Applied Computer Vision

David Vernon
Carnegie Mellon University Africa

vernon@cmu.edu
www.vernon.eu

Lecture 28

Computer Vision and Deep Learning III

Convolutional Neural Networks

Differences from traditional neural networks

- Last layer is the only fully connected FC layer
- New type of layer – convolution CONV – filters the images
  - Hundreds to thousands of filters
  - Filters are learned
  - Most filters in CNNs are square n x n matrices, where n is odd
- Non-linear activation function, e.g. RELU, is applied to CONV layers
- Multiple sequences of CONV => ReLU layers

Convolutional Neural Networks

Differences from traditional neural networks

- Layers in a CNN are arranged in a 3D volume in three dimensions
  - Width
  - Height
  - Depth
    - Number of channels in an image
    - Number of filters in a layer
- For example, CIFAR10:
  - Input layer 32 x 32 x 3
  - Output layer 1 x 1 x 10 … Single vector with ten class scores (or probabilities)

Convolutional Neural Networks

Key benefits

- Local invariance / translation invariance: classification not sensitive to position of the object in the image
  - Achieve using pooling layers POOL
- Compositionality: later layers build increasingly rich features (representations) by building on features detected in earlier layers

Convolutional Neural Networks

Layer Types

Convolutional (CONV)

Activation (ACT or RELU)

Pooling (POOL)

Fully-connected (FC)

Batch normalization (BN)

Dropout (DO)

Stacking layers yields a CNN

INPUT => CONV => RELU => FC => SOFTMAX

Only layers that have parameters
that are learned during the training process

Convolutional Neural Networks

Layer Types

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

Activation and Dropout layers
are not considered "true" layers
but are often included
to make the architecture explicit

Stacking layers yields a CNN

INPUT => CONV => RELU => FC => SOFTMAX

Convolutional Neural Networks

Layer Types

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

Stacking layers yields a CNN

INPUT => CONV => RELU => FC => SOFTMAX

Often omitted and
simply assumed it follows FC

Convolutional Neural Networks

Layer Types

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

The most important when defining the network architecture

Stacking layers yields a CNN

INPUT => CONV => RELU => FC => SOFTMAX

Convolutional Neural Networks

Layer Types

Convolutional (CONV)

Activation (ACT or RELU)

Pooling (POOL)

Fully-connected (FC)

Batch normalization (BN)

Dropout (DO)

Practically assumed to be part of the architecture

Often omitted from architecture table/diagram to save space

But are implicitly assumed to be part of the architecture

Stacking layers yields a CNN

INPUT => CONV => RELU => FC => SOFTMAX

Convolutional Neural Networks

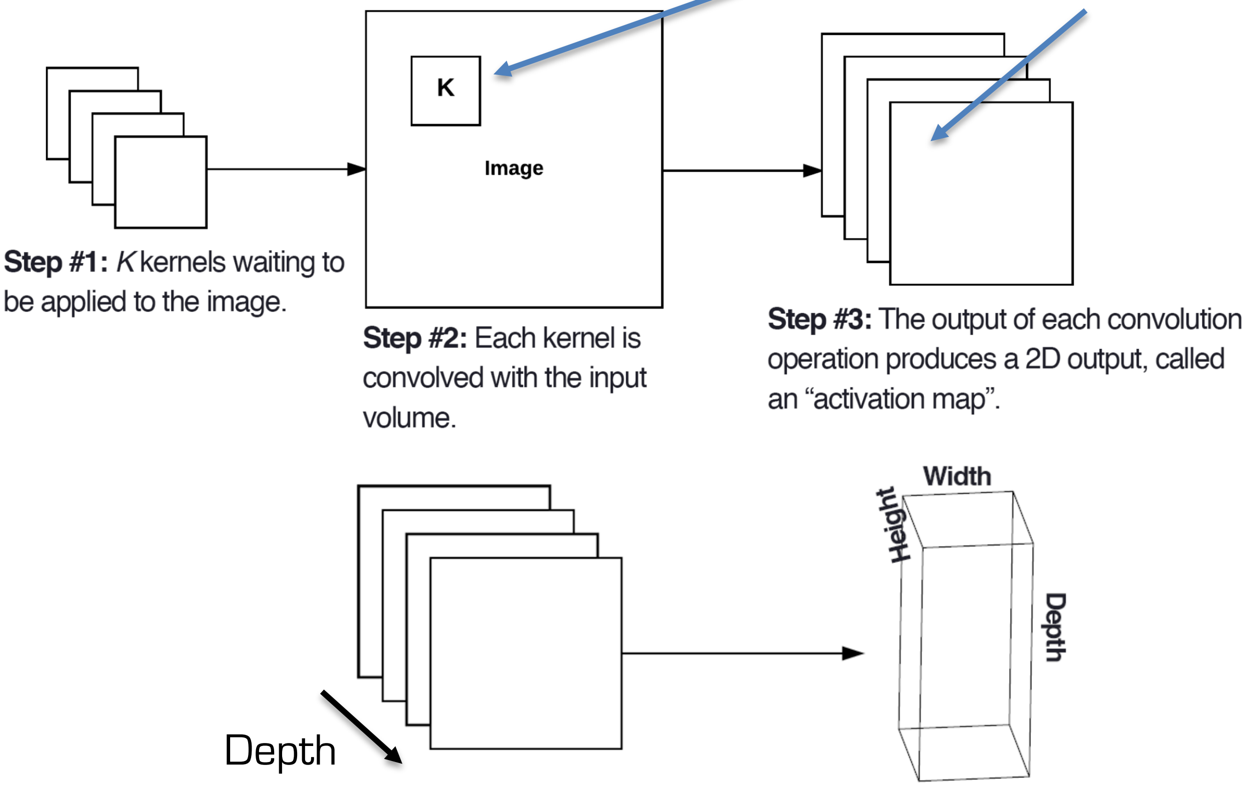
Convolutional Layers CONV

- The CONV layer parameters consist of a set of K learnable filters (kernels)
- Width normally equals height (i.e. square kernel)
- Usually small size (e.g. 3 x 3, 5 x 5, ...)
- Extend throughout the full depth of the volume
  - Number of channels (input layer)
  - Number of filters applied in previous layer (deeper in the network)

Convolutional Neural Networks

Convolutional Layers CONV

Receptive Field
of a single neuron in the output
= size of the filter kernel



Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Convolutional Neural Networks

Convolutional Layers CONV

- Depth
  - Number of filters K in the CONV layer that connect to a local region in the input volume
  - The set of filters that are "looking at" the same (x, y) location of the input is call the depth column

- Stride
  - The step size S (in pixels) between each application of a convolution kernel
  - Typically, $S = 1$ or $S = 2$
  - Larger strides imply less overlap in receptive field
  - Larger strides generate outputs with reduced spatial dimensions

Convolutional Neural Networks

Convolutional Layers CONV

- Padding
  - Enlarge the input to ensure that the output spatial dimensions = input spatial dimensions
  - Assuming stride of 1

Zero-padding

692	-315	-6
-680	-194	305
153	-59	-86

0	0	0	0	0	0	0
0	95	242	186	152	39	0
0	39	14	220	153	180	0
0	5	247	212	54	46	0
0	46	77	133	110	74	0
0	156	35	74	93	116	0
0	0	0	0	0	0	0

