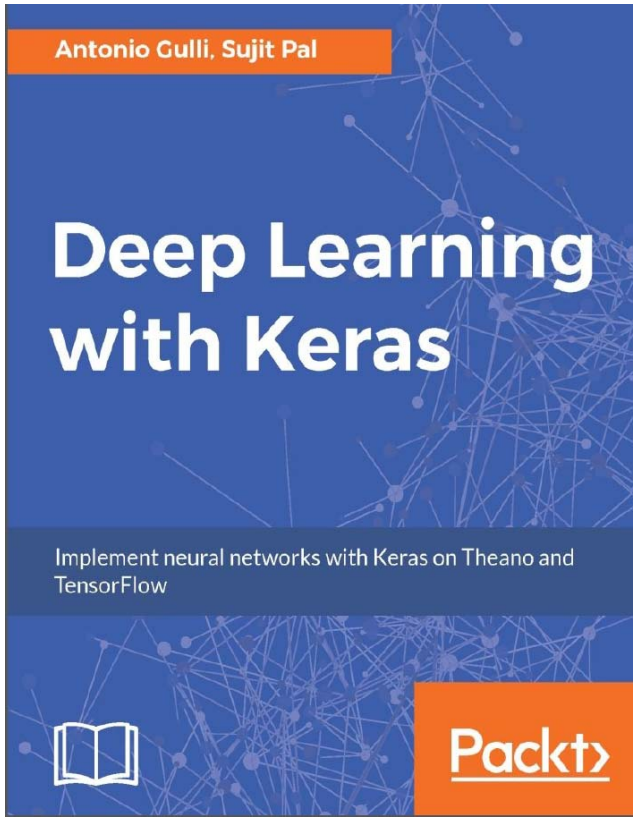

GULF COAST
CODE CAMP
SPONSORS
SSI
TEKsystems
Our people make IT possible.
BW
BIT-WIZARDS
GATEWAY SOURCING
SOFTWARE ARCHITECTS
Dorger
PILOT
INFRAGISTICS
MIGHTY
sparkhound
SPRING HILL
COLLEGE
ApexSQL
ASPOSE
Your File Format APIs
devart
DevExpress
THANK YOU FOR YOUR SUPPORT
GrapeCity
CODESMITH
Tools

Starting Deep Learning with Python

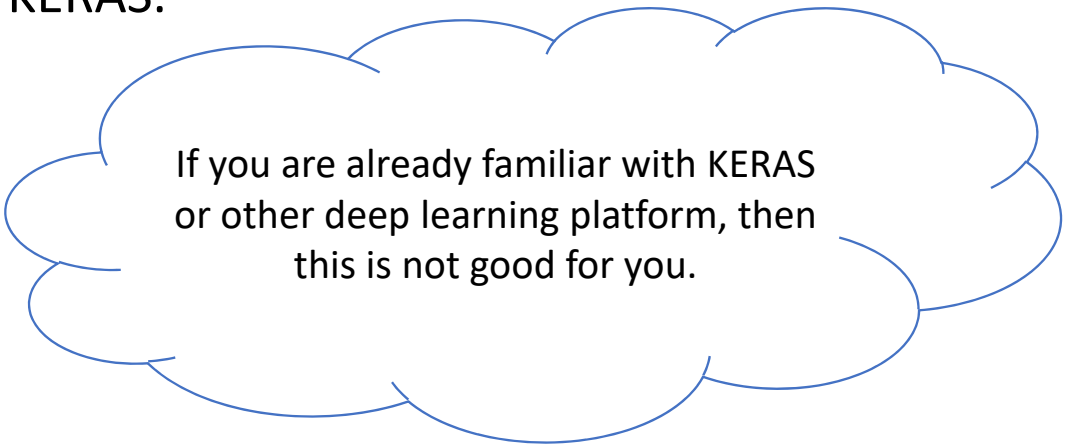
– Keras –

Ingyu Lee
Troy University



Contents

- What is this talk about?
  - Introduce Deep Learning with KERAS.
- Who needs this talk?
  - Who are interested in Deep Learning and about to start develop one.
- What will be covered?
  - How to start with deep learning with KERAS.
- What will not be covered?
  - Detail applications on deep learning.
  - Theory on deep learning.



