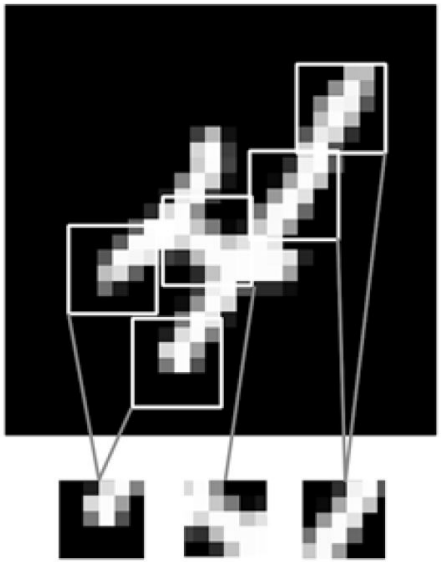
Module III

Deep Learning for Computer Vision

5.1.1 The Convolution Operation

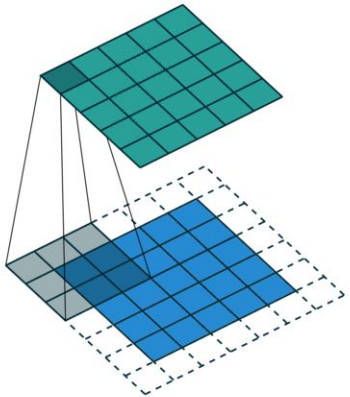
What is the difference between a densely connected layer and a convolution layer?

- Dense layers learn global patterns in their input feature space
- Convolution layers learn local patterns (patterns found in small 2D windows of 3x3)



Global pattern

Local patterns



If dense layers can learn “global” patterns (generalization) then why do we need a convolutional layer?

THINK PAIR SHARE

Property 1 of CNNs - Translation Invariance

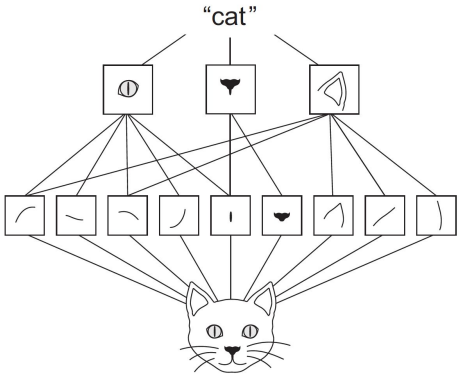
The patterns they learn are “translation invariant”

- The visual world is translation invariant
  - You don’t always have Chicago on your North
- Convolutional layer learns a pattern in lower-right corner of a picture!
  - Then, it can recognize it anywhere else in another/same picture
- This makes convnets data efficient
  - They can learn more patterns with fewer training samples

Property 2 of CNNs - Hierarchies of Patterns

Property 2 of CNNs - They can learn spatial “hierarchies of patterns”

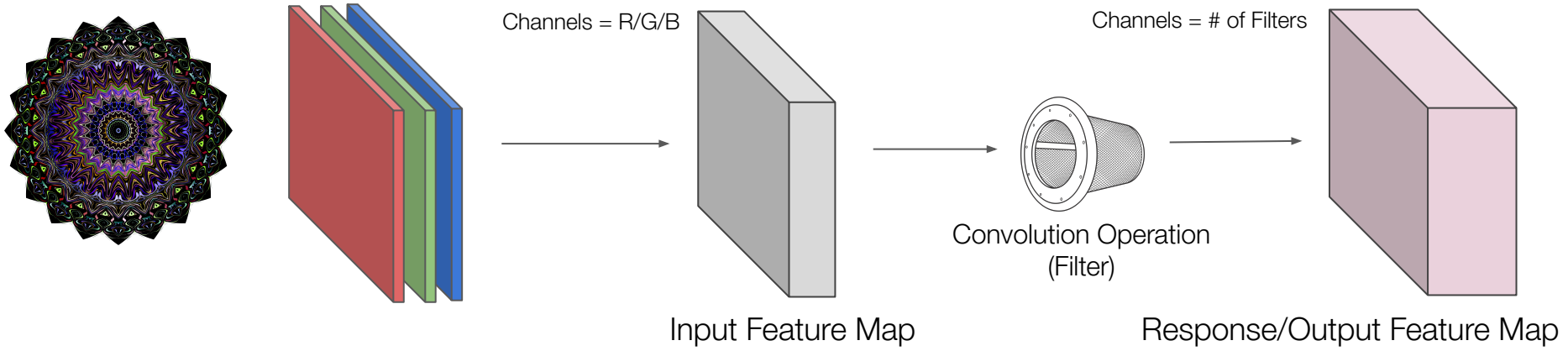
- The visual world is spatially hierarchical



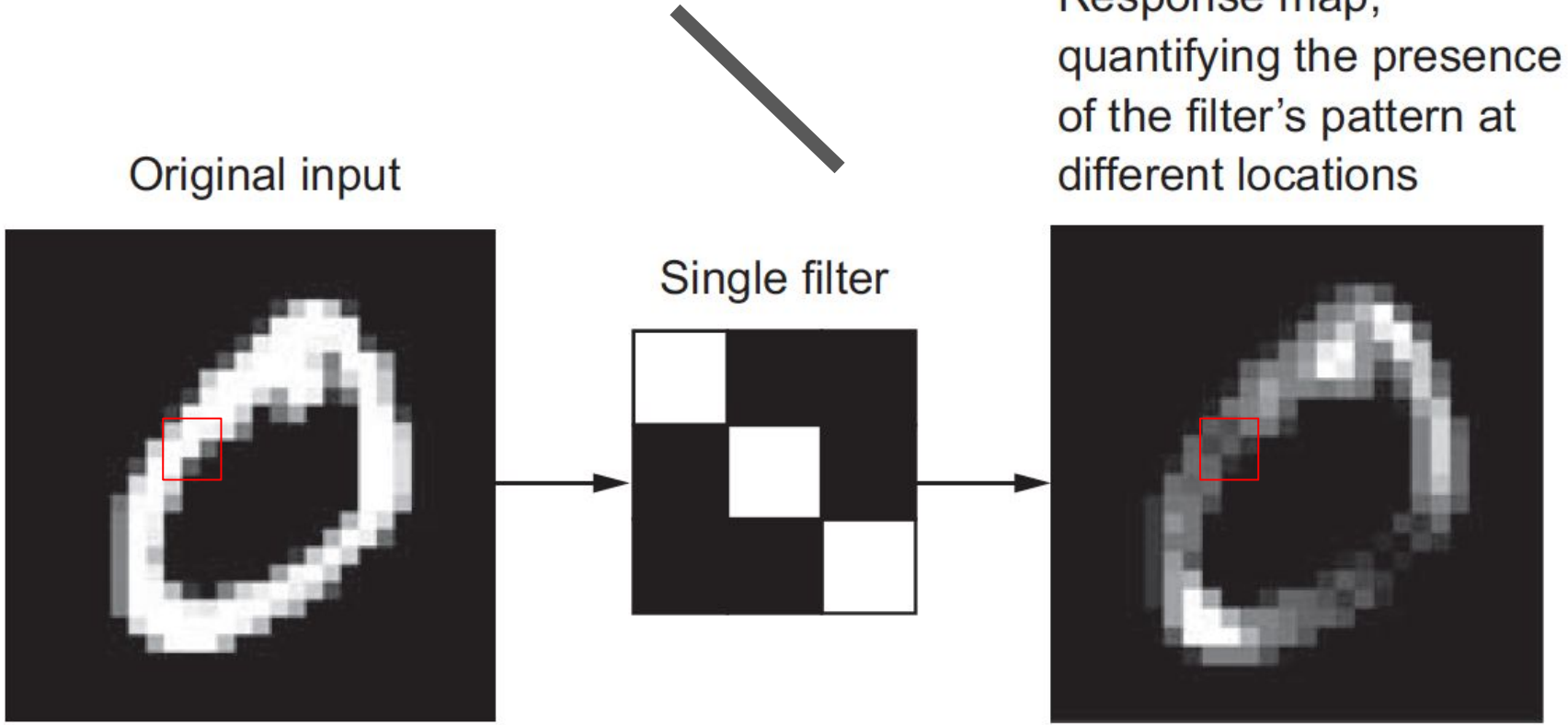
- A first convolution layer can learn small local patterns such as edges
  - Second convolution layer can learn larger patterns made of the features of the first layers, and so on
- This allows convnets to efficiently learn increasingly complex and abstract visual concepts

Basics of a Convolutional Layer (Conv2D)

- Convolutions operate over 3D tensors, called feature maps
  - Feature map - two spatial axes (height and width) & a depth axis (also called the channels axis)
  - For an RGB image, # of channels = 3
  - For a black-and-white picture, depth/channels = 1
- The convolution operation *extracts patches from its "input feature map"* and *applies the same transformation to all of these patches*, producing an "output feature map"
  - Depth of a feature map can be arbitrary (channels in the depth axis stand for filters)



Example of a Filter



Two Parameters of a Convolutional Layer

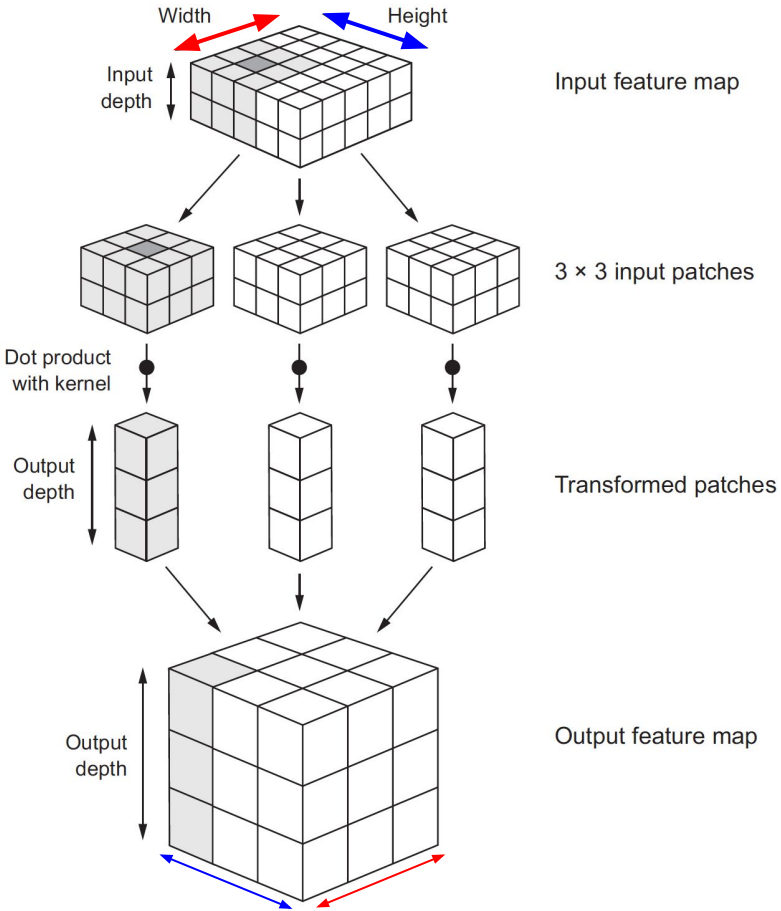
1. Size of the patches extracted from the inputs
   - Typically 3 × 3 or 5 × 5

2. Depth of the output feature map
   - The number of filters computed by the convolution

```
keras.layers.Conv2D(filters, kernel_size, ...)
```


Understood as `(kernel_size, kernel_size)`

The Convolution Process

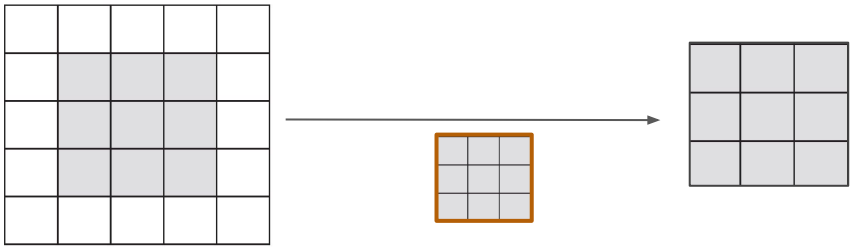


The height and width of the output feature map may not be same as the input. Why ? (two reasons)

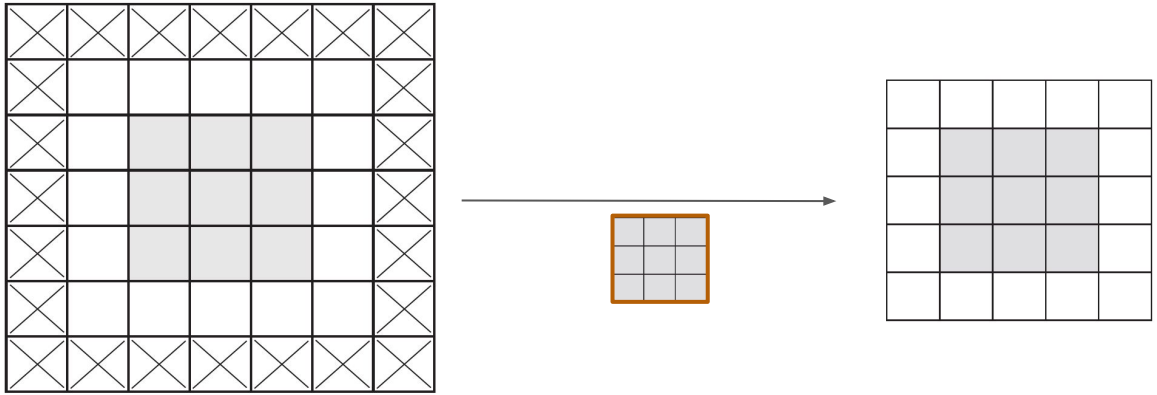
THINK PAIR SHARE

Border Effect & Padding

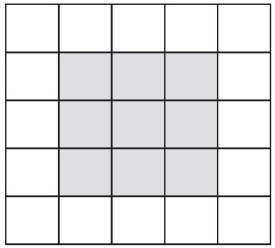
When a convolution of 3x3 is applied to an input of 5x5, the output is 3x3