-99	-673	-130	-230	176
-42	692	-315	-6	-482
312	-680	-194	305	124
54	153	-59	-86	-24
-543	167	-35	-72	-297

  - To ensure the can be tiled such that they fit across the input volume

$$(W - F + 2\ P)\ /\ S) + 1$$ must be an integer

Size of (square) input

Receptive Field

Padding

Stride

Convolutional Neural Networks

Convolutional Layers CONV

To summarize, the `CONV` layer in the same, elegant manner as Karpathy [121]:

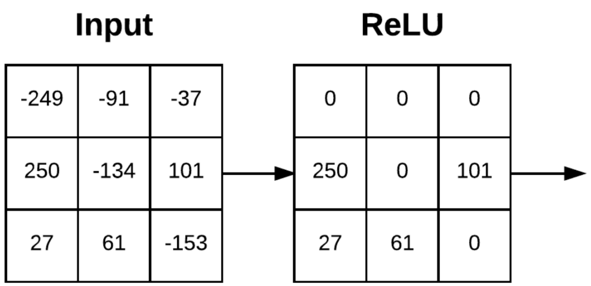
- Accepts an input volume of size $W_{input} \times H_{input} \times D_{input}$ (the input sizes are normally square, so it's common to see $W_{input} = H_{input}$).
- Requires four parameters:
  1. The number of filters K (which controls the *depth* of the output volume).
  2. The receptive field size F (the size of the K kernels used for convolution and is nearly always *square*, yielding an $F \times F$ kernel).
  3. The stride S.
  4. The amount of zero-padding P.
- The output of the `CONV` layer is then $W_{output} \times H_{output} \times D_{output}$, where:
  - $W_{output} = ((W_{input} - F + 2P)/S) + 1$
  - $H_{output} = ((H_{input} - F + 2P)/S) + 1$
  - $D_{output} = K$

121. Andrej Karpathy. *Convolutional Networks*. http://cs231n.github.io/convolutional- networks/

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Convolutional Neural Networks

Activation Layers ACT, RELU, ELU, ...



- Apply a nonlinear activation function after each CONV layer

- Technically not "layers" since no weights are learned during training

- Sometimes omitted (and assumed to be present)

  INPUT => CONV => RELU => FC
  INPUT => CONV => FC

- Dimensions of the output volume is the same as the input volume
  - Activation function is applied to each neuron individually

Convolutional Neural Networks

Pooling Layers POOL

- Used to reduce size (width, height) of an input volume
- Can also use CONV with a stride > 2
- Insert POOL in between consecutive CONV layers

  INPUT => CONV => RELU => POOL => CONV => RELU => POOL => FC

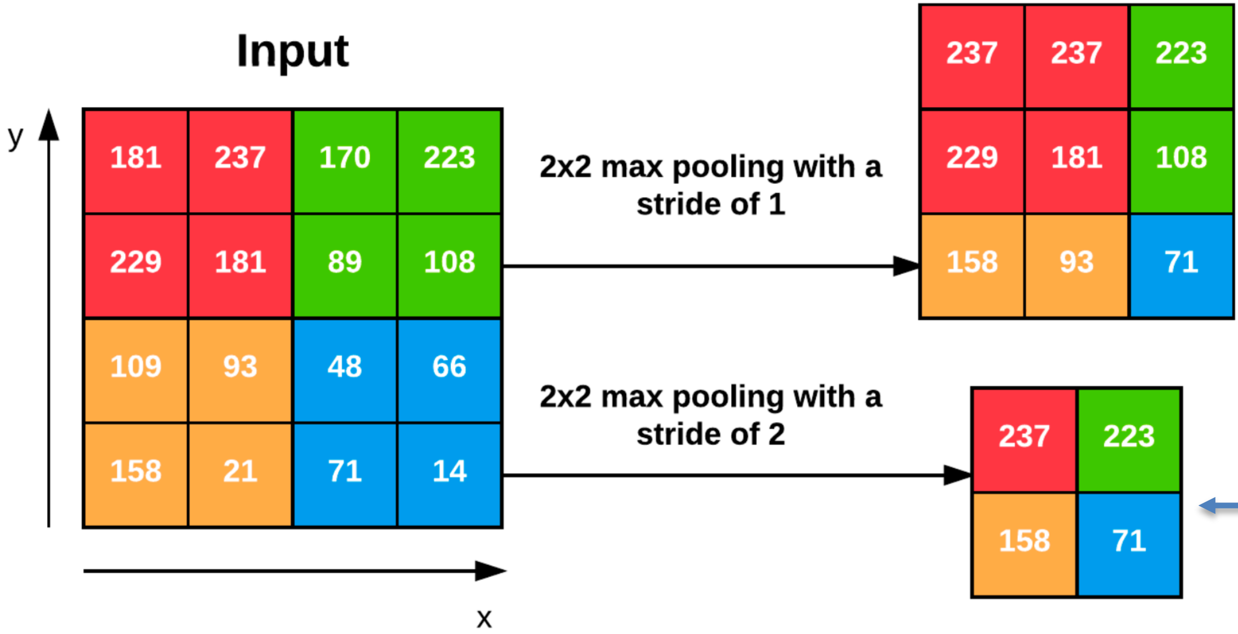
- Also helps control overfitting
- Operate on each depth slice independently
  - Max function
  - Average function

Convolutional Neural Networks

Pooling Layers POOL

- Typically, pool size of 2 x 2; images > 200 pixels may use 3 x 3
- Stride of 1 or 2

Overlapping pooling:
typically used with images with
large spatial dimensions



Non-overlapping pooling:
typically used with images with
small spatial dimensions

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Convolutional Neural Networks

Fully-connected Layers FC

- Neurons in FC layers are fully-connected to all activation outputs in the previous layer

- FC are always placed at the end of the network (and only at the end)

- It is common to use two FC layers prior to applying the softmax classifier

INPUT => CONV => RELU => POOL => CONV => RELU => POOL => FC => FC

Convolutional Neural Networks

Batch Normalization BN

- Batch normalization layers are used to normalize the activations of a given input volume before passing them to the next layer
  - Reduces the number of epochs needed to train the network
  - "Stabilizes" training: makes learning rate and regularization easier to tune
  - Lower final loss & more stable loss curve
  - Helps prevent overfitting
  - Use it in nearly every situation
- Apply BN after RELU: INPUT => CONV => RELU => BN ...