What is Deep Learning?

Deep Learning: machine learning algorithms based on learning multiple levels of representation / abstraction.

Amazing improvements in error rate in object recognition, object detection, speech recognition, and more recently, in natural language processing / understanding

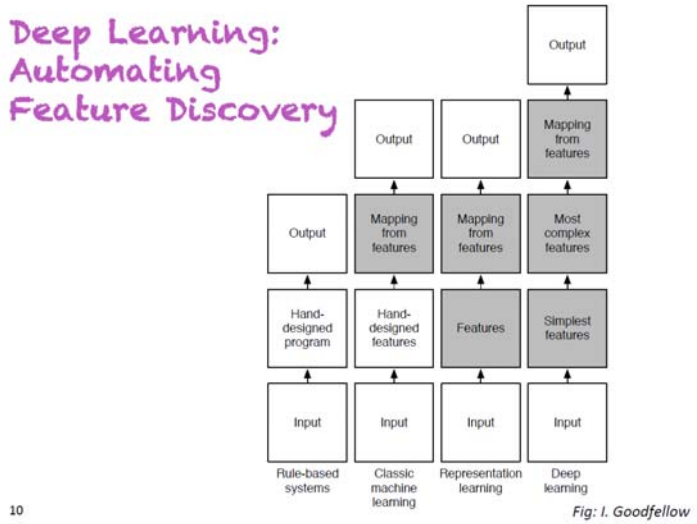
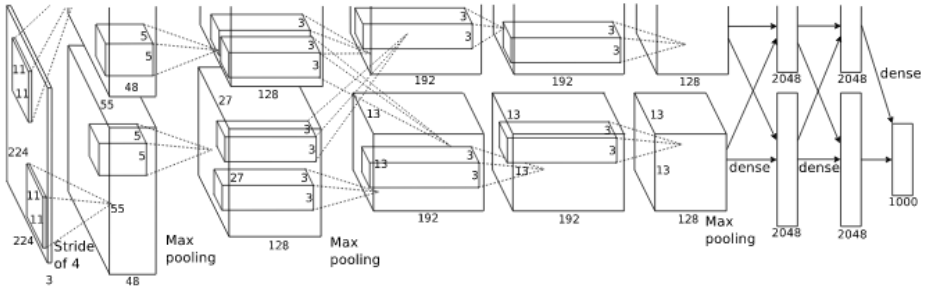


Fig: I. Goodfellow

Deep Learning: Applications

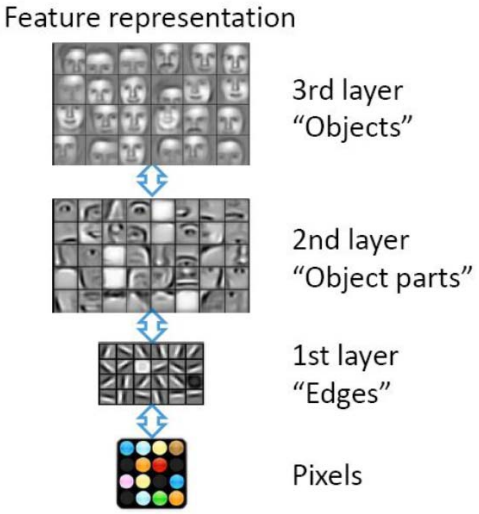


https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Why Deep Learning?