To obtain a 5x5 output, we can pad the input with zeros (called **zero-padding**)



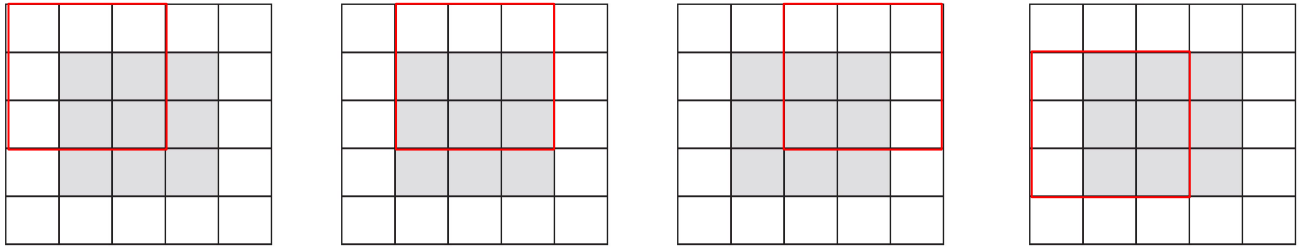
Convolution Strides (rarely used these days)



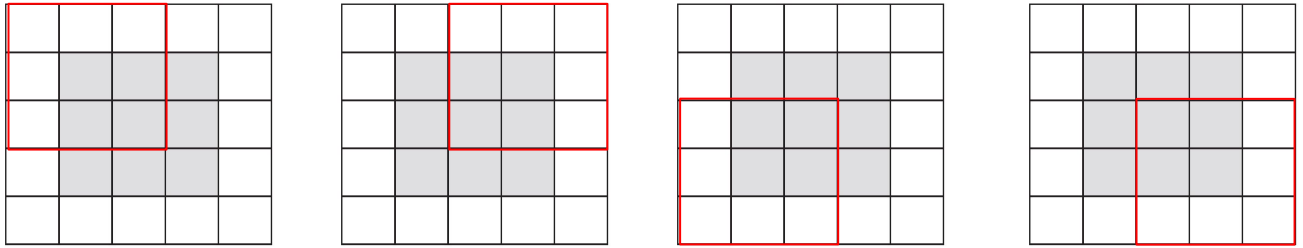
Input Image

Stride:
Distance between two successive windows of convolution

Stride = 1 (default)



Stride = 2

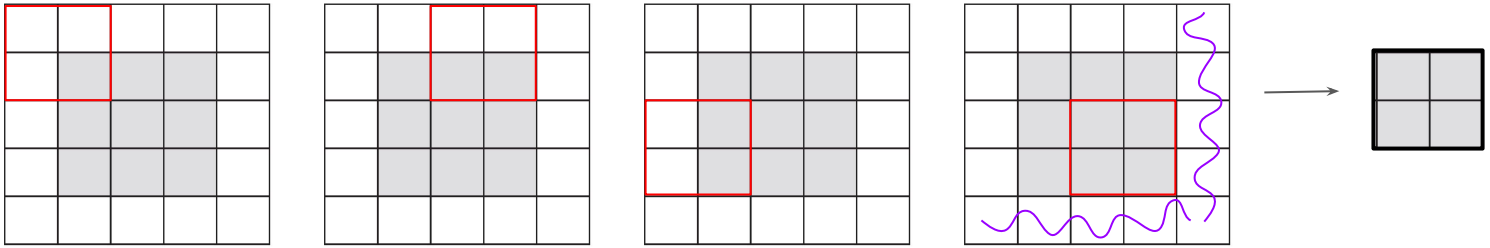


Q1. Why would someone want strides > 1?
Q2. How do they affect output size?

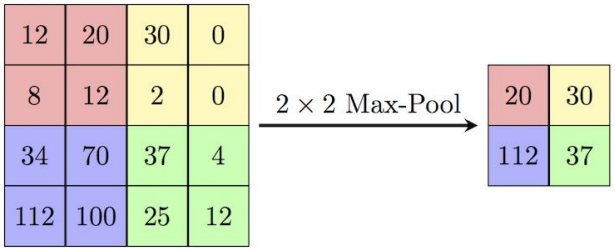
THINK PAIR SHARE

5.1.2 Max-Pooling

- Max pooling consists of extracting windows from the input feature maps and outputting the max value of each channel
- MaxPooling2D halves the size of the feature maps
  - For example, input of 26 × 26 halves to 13 × 13

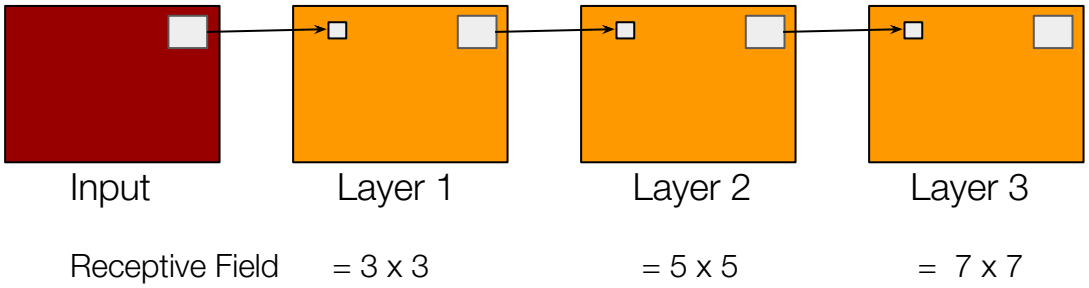


- It's conceptually similar to convolution
  - Instead of transforming local patches via a learned linear transformation (the convolution kernel), they're transformed via a hardcoded **max** tensor operation



Why Maxpool?

```
model_no_max_pool = models.Sequential()
model_no_max_pool.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model_no_max_pool.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_no_max_pool.add(layers.Conv2D(64, (3, 3), activation='relu'))
```



- The 3 x 3 windows in the third layer will contain information coming from 7 x 7 windows in the initial input
- We need the features from the last convolution layer to contain information about the totality of the input
  - How can we achieve that?

Why Maxpool?

```
model_no_max_pool = models.Sequential()
model_no_max_pool.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model_no_max_pool.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_no_max_pool.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

- We need to add at least one dense layer at the end
  - So that we have one neuron that can predict [0, 1]
- If we add 1 Dense layer to this network, how many parameters are we adding?
  - 64 x (22 x 22) + 1 = 30,977 parameters
- If we add 1 Dense layer with 100 neurons and second with 1 neuron?
  - ?
- Having too many parameters will cause intense overfitting issues!
  - Stacks of Max-pooling layers can decrease the size of feature maps

Classroom Activity

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

How many parameters does this network have?

Maxpool Along the Channels

Maxout Networks

Ian J. Goodfellow GOODFELI@IRO.UMONTREAL.CA
David Warde-Farley WARDEFAR@IRO.UMONTREAL.CA
Mehdi Mirza MIRZAMOM@IRO.UMONTREAL.CA
Aaron Courville AARON.COURVILLE@UMONTREAL.CA
Yoshua Bengio YOSHUA.BENGIO@UMONTREAL.CA
Département d'Informatique et de Recherche Opérationelle, Université de Montréal
2920, chemin de la Tour, Montréal, Québec, Canada, H3T 1J8

5.2.4 Data Preprocessing

- The Dogs vs. Cats dataset
  - Is not packaged with Keras
  - Original dataset at www.kaggle.com/c/dogs-vs-cats/data/
  - The pictures are medium-resolution color JPEGs
  - inputs of size 150 x 150
  - 812 MB
  - Contains 25,000 images of dogs and cats (12,500 from each class)

- We will create a smaller subset

```
dogs-vs-cats_small.zip
```

5.2.4 Data Preprocessing

- We should load the data into memory to do the training
  - Here is what we have been doing:

Pima Indians Diabetes Dataset

Split the data:

```
XTRAIN = dataset[:700,0:8]
YTRAIN = dataset[:700,8]
XVALIDATION = dataset[700:,0:8]
YVALIDATION = dataset[700:,8]
```

Preview the shapes:

```
print(XTRAIN.shape)
print(YTRAIN.shape)
print(XVALIDATION.shape)
print(YVALIDATION.shape)
```

Do the training:

```
model.fit(XTRAIN, YTRAIN, epochs=15, batch_size=10)
```

- Can we do the same for the cats/dogs dataset?
  - I.e. Load the entire dataset into XTRAIN and YTRAIN? Why?

An Ideal Way of Loading the Data & Training

- As the training starts:
    1. Read a picture file
    2. Decode the JPEG content to RGB grids of pixels
    3. Convert these into floating-point tensors
    4. Rescale the pixel values (between 0 and 255) to the [0, 1] interval
    5. Add it to the pool of training/validation dataset

- Keras has utilities to take care of these steps automatically
    - contains the class `ImageDataGenerator` which lets you quickly set up Python generators that can automatically turn image files on disk into batches of preprocessed tensors

ImageDataGenerator

Listing 5.7 Using `ImageDataGenerator` to read images from directories

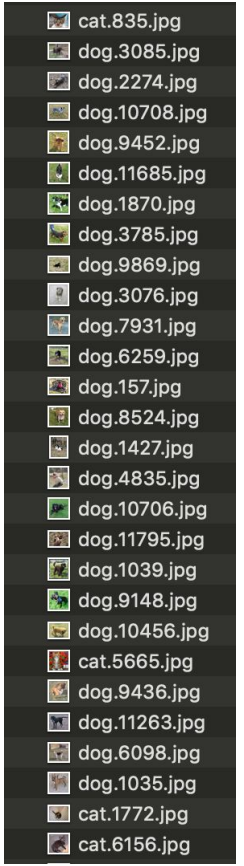
```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(150, 150)
        batch_size=20,
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

Rescales all images by 1/255

Target directory

Resizes all images to 150 × 150

Because you use binary_crossentropy loss, you need binary labels.



ImageDataGenerator

Understanding Python generators

A *Python generator* is an object that acts as an iterator: it's an object you can use with the `for ... in` operator. Generators are built using the `yield` operator.

Here is an example of a generator that yields integers:

```
def generator():
    i = 0
    while True:
        i += 1
        yield i

for item in generator():
    print(item)
    if item > 4:
        break
```

It prints this:

```
1
2
3
4
5
```

DEMO

Branch: master ▾ 2019-Spring-DL / course_content / module3_convnets / DataGenerator.ipynb Find file Copy path

badriadhikari Add files via upload d5a8466 6 minutes ago

1 contributor

794 lines (793 sloc) | 132 KB Raw Blame History

Open in Colab

Here is the Original example.

In [1]:
```
import keras
import os, shutil
from keras import optimizers
from keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
from keras import layers
from keras import models

keras.__version__
```

```
/home/notebook/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning: Conversio
n of the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it
will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

Out[1]: '2.2.2'

In [6]:
```
base_dir = '/Users/badriadhikari/Downloads/dogs-vs-cats_small/'
base_dir = '/home/notebook/AnacondaNotebooks/dogs-vs-cats_small/'

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')
```

5.2.5 Using Data Augmentation

- The approach of generating more training data from existing training samples
  - by augmenting the samples via a number of random transformations that yield believable-looking images
- The goal is: "at training time, the model will never see the exact same picture twice"
  - This helps expose the model to more aspects of the data and generalize better
- If we read images using ImageDataGenerator instance (in Keras)
  - a number of random transformations can be performed

Listing 5.7 Using `ImageDataGenerator` to read images from directories

```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(150, 150)
        batch_size=20,
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

Rescales all images by 1/255

Target directory

Resizes all images to 150 × 150

Because you use binary_crossentropy loss, you need binary labels.

Previous Code

Listing 5.11 Setting up a data augmentation configuration via `ImageDataGenerator`

```
datagen = ImageDataGenerator(
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
```

Augmentation using `ImageDataGenerator`

- `rotation_range`

  a value in degrees (0–180), a range within which to randomly rotate pictures

- `width_shift` and `height_shift`

  ranges (as a fraction of total width or height) within which to randomly translate pictures vertically or horizontally

- `shear_range`

  randomly applying shearing transformations.

- `zoom_range`

  randomly zooming inside pictures.

- `horizontal_flip`

  is for randomly flipping half the images horizontally—relevant when there are no assumptions of horizontal asymmetry (for example, real-world pictures)

- `fill_mode`

  the strategy used for filling in newly created pixels, which can appear after a rotation or a width/height shift

… (there are more)

```
datagen = ImageDataGenerator(
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
```

DEMO

Branch: master ▾ 2019-Spring-DL / course_content / module3_convnets / DL_DataAugmentation.ipynb Find file Copy path

badriadhikari Created using Colaboratory e09889d a minute ago

1 contributor

291 lines (291 sloc) 471 KB Raw Blame History

Open in Colab

In [6]:
```
from google.colab import files
uploaded = files.upload()
```

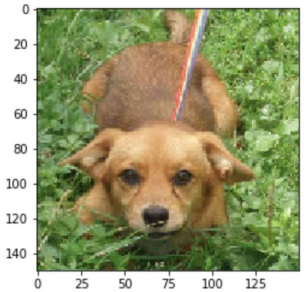
```
Saving dog-2109.jpg to dog-2109.jpg
```

In [0]:
```
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
```

In [0]:
```
img = image.load_img('dog-2109.jpg', target_size=(150, 150))
```

In [20]:
```
plt.imshow(img)
```

Out[20]:
```
<matplotlib.image.AxesImage at 0x7ff033246be0>
```



Training with Augmentation

- Data-augmentation ‘implies’ the network will never see the same input twice
  - But the inputs it sees are still heavily intercorrelated, because they come from a small number of original images
  - You can’t produce new information, you can only remix existing information
- This could ‘increase’ the chances of overfitting
- Add a Dropout layer to your model, right before the densely connected classifier

Homework:
Train CNN on MNIST
with data augmentation!