Convolutional Neural Networks

Batch Normalization BN

Unnormalized Activation

Mean and standard deviation of activation value in each mini-batch β

$$\hat{x}_i = \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \varepsilon}}$$

Small positive value e.g. 1 E-7

Normalized Activation

Produces values with approx. zero mean and unit variance, i.e. zero-centred values

$$\mu_\beta = \frac{1}{M}\sum_{i=1}^{m} x_i$$

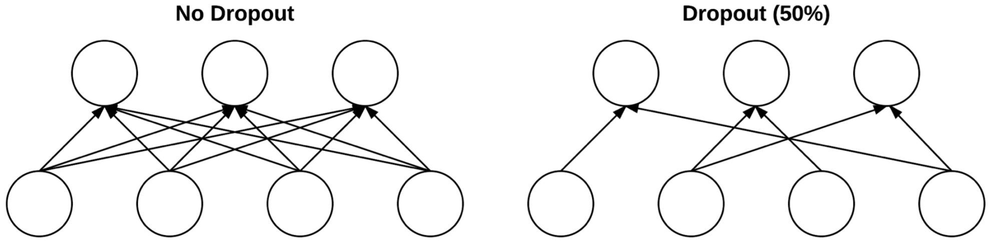
At testing time, replace mini-batch μ_β and σ_β with unning averages of μ_β and σ_β computed during the training process

$$\sigma_\beta^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\beta)^2$$

Convolutional Neural Networks

Dropout DO

- Regularization technique to help prevent overfitting
  - Increasing testing accuracy
  - Possibly at the expense of training accuracy
- Randomly disconnect connections between two FC layers; probability p (e.g. 0.5)
- ... CONV => RELU => POOL => FC => DO => FC => DO => FC

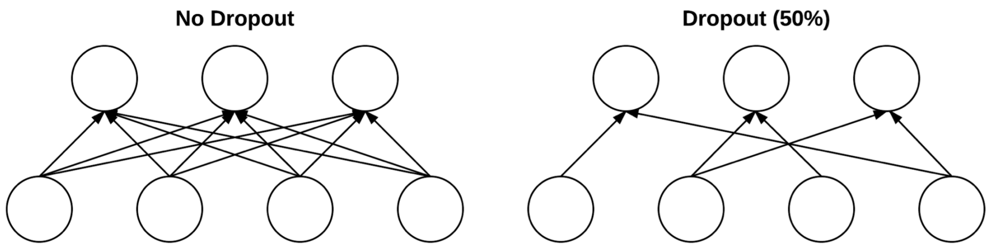


Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Convolutional Neural Networks

Dropout DO

- Randomly dropping connections ensures no single node in the network is responsible for “activating” when presented with a given pattern
- Ensures there are multiple redundant nodes
- Help the network to generalize



Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Convolutional Neural Networks

Common Architectures and Training Patterns

INPUT => [[CONV => RELU] * N => POOL?] * M => [FC => RELU] * K => FC

Typically:

0 <= N <= 3

M >= 0

0 <= K <= 2

AlexNet-like:

INPUT => [CONV => RELU => POOL] * 2 => [CONV => RELU] * 3 => POOL =>
[FC => RELU => DO] * 2 => SOFTMAX

VGGNet

INPUT => [CONV => RELU] * 2 => POOL => [CONV => RELU] * 2 => POOL =>
[CONV => RELU] * 3 => POOL => [CONV => RELU] * 3 => POOL =>
[FC => RELU => DO] * 2 => SOFTMAX

Convolutional Neural Networks

Common Architectures and Training Patterns

Apply deeper networks when we have

- lots of training data
- a sufficiently challenging classification problem

Deep Learning with Keras and Python

Installation instructions

`https://www.pyimagesearch.com/2017/09/25/configuring-ubuntu-for-deep-learning-with-python/`

Possible issues (these arose with Ubuntu 18 but may not with Ubuntu 16.04)

Ubuntu system dependencies

```
$ sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev
```

Error: No installable candidates

Solution: before this command, execute:

```
$ sudo add-apt-repository "deb http://security.ubuntu.com/ubuntu xenial-security main"
$ sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev
```

Compiling OpenCV

Error: Possible removal of the OpenCV folder

Do not execute the direct command (provided in the instructions)

```
$ rm -rf opencv-3.3.0 opencv.zip
$ rm -rf opencv_contrib-3.3.0 opencv_contrib.zip
```

Solution:

```
$ rm -rf opencv.zip
$ rm -rf opencv_contrib-3.3.0 opencv_contrib.zip
```

Deep Learning with Keras and Python

Software

- Software Development Environment
- ACV.zip
- opencv-2.4.10.exe
- chapter12-first_cnn.zip
- chapter15-minivggnet.zip
- chapter20-out_of_the_box.zip

Deep Learning with Keras and Python

When you import the Keras library into a Python script, it generate a keras.json configuration file in ~/.keras/keras.json

```
{
    "epsilon": 1e-07,
    "floatx": "float32",
    "image_data_format": "channels_last",
    "backend": "tensorflow"
}
```

rows, columns, channels

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

ShallowNet with Keras and Python

ShallowNet: INPUT => CONV => RELU => FC

```
--- pyimagesearch
|     |--- __init__.py
|     |--- datasets
|     |--- nn
|     |     |--- __init__.py
...
|     |     |--- conv
|     |     |     |--- __init__.py
|     |     |     |--- shallownet.py
|     |--- preprocessing
```

New

We will put the CNN implementations here

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

ShallowNet with Keras and Python

ShallowNet: INPUT => CONV => RELU => FC

shallownet.py

```
# import the necessary packages
from keras.models import Sequential
from keras.layers.convolutional import Conv2D
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers.core import Dense
from keras import backend as K
```

Keras implementation of the convolutional layer

Activation layer

The Flatten classes takes our multi-dimensional volume and "flattens" it into a 1D array prior to feeding the inputs into the Dense (i.e, fully-connected) layers.

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

ShallowNet with Keras and Python

ShallowNet: INPUT => CONV => RELU => FC

`shallownet.py`

Image dimensions

Number of classes

```python
class ShallowNet:
    @staticmethod
    def build(width, height, depth, classes):
        # initialize the model along with the input shape to be
        # "channels last"
        model = Sequential()
        inputShape = (height, width, depth)

        # if we are using "channels first", update the input shape
        if K.image_data_format() == "channels_first":
            inputShape = (depth, height, width)
```

Defensive programming

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

ShallowNet with Keras and Python

ShallowNet: INPUT => CONV => RELU => FC

`shallownet.py`

```
# define the first (and only) CONV => RELU layer
model.add(Conv2D(32, (3, 3), padding="same",
    input_shape=inputShape))
model.add(Activation("relu"))

# softmax classifier
model.add(Flatten())
model.add(Dense(classes))
model.add(Activation("softmax"))

# return the constructed network architecture
return model
```

32 filters, all 3x3, padding ensure size of output = size of input

To apply the fully-connected layer:

1. flatten the multi-dimensional representation into a 1D list

2. Add a dense layer using the same number of nodes as our output class labels

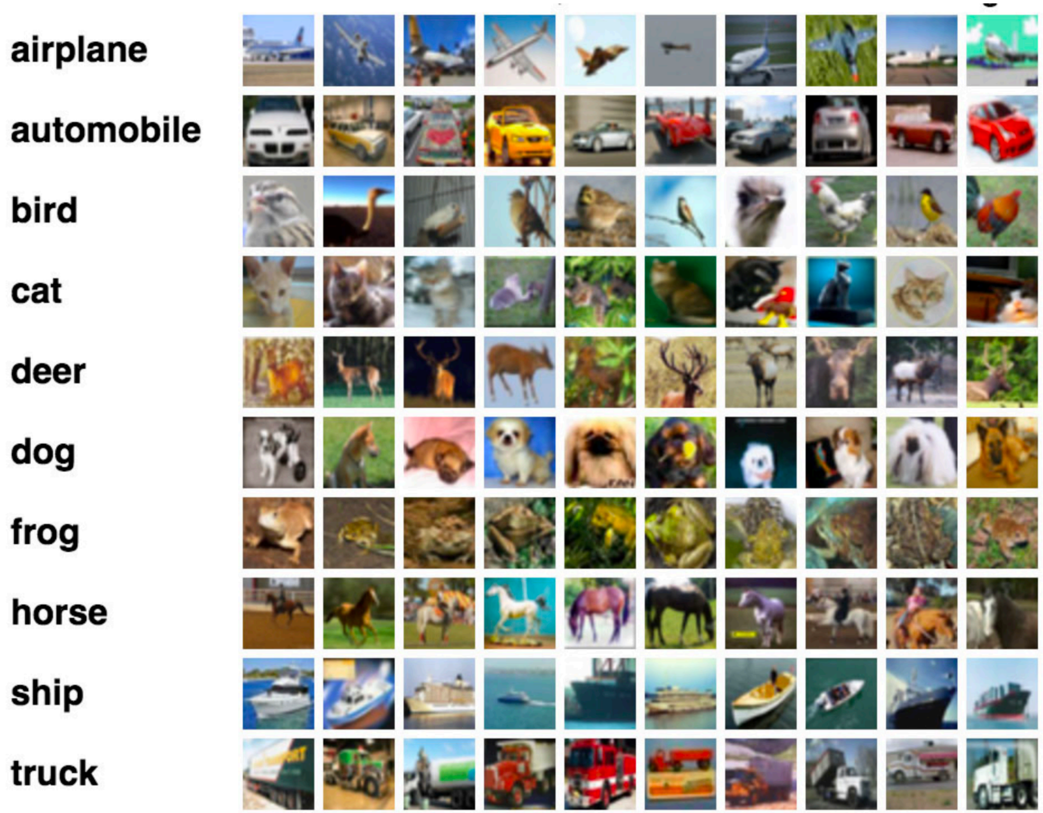
3. Apply a softmax activation function to will generate the class label probabilities for each class

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

ShallowNet with Keras and Python

CIFAR-10

- 60,000 images
- 32 x 32 x 3 (RGB) … 3072 element feature vector
- 10 classes: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks



Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

ShallowNet with Keras and Python

Apply to the CIFAR-10 dataset: shallownet_cifar10.py

Driver script to load a dataset, preprocess it, and then train the network

```
# import the necessary packages
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import classification_report
from pyimagesearch.nn.conv import ShallowNet
from keras.optimizers import SGD
from keras.datasets import cifar10
import matplotlib.pyplot as plt
import numpy as np

# load the training and testing data, then scale it into the
# range [0, 1]
print("[INFO] loading CIFAR-10 data...")
((trainX, trainY), (testX, testY)) = cifar10.load_data()
trainX = trainX.astype("float") / 255.0
testX = testX.astype("float") / 255.0
```

Preprocessing and channel ordering handled automatically in this function. If this is the first time calling cifar10.load_data(), the function will load the dataset for you. The file is ~170Mb so be patient. Once downloaded, it is cached and doesn't need to be downloaded again.

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

ShallowNet with Keras and Python

Apply to the CIFAR-10 dataset: `shallownet_cifar10.py`

```
# convert the labels from integers to vectors
lb = LabelBinarizer()
trainY = lb.fit_transform(trainY)
testY = lb.transform(testY)

# initialize the label names for the CIFAR-10 dataset
labelNames = ["airplane", "automobile", "bird", "cat", "deer",
	"dog", "frog", "horse", "ship", "truck"]
```

Labels are one-hot encoded, i.e. represented as a vector of 0 and 1, where 1 represents the correct class

[1 0 0 0 0 0 0 0 0 0]
[0 1 0 0 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 1]

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

ShallowNet with Keras and Python

Apply to the CIFAR-10 dataset: `shallownet_cifar10.py`

```
# initialize the optimizer and model
print("[INFO] compiling model...")
opt = SGD(lr=0.01)
model = ShallowNet.build(width=32, height=32, depth=3, classes=10)
model.compile(loss="categorical_crossentropy", optimizer=opt,
	metrics=["accuracy"])

# train the network
print("[INFO] training network...")
H = model.fit(trainX, trainY, validation_data=(testX, testY),
	batch_size=32, epochs=40, verbose=1)
```

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

ShallowNet with Keras and Python

Apply to the CIFAR-10 dataset: `shallownet_cifar10.py`

```
# evaluate the network
print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=32)
print(classification_report(testY.argmax(axis=1),
    predictions.argmax(axis=1), target_names=labelNames))
```

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

ShallowNet with Keras and Python

Apply to the CIFAR-10 dataset: `shallownet_cifar10.py`

```
# plot the training loss and accuracy
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 40), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 40), H.history["val_loss"], label="val_loss")

plt.plot(np.arange(0, 40), H.history["acc"], label="train_acc")
plt.plot(np.arange(0, 40), H.history["val_acc"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.show()
```

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

ShallowNet with Keras and Python

To train ShallowNet on CIFAR-10: `$ python shallownet_cifar10.py`

	precision	recall	f1-score	support
airplane	0.62	0.68	0.65	1000
automobile	0.79	0.64	0.71	1000
bird	0.43	0.46	0.44	1000
cat	0.42	0.38	0.40	1000
deer	0.52	0.51	0.52	1000
dog	0.44	0.57	0.50	1000
frog	0.74	0.61	0.67	1000
horse	0.71	0.61	0.66	1000
ship	0.65	0.77	0.70	1000
truck	0.67	0.66	0.66	1000
avg / total	0.60	0.59	0.59	10000

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

ShallowNet with Keras and Python

To train ShallowNet on CIFAR-10: `$ python shallownet_cifar10.py`

	precision	recall	f1-score	support
airplane	0.70	0.54	0.61	1000
automobile	0.68	0.77	0.72	1000
bird	0.53	0.37	0.44	1000
cat	0.36	0.57	0.45	1000
deer	0.60	0.45	0.51	1000
dog	0.54	0.44	0.49	1000
frog	0.64	0.76	0.69	1000
horse	0.64	0.70	0.67	1000
ship	0.75	0.69	0.72	1000
truck	0.67	0.71	0.69	1000
avg / total	0.61	0.60	0.60	10000