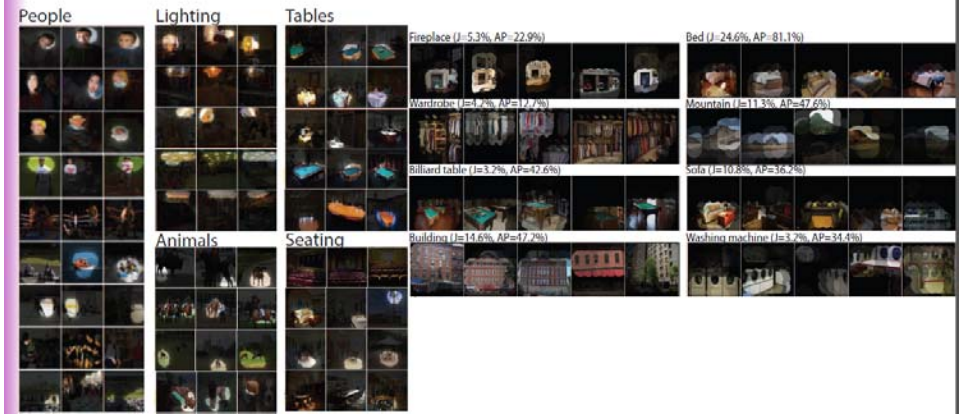
Different Levels of Abstraction

- Hierarchical Learning
  - Natural progression from low level to high level structure as seen in natural complexity
  - Easier to monitor what is being learnt and to guide the machine to better subspaces
  - A good lower level representation can be used for many distinct tasks

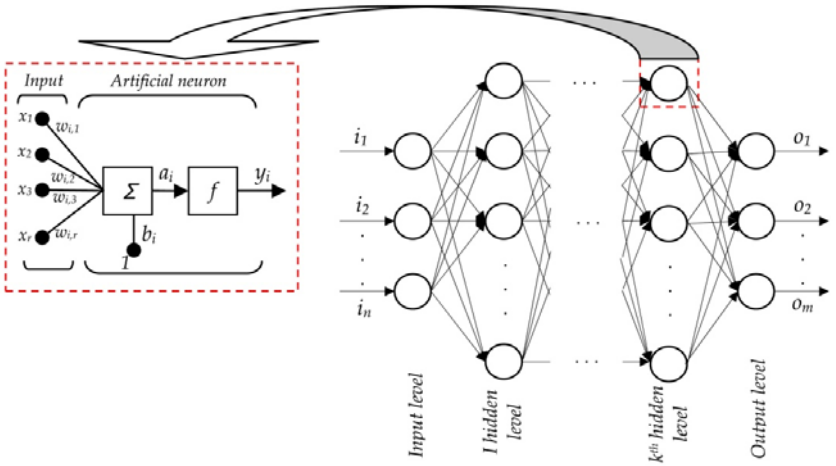


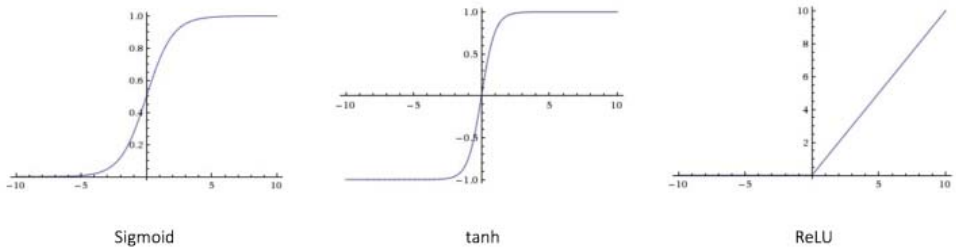
Hidden Units Discover Semantically Meaningful Concepts