Listing 5.13 Defining a new convnet that includes dropout

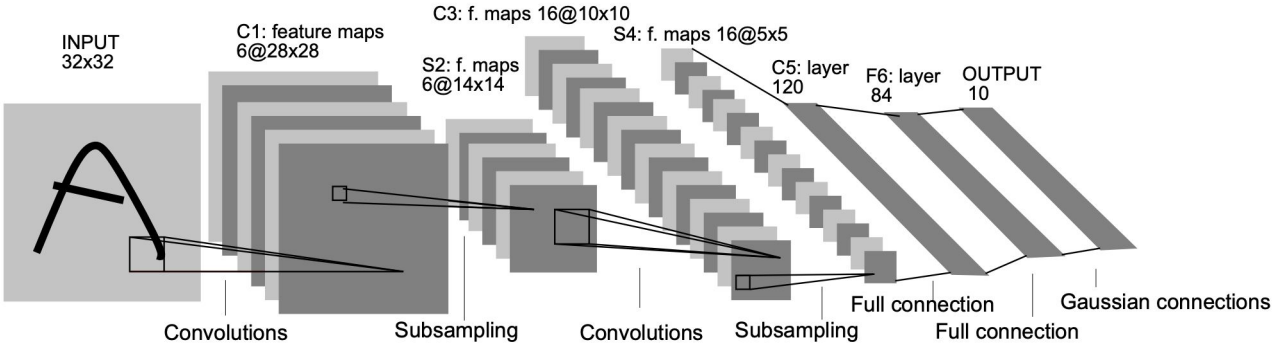
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

Some Classic Networks

Reference: https://www.coursera.org/learn/convolutional-neural-networks

LeNet-5

- Trained on the MNIST dataset (Grayscale images - 60K + 10K) to recognize handwritten digits



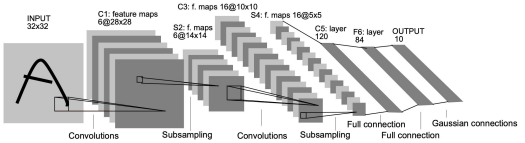
PROC. OF THE IEEE, NOVEMBER 1998

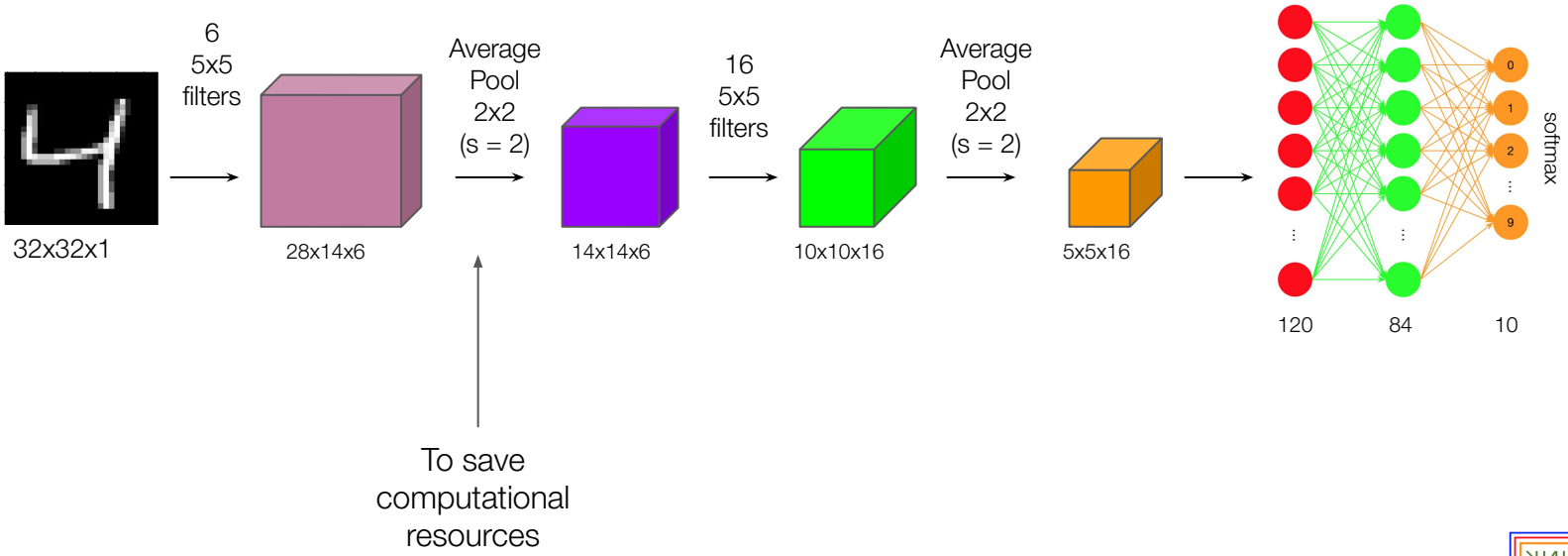
Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

LeNet-5





Approx. how many parameters?

DEMO

Branch: master ▾ 2019-Spring-DL / course_content / module3_convnets / LeNet_5.ipynb Find file Copy path

badriadhikari Created using Colaboratory a8e7c8f just now

1 contributor

313 lines (313 sloc) 15 KB Raw Blame History

Open in Colab

In [0]:
```
from keras.datasets import mnist
from keras.utils import to_categorical
from keras.layers import *
from keras.models import *
import matplotlib.pyplot as plt
```

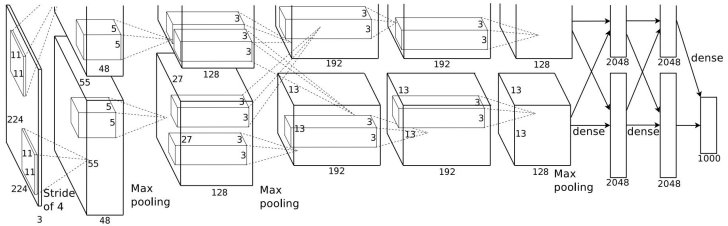
In [10]:
```
( train_images, train_labels ), ( validation_images, validation_labels ) = mnist.load_data()

print("Training dataset:")
print(train_images.shape)
print(train_labels.shape)
print('Validation dataset:')
print(validation_images.shape)
print(validation_labels.shape)
```

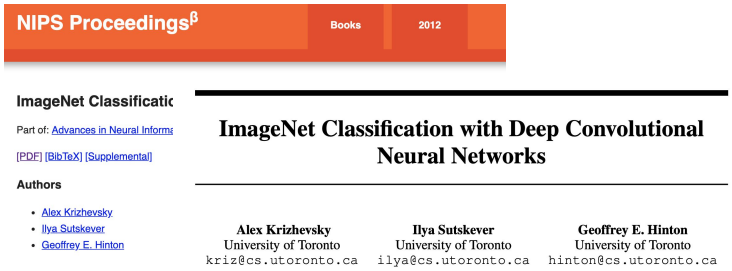
```
Training dataset:
(60000, 28, 28)
(60000,)
Validation dataset:
(10000, 28, 28)
(10000,)
```

AlexNet

- ImageNet (ImageNet Large Scale Visual Recognition Competition) 2012 Challenge
  - Classify the 1.2 million high-resolution images in the contest into the 1000 different classes
- ILSVRC uses a subset of ImageNet
  - Roughly 1000 images in each of 1000 categories
  - Roughly 1.2 million training images, 50,000 validation images, and 150,000 testing images
  - There are roughly 1.2 million training images, 50,000 validation images, and 150,000 testing images
- AlexNet achieved a winning top-5 test error rate of 15.3% in the competition
  - compared to 26.2% achieved by the second-best entry
- The network has ? million parameters and 650,000 neurons
  - Consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax

Training on multiple GPUs




NIPS Proceedings β Books 2012

ImageNet Classificatic

Part of: Advances in Neural Informa

[PDF] [BibTeX] [Supplemental]

Authors

- Alex Krizhevsky
- Ilya Sutskever
- Geoffrey E. Hinton

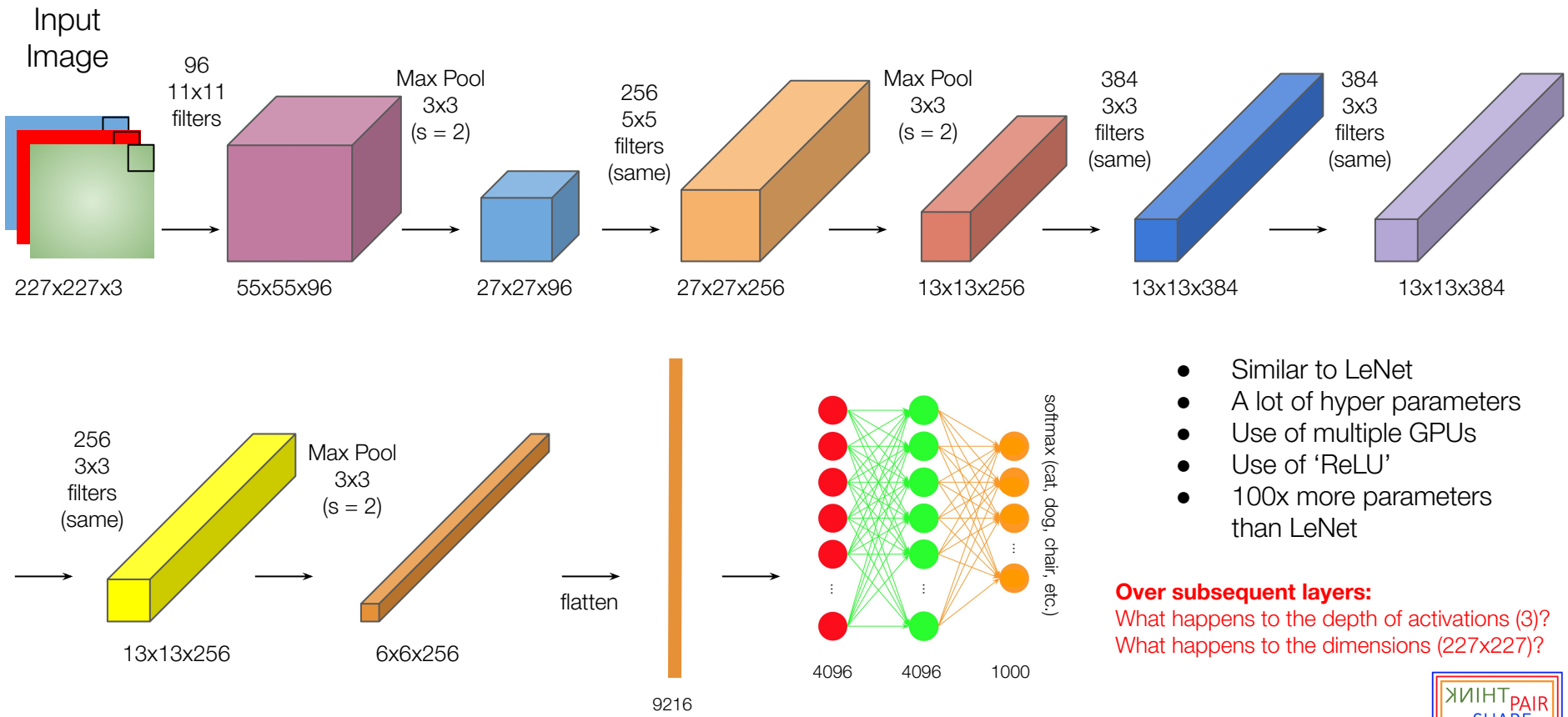
ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky University of Toronto kriz@cs.utoronto.ca

Ilya Sutskever University of Toronto ilya@cs.utoronto.ca

Geoffrey E. Hinton University of Toronto hinton@cs.utoronto.ca

AlexNet

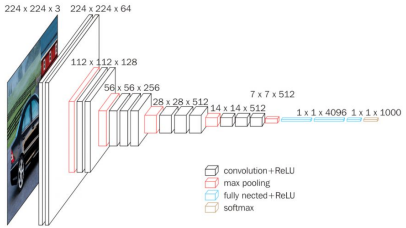


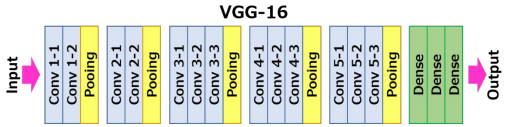
- Similar to LeNet
- A lot of hyper parameters
- Use of multiple GPUs
- Use of 'ReLU'
- 100x more parameters than LeNet

Over subsequent layers:
What happens to the depth of activations (3)?
What happens to the dimensions (227x227)?