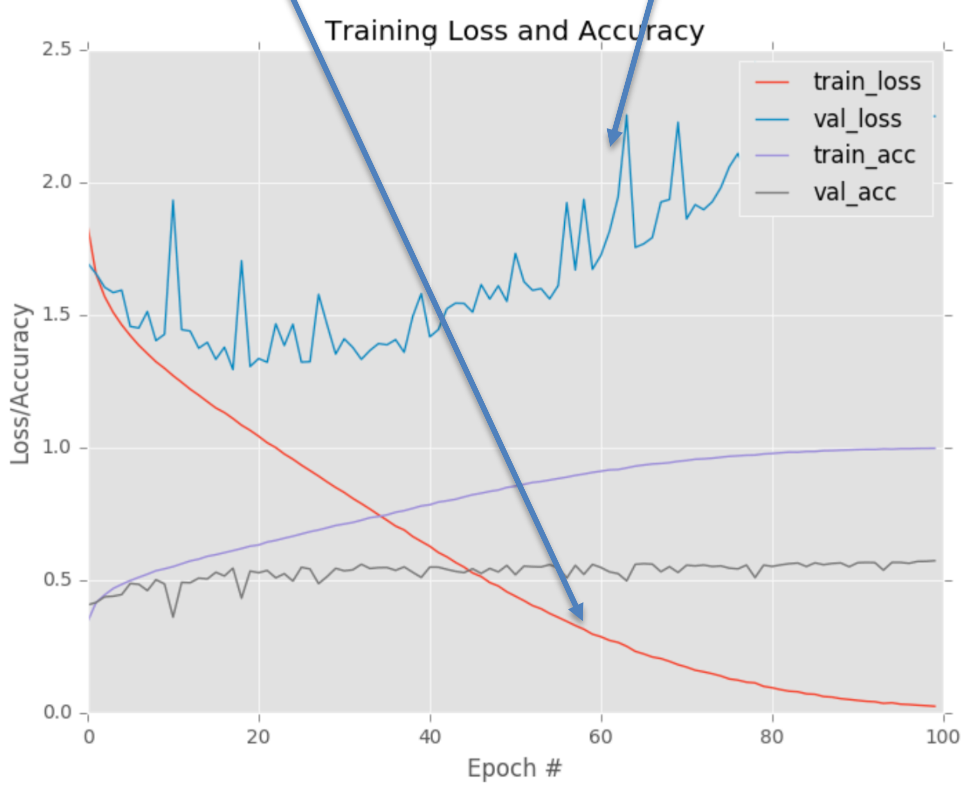
(43 s / epoch on Ubuntu virtual machine)

Recall: 3072-1024-512-10 MLP

CIFAR-10

	precision	recall	f1-score	support
airplane	0.63	0.66	0.64	1000
automobile	0.69	0.65	0.67	1000
bird	0.48	0.43	0.45	1000
cat	0.40	0.38	0.39	1000
deer	0.52	0.51	0.51	1000
dog	0.48	0.47	0.48	1000
frog	0.64	0.63	0.64	1000
horse	0.63	0.62	0.63	1000
ship	0.64	0.74	0.69	1000
truck	0.59	0.65	0.62	1000
avg / total	0.57	0.57	0.57	10000

Dramatic over-fitting:
Training loss falls but validation loss rises



Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

MiniVGGNet

VGGNet (or simply VGG)

- Introduced by Simonyan and Zisserman in 2014: "Very Deep Learning Convolutional Neural Networks for Large-Scale Image Recognition"
- Convolution filters are 3 x 3 throughout the architecture
- Stacking multiple CONV => RELU layer sets
  - more repetitions deeper in the architecture
- 16 & 19 layers
- 2^{nd} in ImageNet classification challenge

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

MiniVGGNet

MiniVGGNet

- INPUT => CONV => ACT => BN => CONV => ACT => BN => POOL => DO =>
  CONV => ACT => BN => CONV => ACT => BN => POOL => DO =>
  FC => ACT => BN => DO => FC => SOFTMAX

- First two CONV layers learn 32 filters

- Second two CONV layer learn 64 filters

- POOL layers perform max pooling over a 2 x 2 window with a 2 x 2 stride

- Input is 32 x 32 x 3 … colour CIFAR-10 images

MiniVGGNet

MiniVGGNet

Layer Type	Output Size	Filter Size / Stride
INPUT IMAGE	$32 \times 32 \times 3$	
CONV	$32 \times 32 \times 32$	$3 \times 3, K = 32$
ACT	$32 \times 32 \times 32$	
BN	$32 \times 32 \times 32$	
CONV	$32 \times 32 \times 32$	$3 \times 3, K = 32$
ACT	$32 \times 32 \times 32$	
BN	$32 \times 32 \times 32$	
POOL	$16 \times 16 \times 32$	2×2
DROPOUT	$16 \times 16 \times 32$	
CONV	$16 \times 16 \times 64$	$3 \times 3, K = 64$
ACT	$16 \times 16 \times 64$	
BN	$16 \times 16 \times 64$	
CONV	$16 \times 16 \times 64$	$3 \times 3, K = 64$
ACT	$16 \times 16 \times 64$	
BN	$16 \times 16 \times 64$	
POOL	$8 \times 8 \times 64$	2×2
DROPOUT	$8 \times 8 \times 64$	
FC	512	
ACT	512	
BN	512	
DROPOUT	512	
FC	10	
SOFTMAX	10	

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

MiniVGGNet

MiniVGGNet

```
--- pyimagesearch
|    |--- __init__.py
|    |--- nn
|    |    |--- __init__.py
...
|    |    |--- conv
|    |    |    |--- __init__.py
|    |    |    |--- lenet.py
|    |    |    |--- minivggnet.py
|    |    |    |--- shallownet.py
```

MiniVGGNet

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

MiniVGGNet

MiniVGGNet **minivggnet.py**

```
# import the necessary packages
from keras.models import Sequential
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers.core import Dropout
from keras.layers.core import Dense
from keras import backend as K
```

Batch Normalization Layer

Pooling Layer

Dropout Layer

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

MiniVGGNet

MiniVGGNet `minivggnet.py`

Batch normalization operates on the features, i.e. channels, so in order to apply BN, we need to know which axis to normalize over (i.e. the channels axis)

Setting chanDim = -1 implies that the index of the channel dimension last in the input shape (i.e., channels last ordering).

```python
class MiniVGGNet:
    @staticmethod
    def build(width, height, depth, classes):
        # initialize the model along with the input shape to be
        # "channels last" and the channels dimension itself
        model = Sequential()
        inputShape = (height, width, depth)
        chanDim = -1

        # if we are using "channels first", update the input shape
        # and channels dimension
        if K.image_data_format() == "channels_first":
            inputShape = (depth, height, width)
            chanDim = 1
```

However, if we are using channels first ordering, we need need to update the inputShape and set chanDim = 1

Create the model

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

MiniVGGNet

MiniVGGNet `minivggnet.py`

32 3x3 filters

```
    # first CONV => RELU => CONV => RELU => POOL layer set
    model.add(Conv2D(32, (3, 3), padding="same",
        input_shape=inputShape))
    model.add(Activation("relu"))
    model.add(BatchNormalization(axis=chanDim))
    model.add(Conv2D(32, (3, 3), padding="same"))
    model.add(Activation("relu"))
    model.add(BatchNormalization(axis=chanDim))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
```

(CONV => RELU => BN) * 2 => POOL => DO

Dropout propability of 0.25

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

MiniVGGNet

MiniVGGNet `minivggnet.py`

64 3x3 filters

```python
# second CONV => RELU => CONV => RELU => POOL layer set
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

Dropout propability of 0.25

(CONV => RELU => BN) * 2 => POOL => DO

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

MiniVGGNet

MiniVGGNet `minivggnet.py`

```
        # first (and only) set of FC => RELU layers
        model.add(Flatten())
        model.add(Dense(512))
        model.add(Activation("relu"))
        model.add(BatchNormalization())
        model.add(Dropout(0.5))