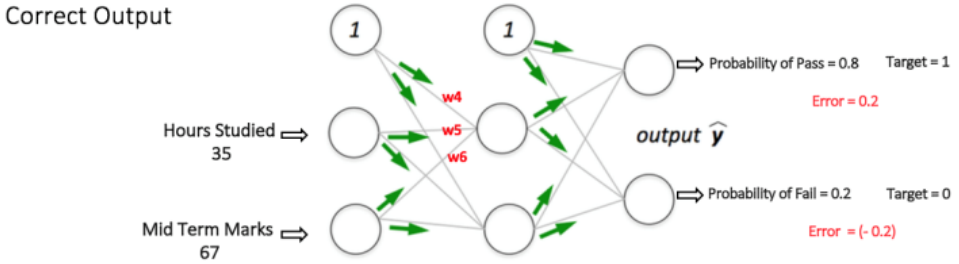
- *Zhou et al & Torralba, arXiv1412.6856* submitted to ICLR 2015
- Network trained to recognize places, not objects

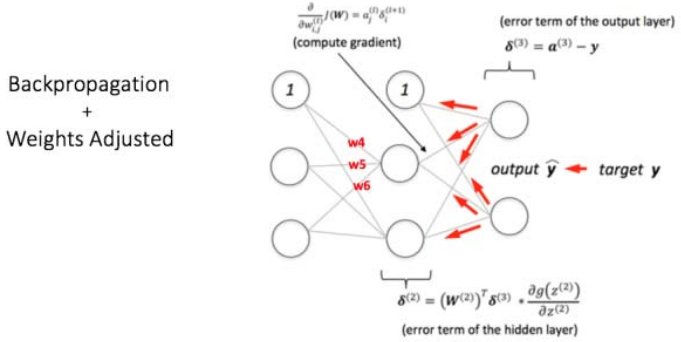


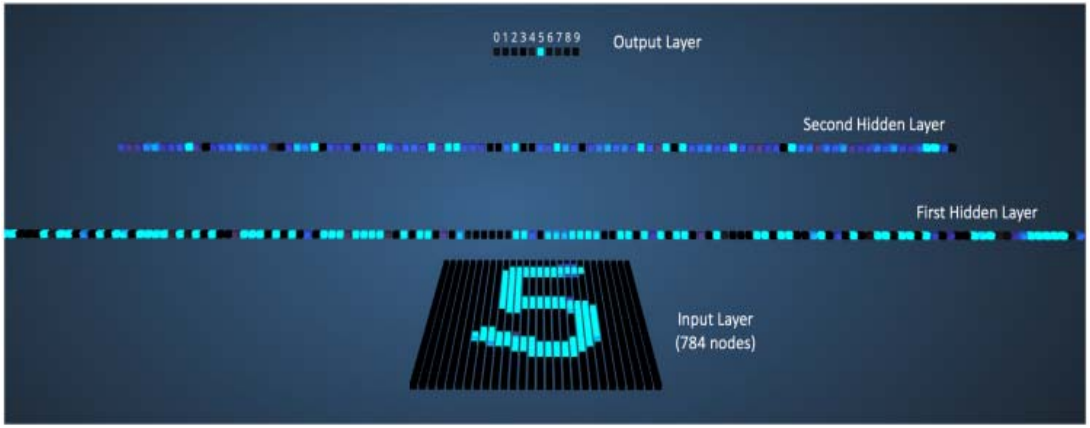
Background


Input
Artificial neuron
Input level
l hidden level
k^th hidden level
Output level


Sigmoid
tanh
ReLU


Correct Output
Hours Studied
35
Mid Term Marks
67
w4
w5
w6
output ŷ
Probability of Pass = 0.8
Target = 1
Error = 0.2
Probability of Fail = 0.2
Target = 0
Error = (- 0.2)


Backpropagation
+
Weights Adjusted
(compute gradient)
(error term of the output layer)
δ(3) = a(3) − y
w4
w5
w6
output ŷ
target y
(error term of the hidden layer)


0123456789
Output Layer
Second Hidden Layer
First Hidden Layer
Input Layer
(784 nodes)

Deep Learning Frameworks

	Languages	Tutorials and training materials	CNN modeling capability	RNN modeling capability	Architecture: easy-to-use and modular front end	Speed	Multiple GPU support	Keras compatible
Theano	Python, C++	++	++	++	+	++	+	+
Tensor-Flow	Python	+++	+++	++	+++	++	++	+
Torch	Lua, Python (new)	+	+++	++	++	+++	++	
Caffe	C++	+	++		+	+	+	
MXNet	R, Python, Julia, Scala	++	++	+	++	++	+++	
Neon	Python	+	++	+	+	++	+	
CNTK	C++	+	+	+++	+	++	+	

Keras

What is Keras?