VGG-16

- ILSVRC (ImageNet Large Scale Visual Recognition Competition) 2014
  - GoogLeNet was winner and VGG-16 was 1st runner-up
- CNN model proposed by K. Simonyan and A. Zisserman from the University of Oxford (Visual Geometry Group)
  - Achieved 92.7% top-5 test accuracy in ImageNet
- Improvements over AlexNet
  - replace large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another
- VGG16 was trained for weeks and was using NVIDIA Titan Black GPUs





Published as a conference paper at ICLR 2015

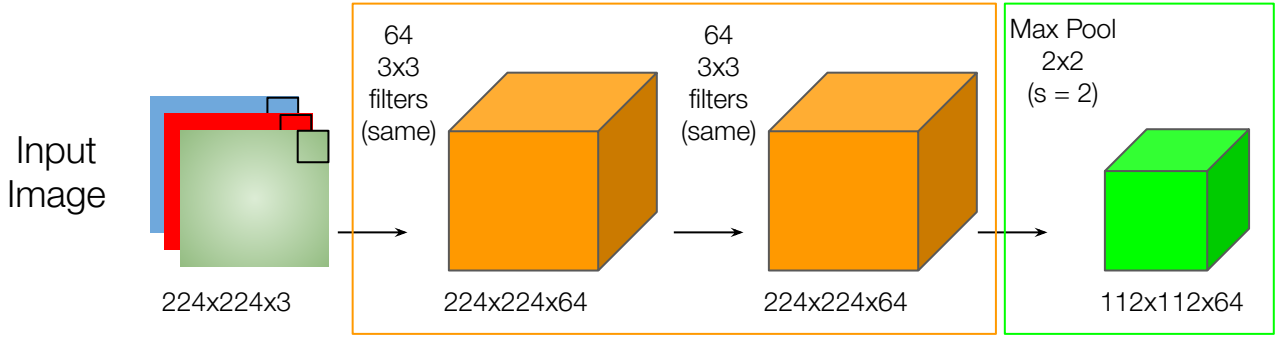
VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

Karen Simonyan* **& Andrew Zisserman+**
Visual Geometry Group, Department of Engineering Science, University of Oxford
{karen,az}@robots.ox.ac.uk

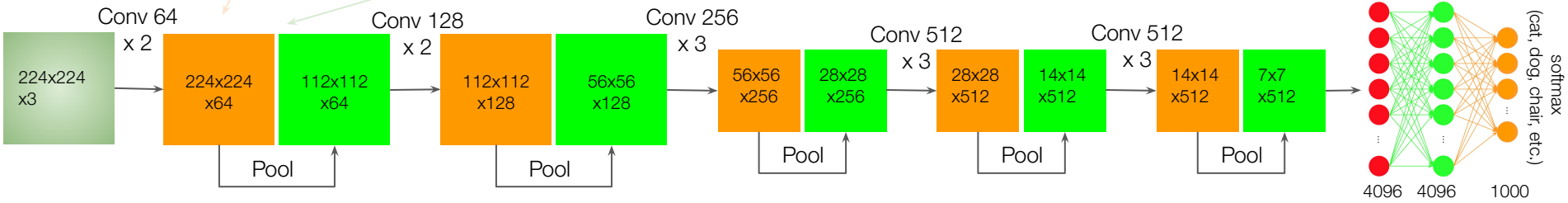
VGG-16 Motivation

- Use of “3x3 convolutions” and “2x2 max-pooling with stride 2” everywhere
  - Occam's razor - “simpler solutions are more likely to be correct than complex ones”
- Always use “same” padding
- Simplified network architecture
- ‘16’ stands for number of layers
  - 16 layers that have weights

VGG-16



- Very large n/w
- Around 138 million parameters
- Relative uniformity of the architecture makes it attractive to many users
- Pattern (as you go deeper): Dimension reduction by ½ and channels increase by 2
- VGG-19 is newer version but is only negligibly better



DEMO

Branch: master ▾ 2019-Spring-DL / course_content / module3_convnets / Using_VGG16.ipynb Find file Copy path

badriadhikari Created using Colaboratory 4dd9e05 just now

1 contributor

448 lines (448 sloc) | 189 KB Raw Blame History

Open in Colab

```
In [1]: from keras.applications.vgg16 import VGG16
        model = VGG16()
```

```
Using TensorFlow backend.
```

Training a Very Deep Convnets

```
model = Sequential()
model.add(Conv2D(filters = 16, kernel_size = 5, activation = 'relu', input_shape = (28,28,1)))

for i in range(1000):
    model.add(Conv2D(filters = 4, kernel_size = 5, activation = 'relu', padding='same'))

model.add(Conv2D(filters = 16, kernel_size = 5, activation = 'relu'))
model.add(MaxPooling2D(pool_size = 2, strides = 2))
model.add(Flatten())
model.add(Dense(units = 16, activation = 'relu'))
model.add(Dense(units = 16, activation = 'relu'))
model.add(Dense(units = 10, activation = 'softmax'))
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])

print(model.summary())
```

Some major villains when training ‘very deep’ networks:

- Vanishing Gradient, Exploding Gradient, and Internal Covariate Shift

‘ReLU’ and ‘BatchNormalization’ greatly resolve this

- But, BatchNormalization slows down training

DEMO

Branch: master ▾ 2019-Spring-DL / course_content / module3_convnets / CNN_Depth_and_Learning.ipynb Find file Copy path

badriadhikari Created using Colaboratory 68f44a1 5 minutes ago

1 contributor

1130 lines (1130 sloc) 64.8 KB Raw Blame History

Open in Colab

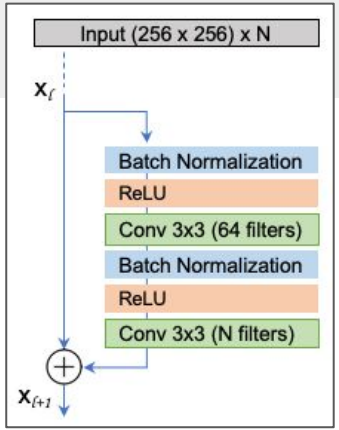
Effect of network depth on learning/prediction accuracy

In [0]:
```
from keras.datasets import mnist
from keras.utils import to_categorical
from keras.layers import *
from keras.models import *
import matplotlib.pyplot as plt
```

Residual Networks

Reference: https://www.coursera.org/learn/convolutional-neural-networks

A Residual Block



Input Image

64 3x3 Filters (same)

ReLU

3 3x3 Filters (same)

ReLU

28x28x3

"main path"

28x28x64

28x28x64

28x28x3

28x28x3

One convolutional layer

One convolutional layer

"shortcut"

Can we add after the first ReLU?

Keras functional API

```
from keras.models import Sequential

model = Sequential()
model.add(Dense(64, input_dim=784, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))
print(model.summary())

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

model.fit(data, labels)
```

Standard Keras Code

Functional API Code

```
from keras.layers import Input, Dense
from keras.models import Model

# This returns a tensor
inputs = Input(shape=(784,))

# a layer instance is callable on a tensor, and returns a tensor
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# This creates a model that includes
# the Input layer and three Dense layers
model = Model(inputs=inputs, outputs=predictions)

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels)  # starts training
```

The Keras functional API is the way to go for defining complex models, such as multi-output models, directed acyclic graphs, or models with shared layers.

A Residual Block in Keras

```
# FCN (fully conv.)
a = Conv2D(...)(input)
b = Conv2D(...)(a)
c = Conv2D(...)(b)
d = Conv2D(...)(c)
e = Conv2D(...)(d)
...
```

```
# Residual
a = Conv2D(...)(input)
b = add(a, input)
c = Conv2D(...)(b)
d = add(c, b)
e = Conv2D(...)(d)
f = add(e, d)
...
```

```
# for loop for depth = 32
```

?

DEMO

Branch: master ▾ **2019-Spring-DL** / course_content / module3_convnets /
CNN_Depth_and_Residual_Connections.ipynb

Find file | Copy path

badriadhikari Created using Colaboratory — a67dcb0 — 2 days ago

1 contributor

1565 lines (1565 sloc) | 139 KB — Raw | Blame | History

In [0]:
```
from keras.datasets import mnist
from keras.utils import to_categorical
from keras.layers import *
from keras.models import *
import matplotlib.pyplot as plt
from keras.utils import plot_model
```

Why Do Residual Networks Work?

$$X_{l+1} = \underbrace{W_b * ReLU(W_a * ReLU(X_l))}_{\text{Learning of the Residual Block}} + X_l$$

If the residual block learns something reasonable:

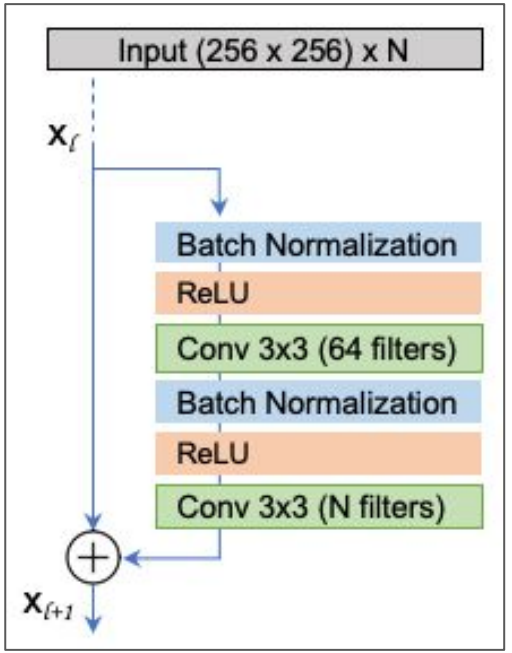
$$X_{l+1} = \text{“something reasonable”} + X_l$$

If the residual block weights are closer to zero (say, during L2 reg.):

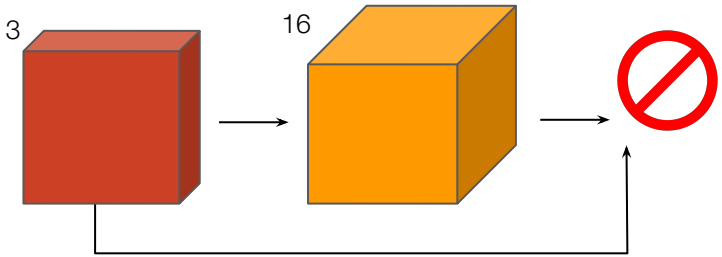
$$X_{l+1} = \text{“almost zero”} + X_l$$
$$= X_l$$

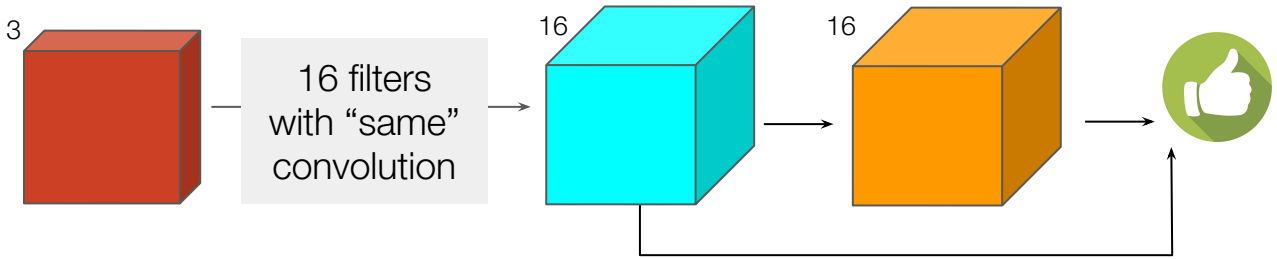
If the residual block weights are closer to one:

$$X_{l+1} = \text{“something reasonable”} + X_l$$
$$= 2 \times X_l$$



Residual Block - Can Add Tensors Same Shape Only





ResNet

- ResNet won the 1st place on the ILSVRC 2015 classification task
- “Ensemble” of these residual nets achieved 3.57% error on the ImageNet test set
- Depth of up to 152 layers
  - 8 times deeper than VGG nets
  - But still lower complexity