        # softmax classifier
        model.add(Dense(classes))
        model.add(Activation("softmax"))

        # return the constructed network architecture
        return model
```

FC => ACT => BN => DO => FC => SOFTMAX

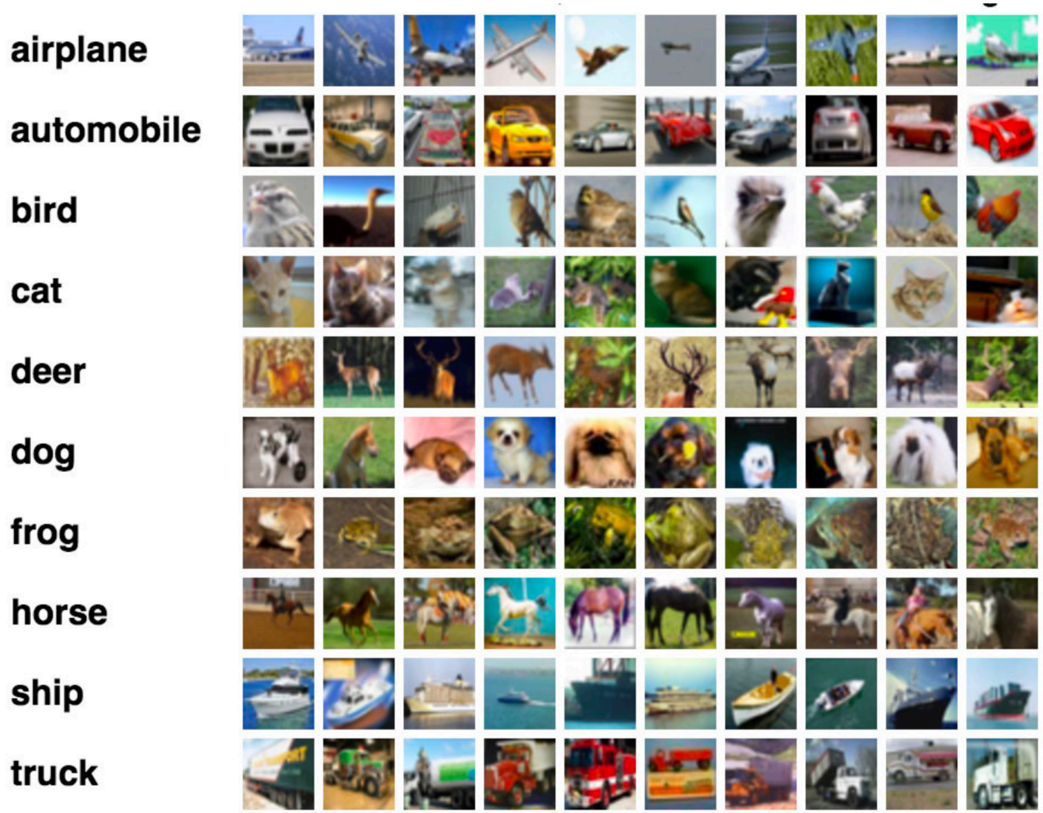
Dropout probability of 0.5

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

MiniVGGNet

CIFAR-10

- 60,000 images
- 32 x 32 x 3 (RGB) … 3072 element feature vector
- 10 classes: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks



Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

MiniVGGNet

MiniVGGNet **minivggnet_cifar10.py** Driver script to load a dataset, preprocess it, and then train the network

```
# set the matplotlib backend so figures can be saved in the background
import matplotlib
matplotlib.use("Agg")

# import the necessary packages
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import classification_report
from pyimagesearch.nn.conv import MiniVGGNet
from keras.optimizers import SGD
from keras.datasets import cifar10
import matplotlib.pyplot as plt
import numpy as np
import argparse
```

Non-interactive: save plot to file

Parse command line arguments

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

MiniVGGNet

MiniVGGNet `minivggnet_cifar10.py`

```
# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-o", "--output", required=True,
    help="path to the output loss/accuracy plot")
args = vars(ap.parse_args())
```

Add a single argument: the path and filename of the file to which the output training and loss plot is saved

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

MiniVGGNet

MiniVGGNet `minivggnet_cifar10.py`

```
# load the training and testing data, then scale it into the
# range [0, 1]
print("[INFO] loading CIFAR-10 data...")
((trainX, trainY), (testX, testY)) = cifar10.load_data()
trainX = trainX.astype("float") / 255.0
testX = testX.astype("float") / 255.0

# convert the labels from integers to vectors
lb = LabelBinarizer()
trainY = lb.fit_transform(trainY)
testY = lb.transform(testY)

# initialize the label names for the CIFAR-10 dataset
labelNames = ["airplane", "automobile", "bird", "cat", "deer",
     "dog", "frog", "horse", "ship", "truck"]
```

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

MiniVGGNet

MiniVGGNet `minivggnet_cifar10.py`

Set learning rate, momentum, and acceleration for stochastic gradient descent. The decay argument causes the learning rate to reduce with time: typically set to learning rate / # epochs

```
# initialize the optimizer and model
print("[INFO] compiling model...")
opt = SGD(lr=0.01, decay=0.01 / 40, momentum=0.9, nesterov=True)
model = MiniVGGNet.build(width=32, height=32, depth=3, classes=10)
model.compile(loss="categorical_crossentropy", optimizer=opt,
     metrics=["accuracy"])

# train the network
print("[INFO] training network...")
H = model.fit(trainX, trainY, validation_data=(testX, testY),
     batch_size=64, epochs=40, verbose=1)
```

Set loss function, optimizer (defined above), and metric

Train the model over 40 epochs with batch size of 64

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

MiniVGGNet

MiniVGGNet `minivggnet.py`

```
# evaluate the network
print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=64)
print(classification_report(testY.argmax(axis=1),
    predictions.argmax(axis=1), target_names=labelNames))
```

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

MiniVGGNet

MiniVGGNet `minivggnet.py`

```
# plot the training loss and accuracy
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 40), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 40), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 40), H.history["acc"], label="train_acc")
plt.plot(np.arange(0, 40), H.history["val_acc"], label="val_acc")
plt.title("Training Loss and Accuracy on CIFAR-10")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.savefig(args["output"])
```

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

MiniVGGNet

MiniVGGNet

