
Answer	Confidence
playing tennis	0.9071
skateboarding	0.0084
tennis	0.0070
yellow	0.0020
blue	0.0017

Demo: https://cloudcv.org/vqa/

What is the color of his shirt?

white	0.2547
blue	0.1810
black	0.1604
yellow	0.1192
red	0.0699

Demo: https://cloudcv.org/vqa/



Where was this picture taken?

Answer	Confidence
baseball field	0.1447
tennis court	0.0676
baseball	0.0401
park	0.0188
skate park	0.0152

Demo: https://cloudcv.org/vqa/

Thanks for your Attention! ☺