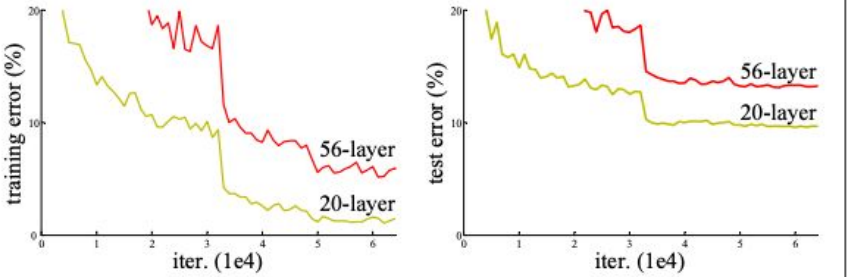


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

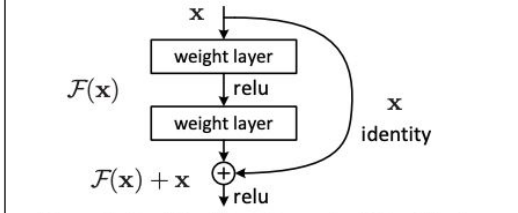
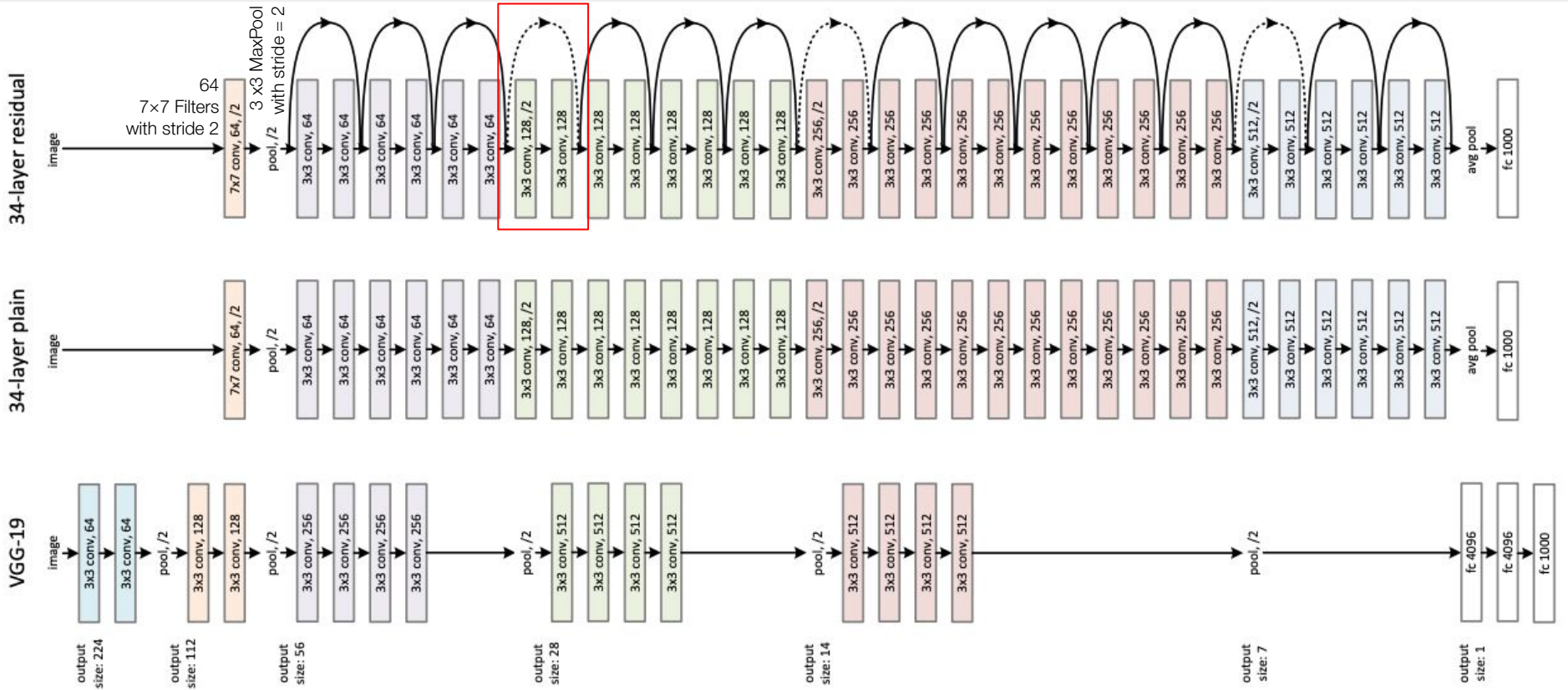


Figure 2. Residual learning: a building block.

Deep Residual Learning for Image Recognition

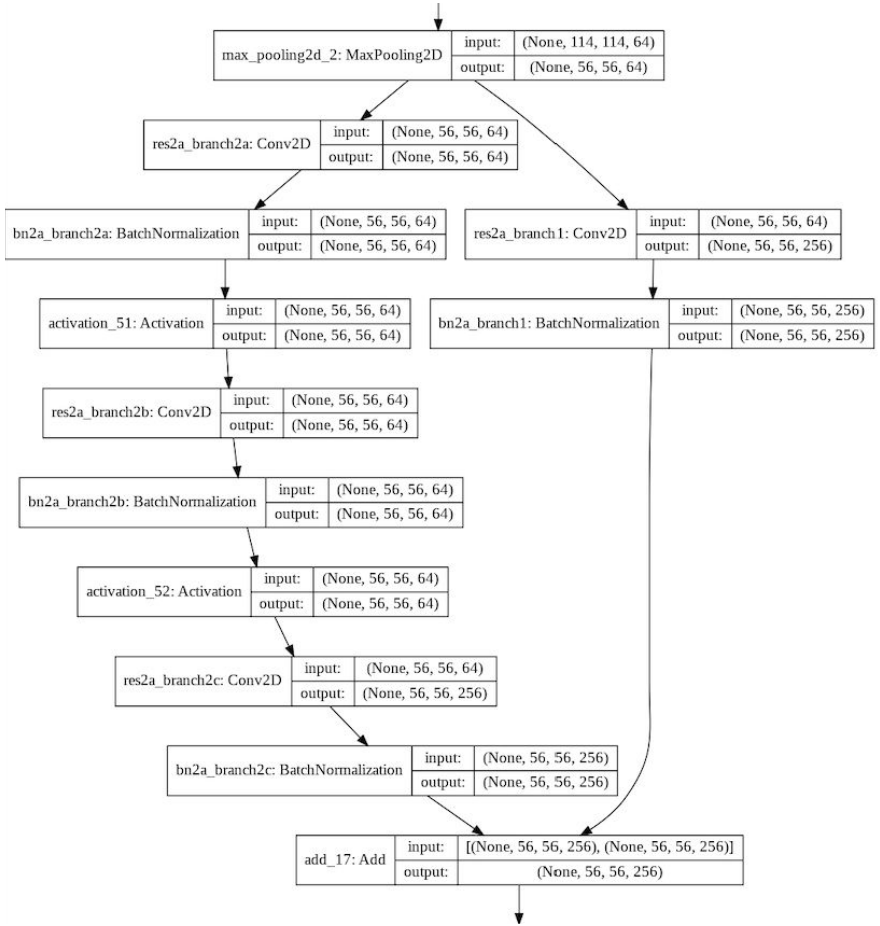
Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

ResNet



Example network architectures for ImageNet. **Bottom:** the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle:** a plain network with 34 parameter layers (3.6 billion FLOPs). **Top:** a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions.

ResNet-ting 64 channels to 256 channels (in ResNet50)



https://keras.io/applications/

The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

Depth refers to the topological depth of the network. This includes activation layers, batch normalization layers etc.

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
ResNeXt50	96 MB	0.777	0.938	25,097,128	-
ResNeXt101	170 MB	0.787	0.943	44,315,560	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

Inception Network

Reference: https://www.coursera.org/learn/convolutional-neural-networks

1 x 1 Convolutions Can Do Something Useful

Input Image

Only 1 channel

*

1 x 1 filter

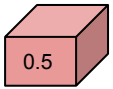
=

Activation is simply Input / 2

Input Image

3 channels

*

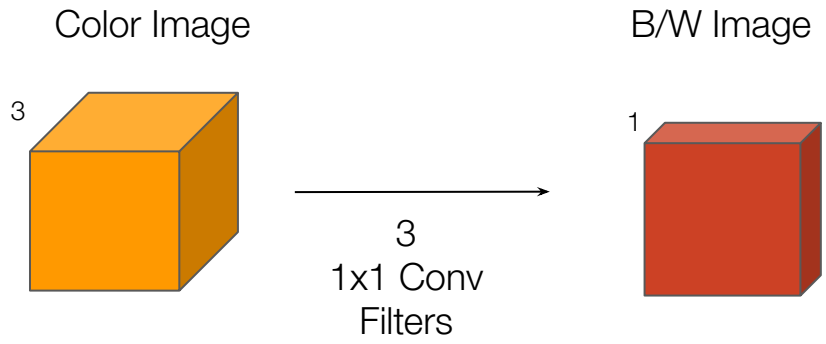


1 x 1 filter

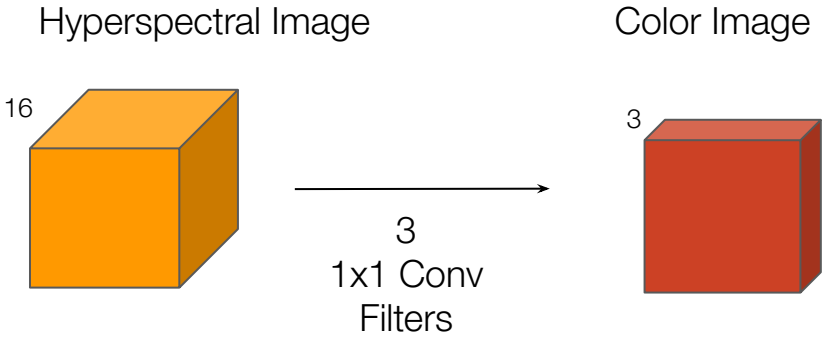
=

Activations represents some inter-channel relationship

1 x 1 Convolutions Can Do Something Useful

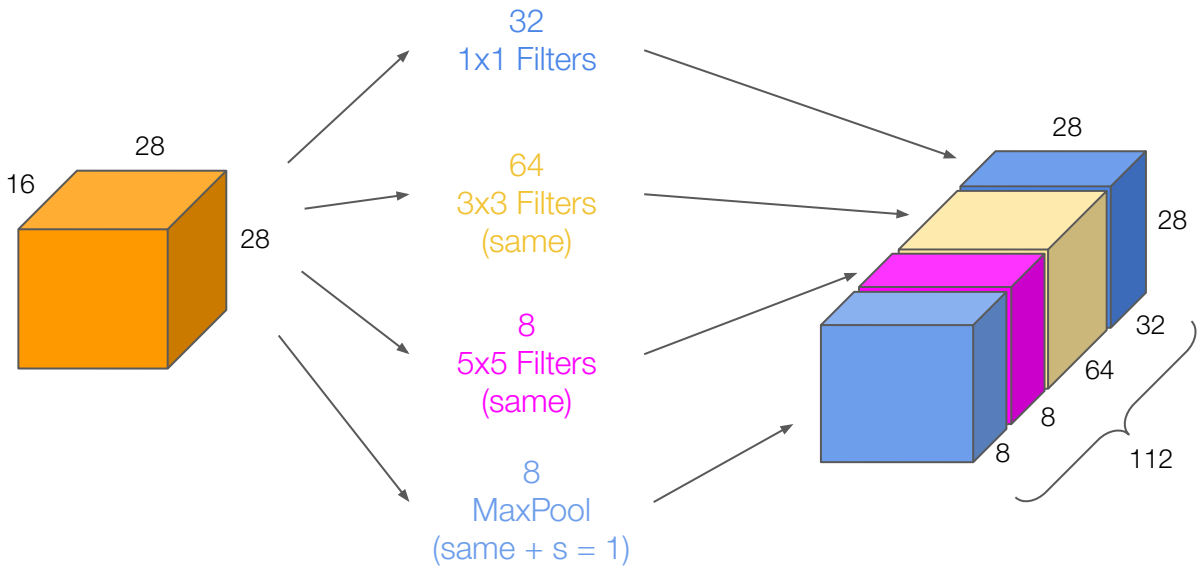


Shrink The Number of Channels



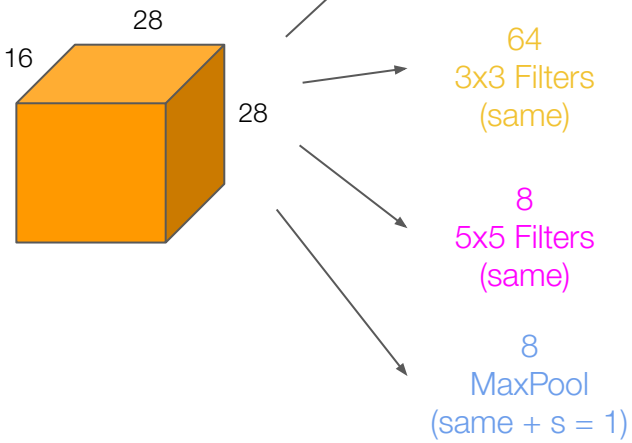
Inception Network - Motivation

- Should we use 1 x 1 Convolutions or 3 x 3, or 5 x 5, or Max-Pooling?
  - Inception networks attempt to resolve this
- Instead of choosing one, it does “all”



Which one of the 4 is most computationally expensive?

Inception Network - Expensive Calculations



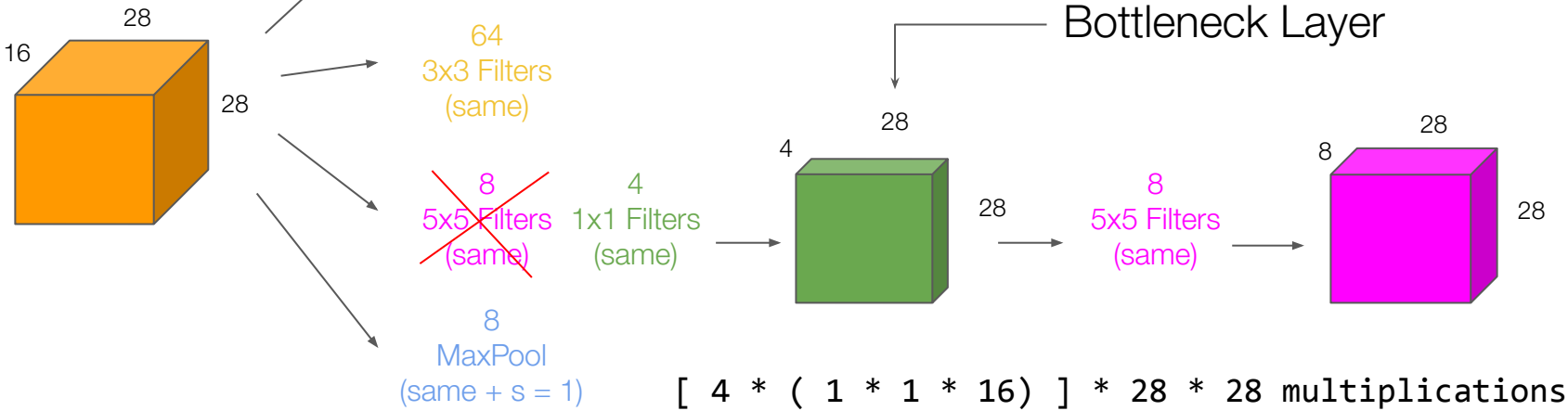

```
[ 32 * ( 1 * 1 * 16) ] * 28 * 28 multiplications
= 401 K
```

```
[ 64 * ( 3 * 3 * 16) ] * 28 * 28 multiplications
= 7,225 K
```

```
[ 8 * ( 5 * 5 * 16) ] * 28 * 28 multiplications
= 2,508 K
= 20,070 K (if we had 64 filters)
```

```
[ 16 ] [ 2 * 2 ] * 28 * 28 comparisons
= 50 K
```

Inception Network - 1 x 1 Conv. as Bottleneck Layer




```
[ 8 * ( 5 * 5 * 16) ] * 28 * 28 multiplications
= 2,508 K
```

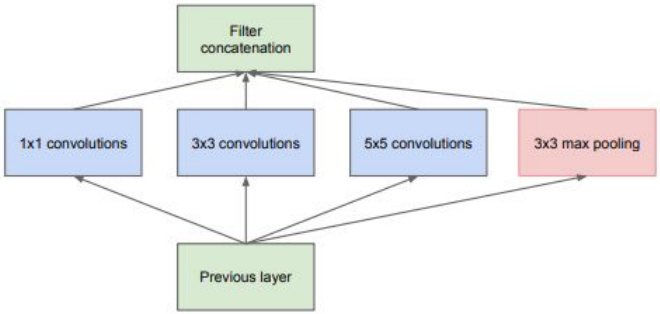
```
[ 4 * ( 1 * 1 * 16) ] * 28 * 28 multiplications
= 50 K
                +
[ 8 * ( 5 * 5 * 4 ) ] * 28 * 28 multiplications
= 627 K
```

If we had 'shrunk' the number of channels from 256 to 64 (using 1 x 1 conv.) the computational savings would be more!