```
$ python minivggnet_cifar10.py --output output/cifar10_minivggnet_with_bn.png
```

	precision	recall	f1-score	support
airplane	0.88	0.81	0.85	1000
automobile	0.93	0.89	0.91	1000
bird	0.83	0.68	0.75	1000
cat	0.69	0.65	0.67	1000
deer	0.74	0.85	0.79	1000
dog	0.72	0.77	0.74	1000
frog	0.85	0.89	0.87	1000
horse	0.85	0.87	0.86	1000
ship	0.89	0.91	0.90	1000
truck	0.88	0.91	0.90	1000
avg / total	0.83	0.82	0.82	10000

Drops to 0.79 if we leave out batch normalization

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Pre-trained CNNs in Keras

The Keras library ships with five CNNs that have been pre-trained on the ImageNet dataset:

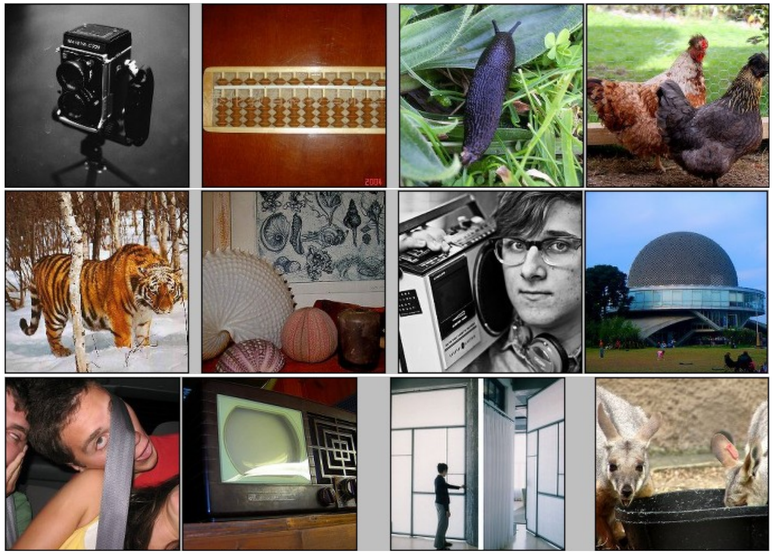
1. VGG16
2. VGG19
3. ResNet50
4. Inception V3
5. Xception

The goal of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is to train a model that can correctly classify an input image into 1000 separate object categories

Pre-trained CNNs in Keras

ImageNet Object Recognition Challenge:

1.2 million training images, 1000 classes



[Deng et al. CVPR 2009]

Credit: Toby Breckon, Durham University

Pre-trained CNNs in Keras

The Keras library ships with five CNNs that have been pre-trained on the ImageNet dataset:

1. VGG16
2. VGG19
3. ResNet50
4. Inception V3
5. Xception

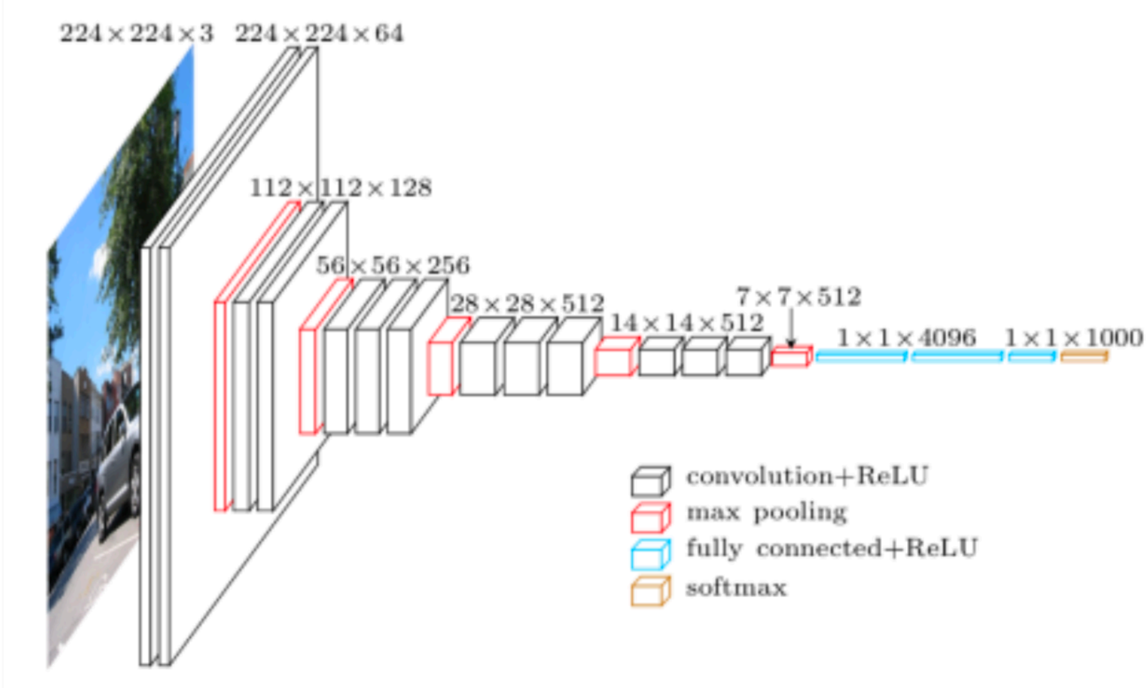
The goal of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is to train a model that can correctly classify an input image into 1000 separate object categories

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Pre-trained CNNs in Keras

VGG -16

K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large- Scale Image Recognition” , arXiv technical report, 2014.



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

Pre-trained CNNs in Keras

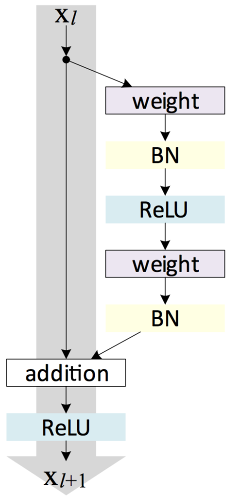
VGG16 / VGG19

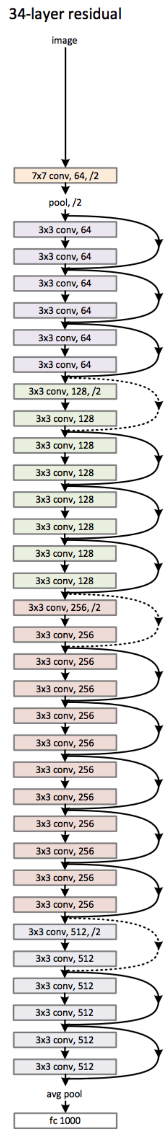
- Only 3 x 3 convolutional layers
- Volume size is reduced by max pooling
- Two fully-connected layers each with 4096 nodes
- Softmax classifier
- Very slow to train
- Network weights file size: 533 MB (VGG16) and 574 MB (VGG19)

Pre-trained CNNs in Keras

ResNet50

K. He et al. “Deep Residual Learning for Image Recognition”, arXiv technical report, 2015.





Pre-trained CNNs in Keras

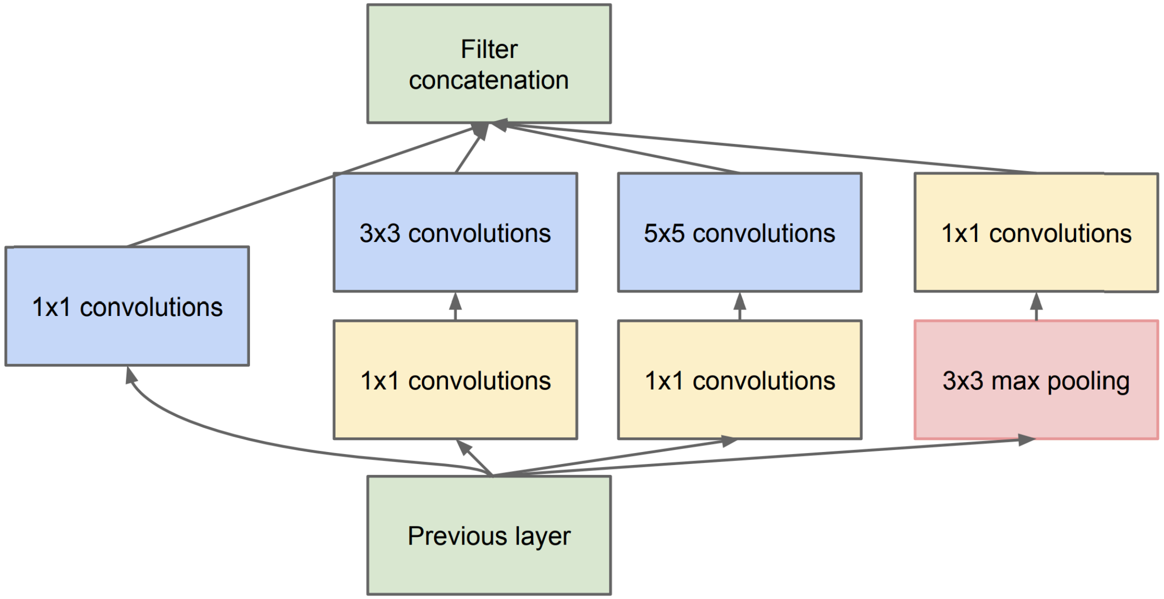
ResNet

- ResNet50 in Keras
- Can be much deeper: up to 200 for ImageNet and 1000 for CIFAR-1-
- Network weights file size: 102 MB for ResNet50

Pre-trained CNNs in Keras

Inception V3

C. Szegedy et al. "Going Deeper with Convolutions". In: *Computer Vision and Pattern Recognition*, 2015.



Inception Module

Credit: https://arxiv.org/abs/1409.4842

Pre-trained CNNs in Keras

Inception V3

- The inception module is a multi-level feature extractor
- Output of each module is stacked along the channel dimension before being fed into the next layer
- Originally called GoogLeNet
- Subsequently called Inception vN, where N denotes the version number
- Keras version is V3 from C. Szegedy et al. “Rethinking the Inception Architecture for Computer Vision”, 2015
- Network weights file size: 96 MB

Pre-trained CNNs in Keras

Xception

- François Chollet, creator and chief maintainer of the Keras Library
- François Chollet. “Xception: Deep Learning with Depthwise Separable Convolutions”
- Network weights file size: 91 MB

Pre-trained CNNs in Keras

imagenet_pretrained.py