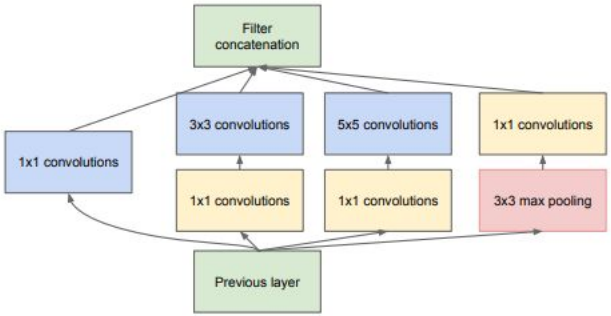
As long as we have a 'reasonable' shrinking ratio, it does not hurt the performance.

Inception Module in GoogLeNet

- They proposed deep convolutional neural network architecture codenamed Inception
  - Achieves the new state of the art for classification and detection in the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14)
    - Particular ‘incarnation’ called GoogLeNet, a 22 layers deep network, was used in the ILSVRC14
  - The main hallmark is - improved utilization of the computing resources inside the network



(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

Going Deeper with Convolutions

Christian Szegedy[1], Wei Liu[2], Yangqing Jia[1], Pierre Sermanet[1], Scott Reed[3], Dragomir Anguelov[1], Dumitru Erhan[1], Vincent Vanhoucke[1], Andrew Rabinovich[4]
[1]Google Inc. [2]University of North Carolina, Chapel Hill
[3]University of Michigan, Ann Arbor [4]Magic Leap Inc.
[1]{szegedy, jiayq, sermanet, dragomir, dumitru, vanhoucke}@google.com
[2]wliu@cs.unc.edu, [3]reedscott@umich.edu, [4]arabinovich@magicleap.com

GoogLeNet

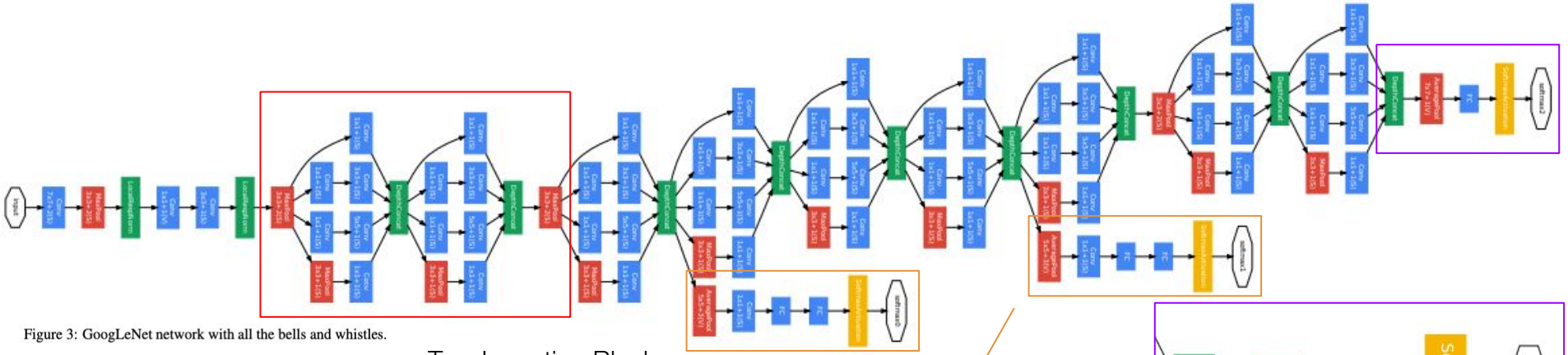
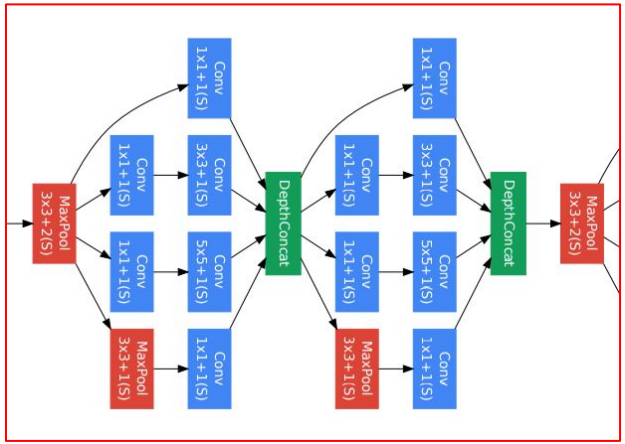
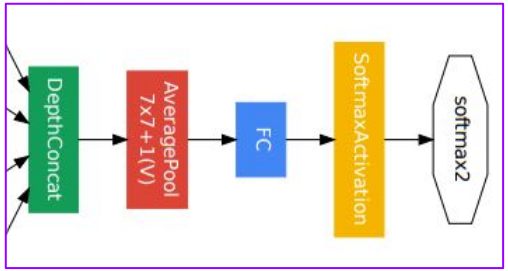


Figure 3: GoogLeNet network with all the bells and whistles.

Two Inception Blocks





- Act as a regularizer
- Help the network focus on the output
- An 'alternative' to residual connections
- ResNets were introduced in the subsequent year

Going Deeper with Convolutions

Christian Szegedy[1], Wei Liu[2], Yangqing Jia[1], Pierre Sermanet[1], Scott Reed[3], Dragomir Anguelov[1], Dumitru Erhan[1], Vincent Vanhoucke[1], Andrew Rabinovich[4]
[1]Google Inc. [2]University of North Carolina, Chapel Hill
[3]University of Michigan, Ann Arbor [4]Magic Leap Inc.
[1]{szegedy, jiayq, sermanet, dragomir, dumitru, vanhoucke}@google.com
[2]wliu@cs.unc.edu, [3]reedscott@umich.edu, [4]arabinovich@magicleap.com

DenseNet

DenseNet

- Idea: Connect each Conv. layer to every other layer in a feed-forward fashion
- Traditional convolutional networks (Resnets) with L layers have L connections
  - one between each layer and its subsequent layer

- A DenseNet has $\frac{L*(L-1)}{2}$ direct connections

How can we connect a layer with N channels to a layer with M channels?

Densely Connected Convolutional Networks

Gao Huang*
Cornell University
gh349@cornell.edu

Zhuang Liu*
Tsinghua University
liuzhuang13@mails.tsinghua.edu.cn

Laurens van der Maaten
Facebook AI Research
lvdmaaten@fb.com

Kilian Q. Weinberger
Cornell University
kqw4@cornell.edu

arXiv:1608.06993v5 [cs.CV] 28 Jan 2018

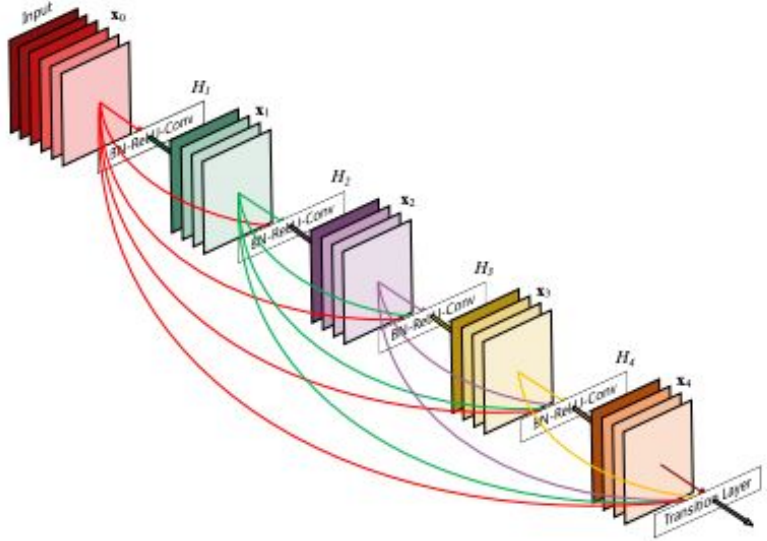


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

DenseNet

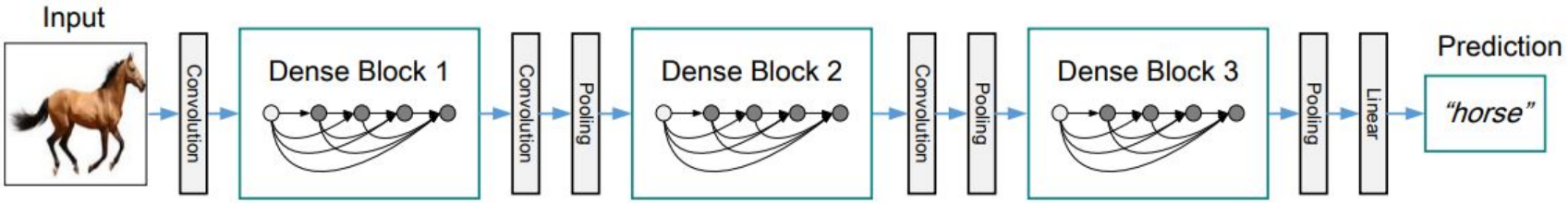


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

Growth rate:

- The l^{th} layer in a dense network has $k_0 + k * (l-1)$ input feature maps
  - Each convolution has k filters
  - k_0 is the input feature maps
- The parameter k is known as 'growth rate'
- An important difference between DenseNet and ResNets is that
  - DenseNet can have very narrow layers, e.g., k = 12

DenseNet

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv, 3 × 3 conv] × 24	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48	[1 × 1 conv, 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv, 3 × 3 conv] × 16	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Table 1: DenseNet architectures for ImageNet. The growth rate for all the networks is $k = 32$. Note that each "conv" layer shown in the table corresponds the sequence BN-ReLU-Conv.

This table does not discuss how the number of channels increase/change over depth

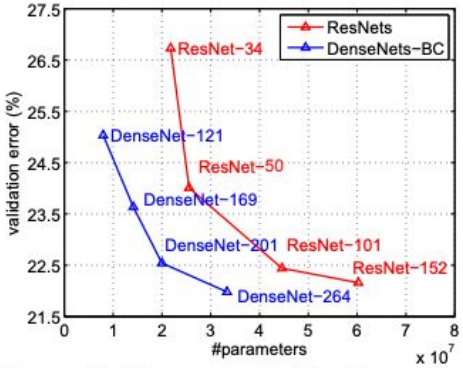


Figure 3: Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*)

Plotting the DenseNet Model

DEMO

Branch: master ▾ **2019-Spring-DL** / course_content / module3_convnets / **DenseNet121_plot_model.ipynb** Find file | Copy path

badriadhikari Created using Colaboratory c3ced88 12 hours ago

1 contributor

73 lines (73 sloc) | 1.79 KB Raw | Blame | History

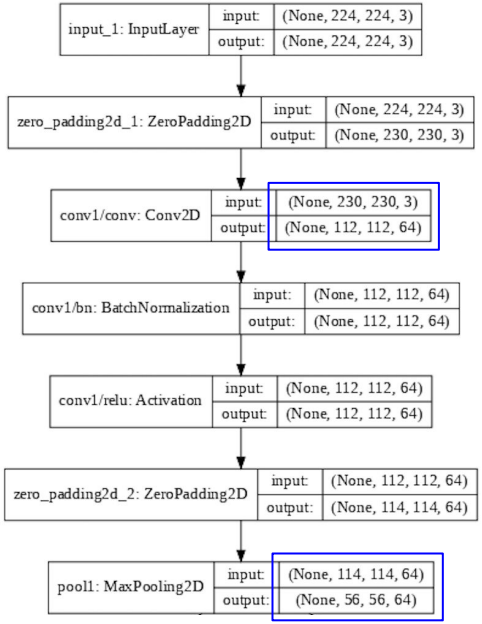
Open in Colab

```
In [0]: from keras.applications.densenet import DenseNet121
        model = DenseNet121(weights='imagenet')
```

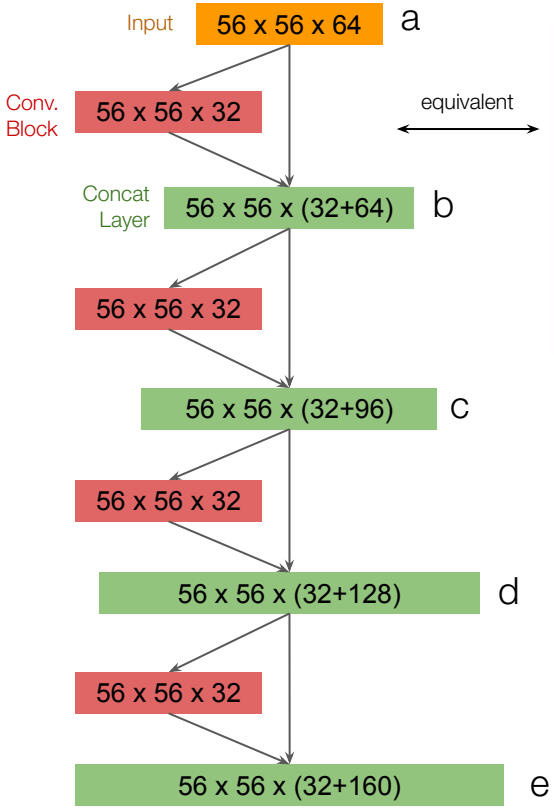
```
In [0]: from keras.utils import plot_model
        plot_model(model, show_shapes=True, show_layer_names=True, to_file='DenseNet121.png')

        from google.colab import files
        files.download('DenseNet121.png')
```

Plotting the DenseNet Model

Initial Layers

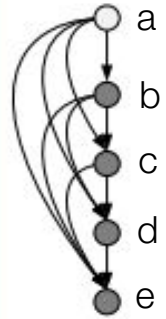


plot_model

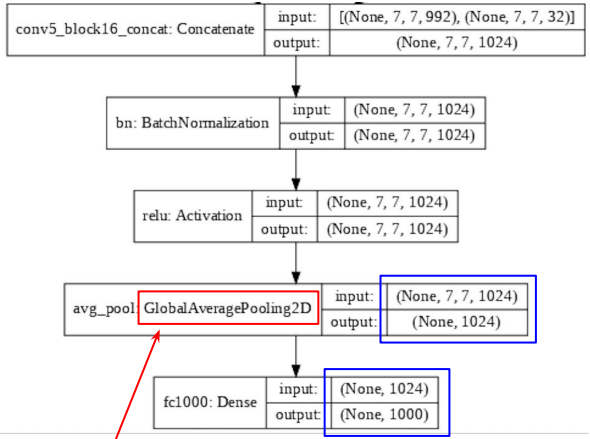


Physical

(how we understand)



Final Layers



It allows you to have the input image be any size, not just a fixed size like 227x227

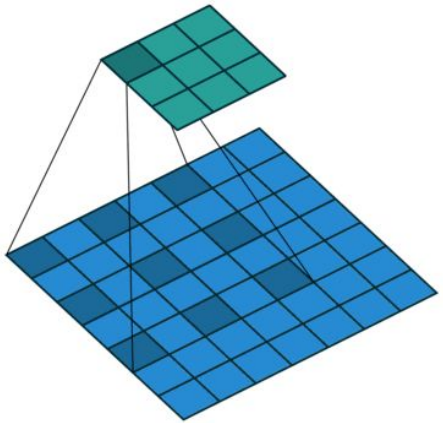
Different Types of Convolutions

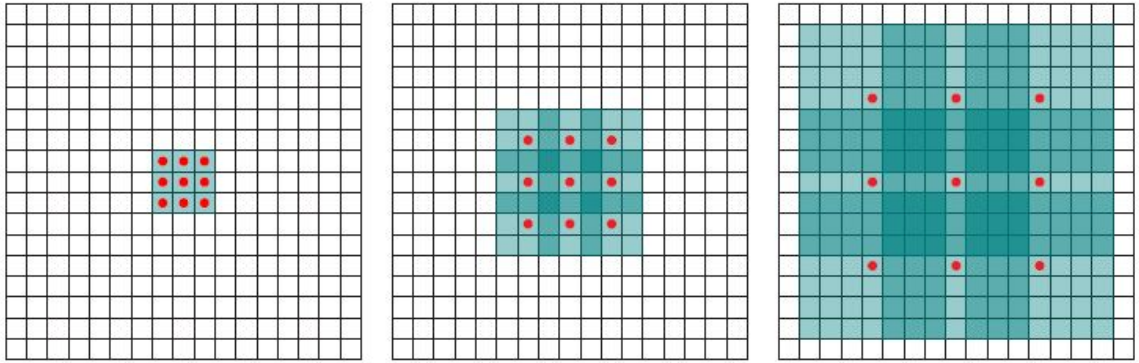
https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d &
https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728

Dilated Convolutions

```
keras.layers.Conv2D(filters, kernel_size, dilation_rate=(1, 1))
```

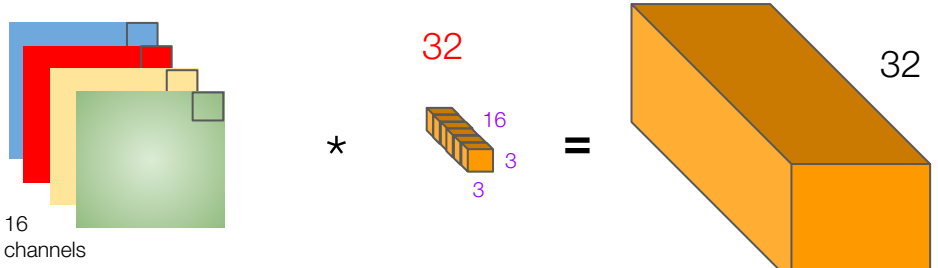
Dilation rate defines a spacing between the values in a kernel.





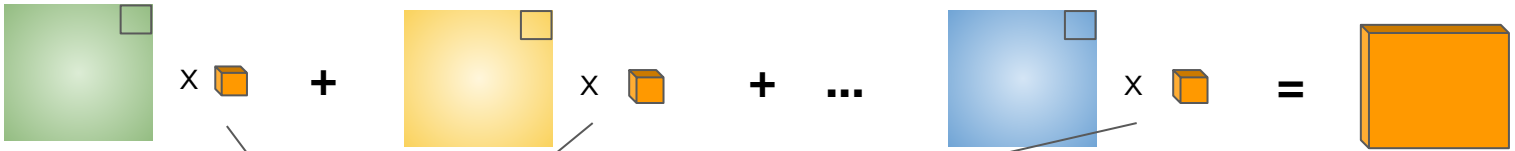
- For increasing the receptive field - Pooling and/or Strided Convolutions are common
  - But both reduce the resolution
- Dilated convolutions allow exponential expansion of the receptive field without loss of resolution
  - With same computation and memory costs
    - Practically, however, it is not equally ‘exciting’ for all problems!

Depthwise Separable Convolutions

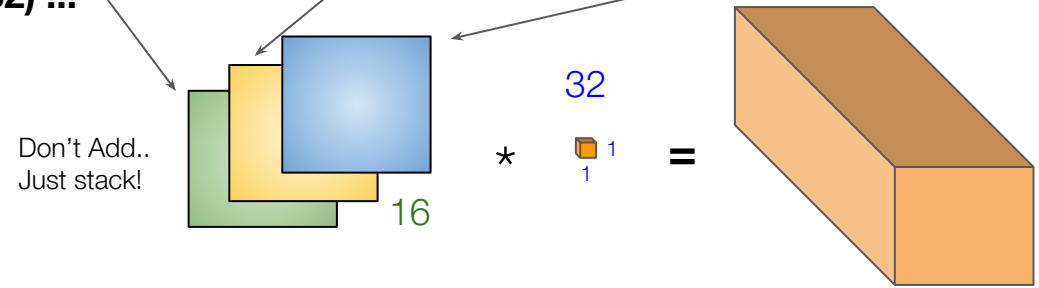


# of params = 32 x 3 x 3 x 16

Each convolution may be seen as ...



Depthwise Separable (32) ...



# of params = 16 x 3 x 3 + 32 x 1 x 1 x 16

NASNet

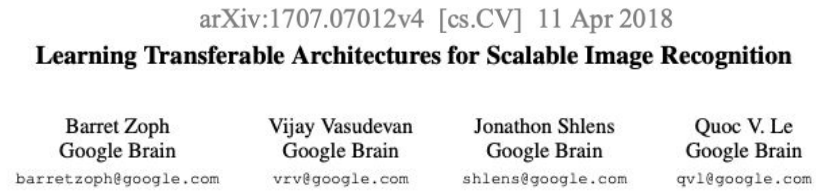
NASNet

- Inspired by the Neural Architecture Search (NAS) framework
  - Which uses a reinforcement learning search method to optimize architecture configurations
- Applying NAS to a large dataset (e.g. ImageNet), is computationally expensive
  - Search for a good architecture on a proxy dataset, for example the smaller CIFAR-10 dataset, and then transfer the learned architecture to ImageNet
- The pool of workers in the workqueue consisted of 500 GPUs
  - Search process over 4 days yields several candidate convolutional cells


Under review as a conference paper at ICLR 2017

NEURAL ARCHITECTURE SEARCH WITH REINFORCEMENT LEARNING

Barret Zoph, Quoc V. Le
Google Brain
{barretzoph,qvl}@google.com


arXiv:1707.07012v4 [cs.CV] 11 Apr 2018

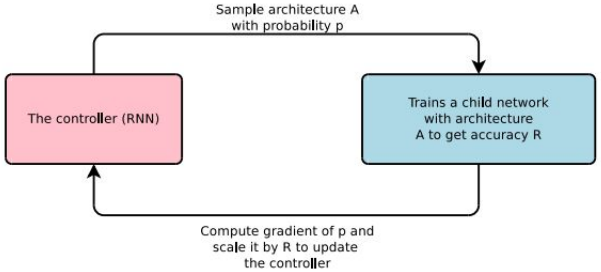
Learning Transferable Architectures for Scalable Image Recognition

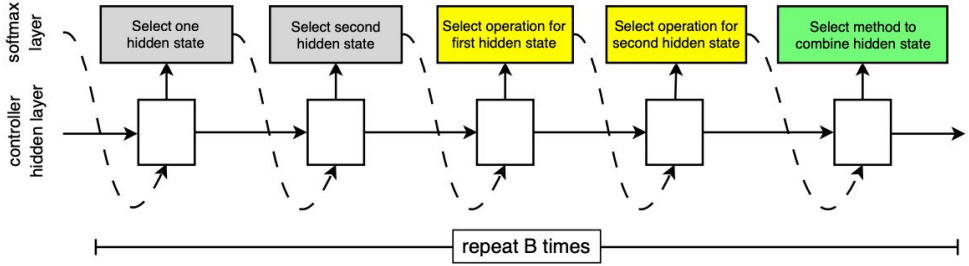
Barret Zoph, Google Brain, barretzoph@google.com
Vijay Vasudevan, Google Brain, vrv@google.com
Jonathon Shlens, Google Brain, shlens@google.com
Quoc V. Le, Google Brain, qvl@google.com

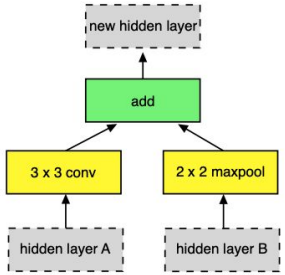
NASNet

NAS Algorithm:

- In NAS, a controller recurrent neural network (RNN) samples child networks with different architectures
- The child networks are trained to convergence to obtain some accuracy on a held-out validation set
- The resulting accuracies are used to update the controller so that the controller will generate better architectures over time. The controller weights are updated with policy gradient.







- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise-seperable conv

Figure 3. Controller model architecture for recursively constructing one block of a convolutional cell. Each block requires selecting 5 discrete parameters, each of which corresponds to the output of a softmax layer. Example constructed block shown on right. A convolutional cell contains B blocks, hence the controller contains $5B$ softmax layers for predicting the architecture of a convolutional cell. In our experiments, the number of blocks B is 5.

NASNet - Architecture of the Best Convolutional Cells

- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise-seperable conv

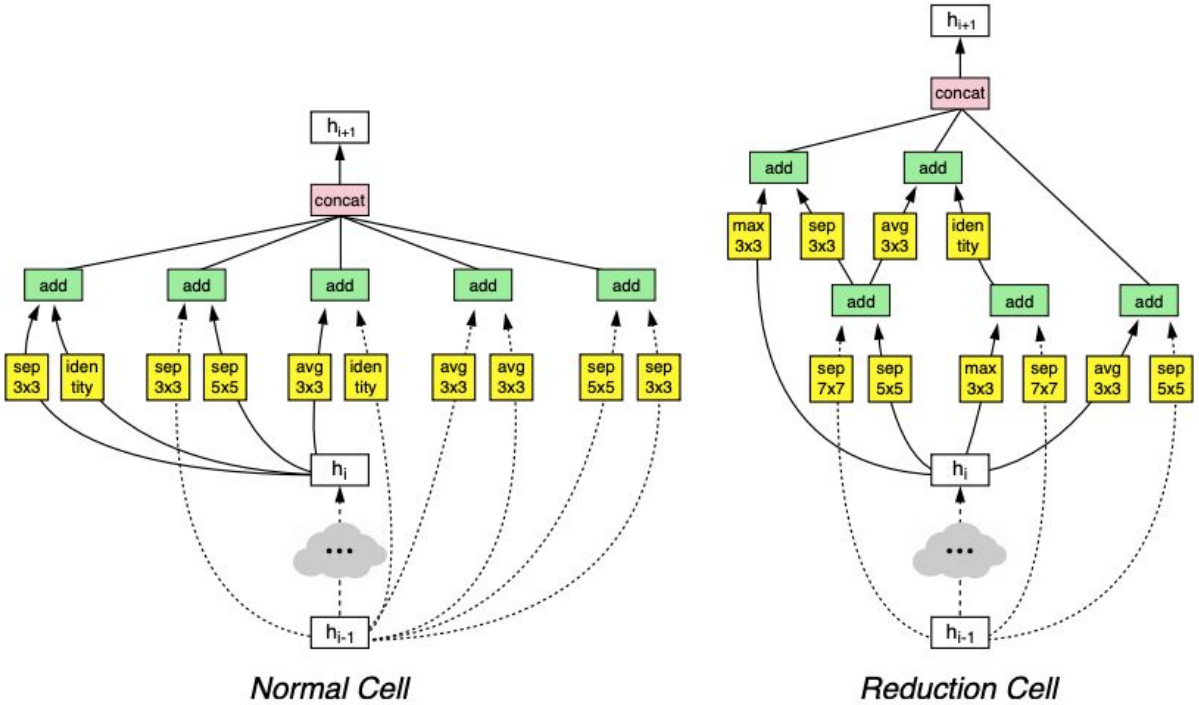


Figure 4. Architecture of the best convolutional cells (NASNet-A) with $B = 5$ blocks identified with CIFAR-10 . The input (white) is the hidden state from previous activations (or input image). The output (pink) is the result of a concatenation operation across all resulting branches. Each convolutional cell is the result of B blocks. A single block is corresponds to two primitive operations (yellow) and a combination operation (green). Note that colors correspond to operations in Figure 3.