```python
# import the necessary packages
from keras.applications import ResNet50
from keras.applications import InceptionV3
from keras.applications import Xception # TensorFlow ONLY
from keras.applications import VGG16
from keras.applications import VGG19
from keras.applications import imagenet_utils
from keras.applications.inception_v3 import preprocess_input
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import load_img
import numpy as np
import argparse
import cv2
```

Keras implementations

Convenience utility functions

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Pre-trained CNNs in Keras

`imagenet_pretrained.py`

```
# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
	help="path to the input image")
ap.add_argument("-model", "--model", type=str, default="vgg16",
	help="name of pre-trained network to use")
args = vars(ap.parse_args())
```

Add a two arguments:
the path and filename of the file to use as input
and the model to use

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Pre-trained CNNs in Keras

`imagenet_pretrained.py`

```
# define a dictionary that maps model names to their classes
# inside Keras
MODELS = {
	"vgg16": VGG16,
	"vgg19": VGG19,
	"inception": InceptionV3,
	"xception": Xception,  # TensorFlow ONLY
	"resnet": ResNet50
}

# ensure a valid model name was supplied via command line argument
if args["model"] not in MODELS.keys():
	raise AssertionError("The --model command line argument should "
		"be a key in the ‘MODELS‘ dictionary")
```

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Pre-trained CNNs in Keras

`imagenet_pretrained.py`

```python
# initialize the input image shape (224x224 pixels) along with
# the pre-processing function (this might need to be changed
# based on which model we use to classify our image)
inputShape = (224, 224)
preprocess = imagenet_utils.preprocess_input

# if we are using the InceptionV3 or Xception networks, then we
# need to set the input shape to (299x299) [rather than (224x224)]
# and use a different image processing function
if args["model"] in ("inception", "xception"):
    inputShape = (299, 299)
    preprocess = preprocess_input
```

VGG16, VGG19, and ResNet use 224x224 images

But InceptionV3 and Xception use 229x229

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Pre-trained CNNs in Keras

`imagenet_pretrained.py`

```
# load our the network weights from disk (NOTE: if this is the
# first time you are running this script for a given network, the
# weights will need to be downloaded first -- depending on which
# network you are using, the weights can be 90-575MB, so be
# patient; the weights will be cached and subsequent runs of this
# script will be *much* faster)
print("[INFO] loading {}...".format(args["model"]))
Network = MODELS[args["model"]]
model = Network(weights="imagenet")
```

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Pre-trained CNNs in Keras

`imagenet_pretrained.py`

```
# load the input image using the Keras helper utility while ensuring
# the image is resized to ‘inputShape‘, the required input dimensions
# for the ImageNet pre-trained network
print("[INFO] loading and pre-processing image...")
image = load_img(args["image"], target_size=inputShape)
image = img_to_array(image)

# our input image is now represented as a NumPy array of shape
# (inputShape[0], inputShape[1], 3) however we need to expand the
# dimension by making the shape (1, inputShape[0], inputShape[1], 3)
# so we can pass it through thenetwork
image = np.expand_dims(image, axis=0)

# pre-process the image using the appropriate function based on the
# model that has been loaded (i.e., mean subtraction, scaling, etc.)
image = preprocess(image)
```

Images are trained/classified in batches with these CNNs so we need to add an extra dimension; failure to do so will cause an error

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Pre-trained CNNs in Keras

`imagenet_pretrained.py`

```
# classify the image
print("[INFO] classifying image with '{}'...".format(args["model"]))
preds = model.predict(image)
P = imagenet_utils.decode_predictions(preds)

# loop over the predictions and display the rank-5 predictions +
# probabilities to our terminal
for (i, (imagenetID, label, prob)) in enumerate(P[0]):
    print("{}. {}: {:.2f}%".format(i + 1, label, prob * 100))
```

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Pre-trained CNNs in Keras

`imagenet_pretrained.py`

```python
# load the image via OpenCV, draw the top prediction on the image,
# and display the image to our screen
orig = cv2.imread(args["image"])
(imagenetID, label, prob) = P[0][0]
cv2.putText(orig, "Label: {}".format(label), (10, 30),
    cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0), 2)
cv2.imshow("Classification", orig)
cv2.waitKey(0)
```

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Pre-trained CNNs in Keras

`imagenet_pretrained.py`

```
$ python imagenet_pretrained.py \
    --image example_images/example_01.jpg --model vgg16
$ python imagenet_pretrained.py \
    --image example_images/example_02.jpg --model vgg19
$ python imagenet_pretrained.py \
    --image example_images/example_03.jpg --model inception
$ python imagenet_pretrained.py \
    --image example_images/example_04.jpg --model xception
$ python imagenet_pretrained.py \
    --image example_images/example_05.jpg --model resnet
```

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017

Pre-trained CNNs in Keras

`imagenet_pretrained.py`

Credit: Adrian Rosebrock, *Deep Learning for Computer Vision*, PyImageSearch, 2017