NASNet

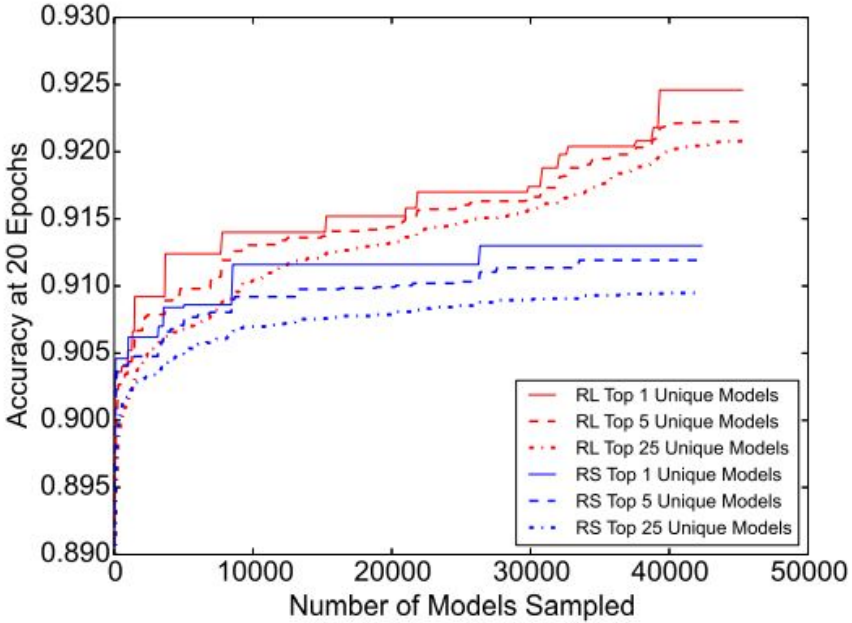


Figure 6. Comparing the efficiency of random search (RS) to reinforcement learning (RL) for learning neural architectures. The x-axis measures the total number of model architectures sampled, and the y-axis is the validation performance on CIFAR-10 after 20 epochs of training.

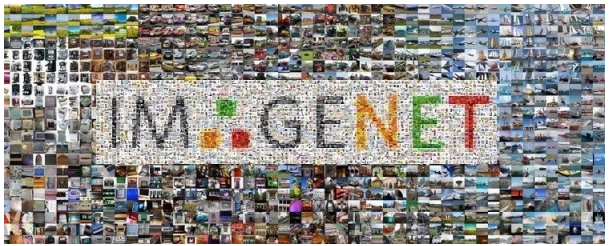
Deep Transfer Learning

5.3 Using a Pre-trained Convnet

- A pretrained network is a saved network that was previously trained
  - on a large dataset, typically on a large-scale image-classification task
- If we train a network on “ImageNet”
  - We can repurpose this trained network for something else
  - Even as remote as identifying furniture items in images, will work fine
  - Some have even applied to medical imaging (pictures taken by regular cameras)
- “Portability of learned features across different problems” is a key advantage of deep learning compared to many older, shallow-learning approaches
  - It makes deep learning very effective for small-data problems
- There are two ways to use a pretrained network:
  - **feature extraction** and **fine-tuning**

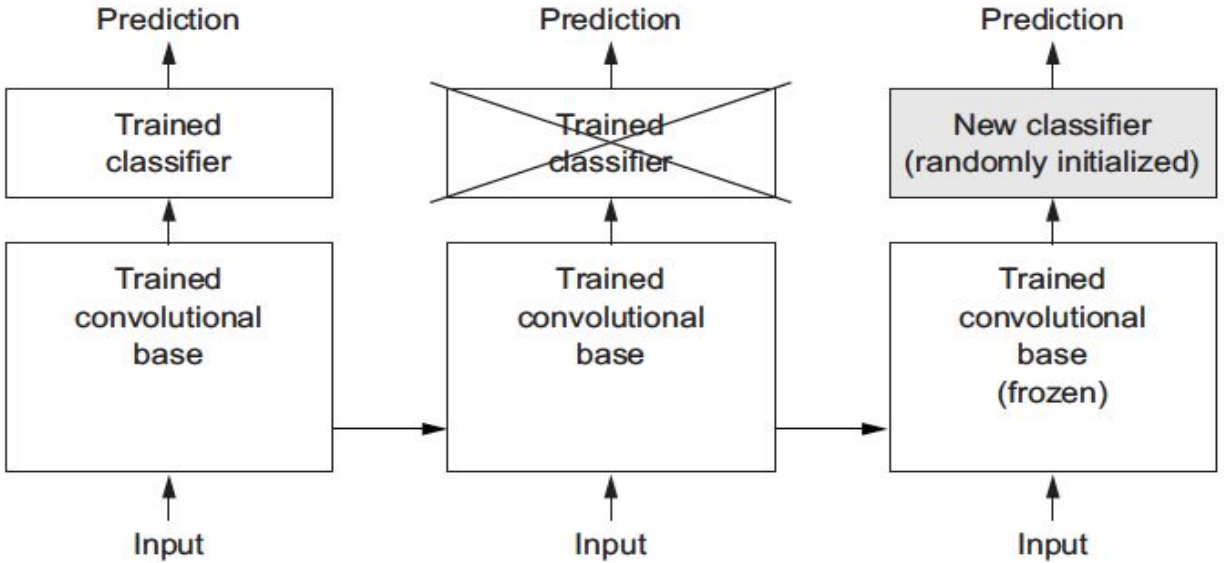
14 million images, 1000 classes

Classes are mostly animals and everyday objects



5.3.1 Feature Extraction

- Feature extraction is taking the ‘convolutional base’ of a previously trained network, running the new data through it, and training a new classifier on top of the output
- Why skip the dense layers?
  - Representations learned by the convolutional base are more generic and therefore more reusable



DEMO

Branch: master ▾ 2019-Spring-DL / course_content / module3_convnets / Transfer_Learning_VGG16_for_CIFAR10.ipynb

Find file | Copy path

badriadhikari Created using Colaboratory a9d0dff 2 minutes ago

1 contributor

844 lines (844 sloc) | 94.9 KB

Raw | Blame | History

Open in Colab

Using the VGG16 Convolutional Base for CIFAR-10 Dataset

The CIFAR-10 Dataset

- The CIFAR-10 dataset (Canadian Institute For Advanced Research) is a collection of images that are commonly used to train machine learning and computer vision algorithms
- It is one of the most widely used datasets for machine learning research
- The CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes
- The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks
- There are 6,000 images of each class

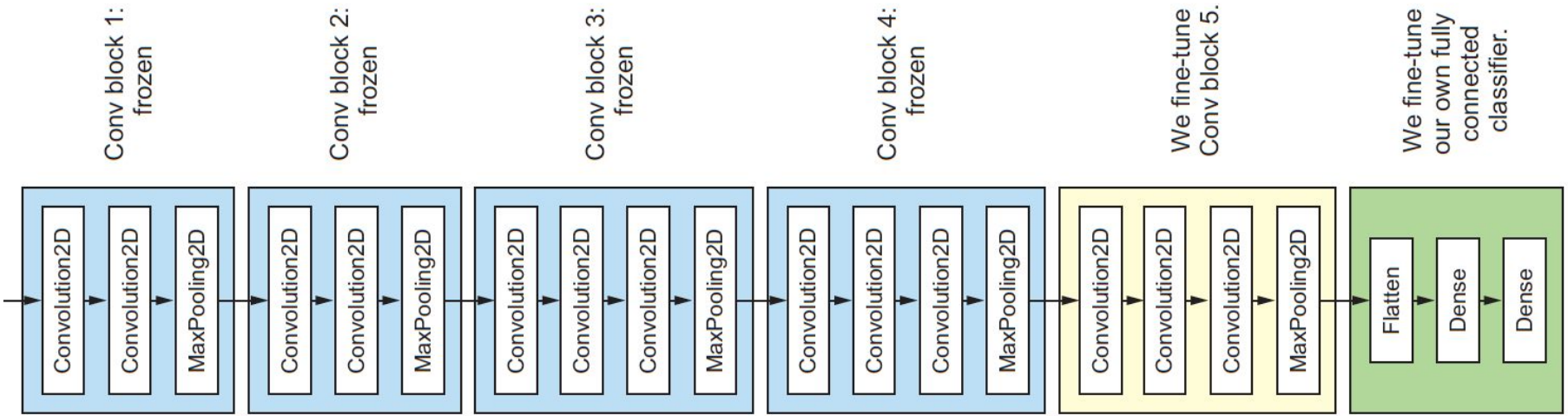
In [1]:
```
from keras.datasets import cifar10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
Using TensorFlow backend.
```

In [2]:
```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

5.3.2 Fine Tuning

- Unfreezing a few of the top layers of a frozen model base used for feature extraction
- Jointly train - both - (a) the newly added part of the model and (b) these top layer
- Why is it called fine-tuning?
  - Because we slightly adjust the more abstract representations of the model being reused, in order to make them more relevant for the problem at hand



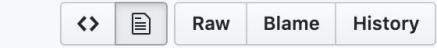
5.3.2 Fine Tuning

Steps:

- Add your custom network on top of an already-trained base network
- Freeze the base network
- Train the part you added
- Unfreeze some layers in the base network
- Jointly train both these layers and the part you added

DEMO

520 lines (520 sloc) | 49.6 KB



Fine Tuning the VGG16 Model for CIFAR-10 Dataset

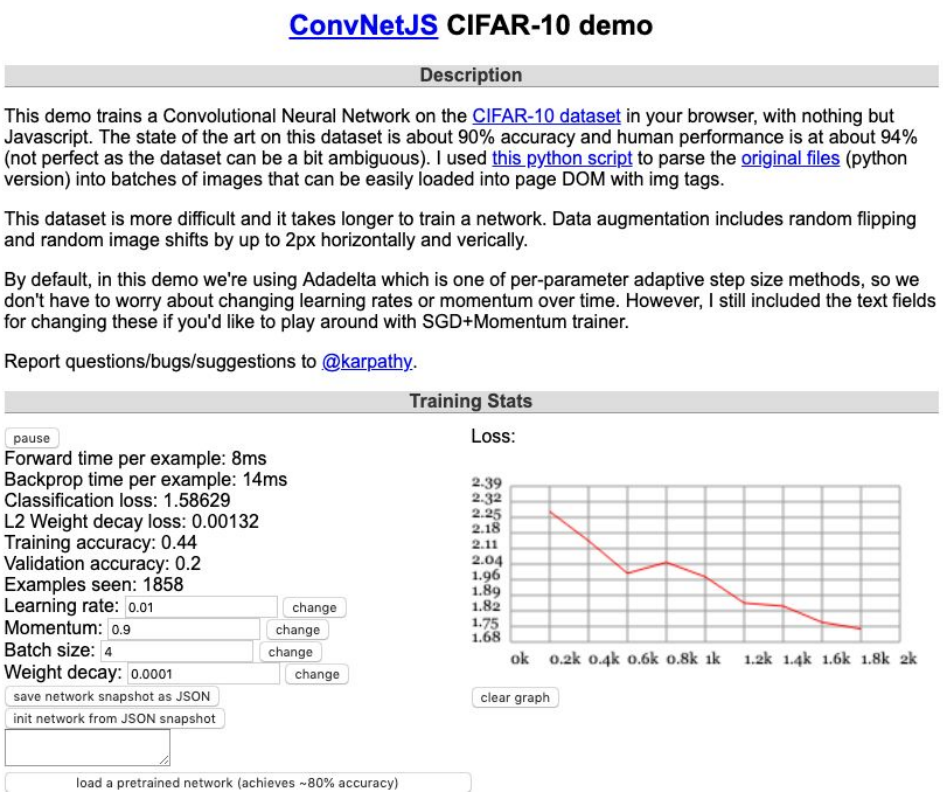
The CIFAR-10 Dataset

- The CIFAR-10 dataset (Canadian Institute For Advanced Research) is a collection of images that are commonly used to train machine learning and computer vision algorithms
- It is one of the most widely used datasets for machine learning research
- The CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes
- The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks
- There are 6,000 images of each class

In [1]:
```
from keras.datasets import cifar10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
Using TensorFlow backend.
```

Train a ConvNet In Your Browser



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