- Neural Network library written in Python
- Designed to be minimalistic & straight forward yet extensive
- Built on top of either Theano as newly TensorFlow

Why use Keras?

- Simple to get started, simple to keep going
- Written in python and highly modular; easy to expand
- Deep enough to build serious models

General idea is to based on layers and their input/output

- Prepare your inputs and output tensors
- Create first layer to handle input tensor
- Create output layer to handle targets
- Build virtually any model you like in between

Keras

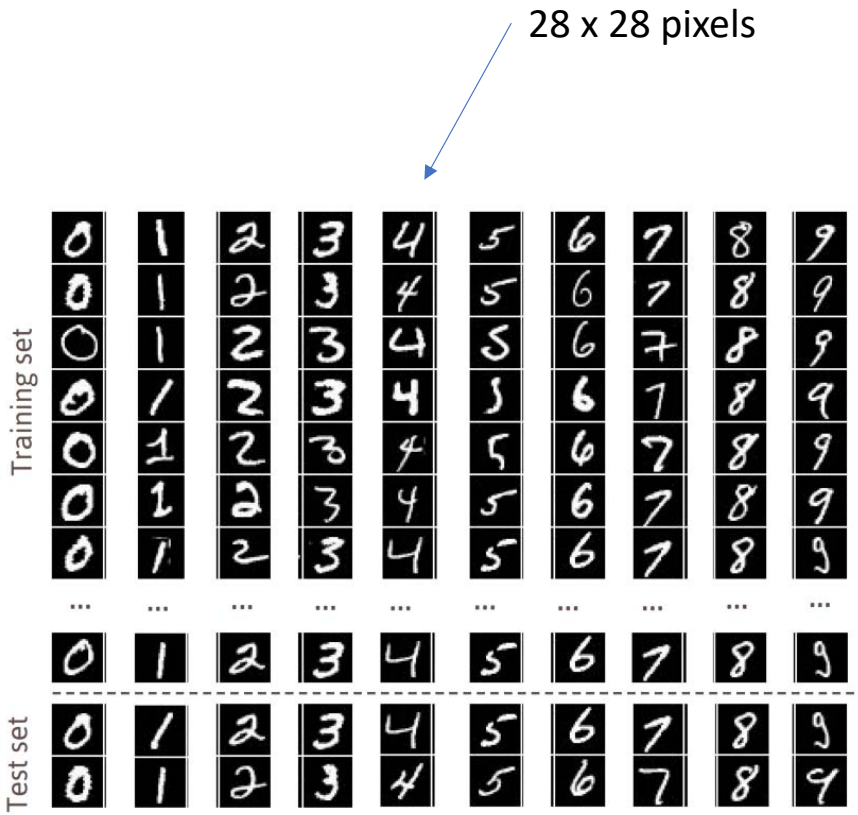
Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

First Applications

- We will build a neural network that can recognize handwritten numbers.
- We use MNIST (a database of handwritten digits made up of a training set of 60,000 examples and a test set of 10,000 examples.
- The examples are annotated by humans with the correct answer. Since a database with correct answers is available, we can perform a supervised learning.



Step 1: Set up your environment

- Python 2.7+
- SciPy and NumPy
- Matplotlib
- Theano
- Tensorflow

```
sudo apt-get update
sudo apt-get install python-pip python-dev build-essential
sudo pip install --upgrade pip
sudo pip install ipython
sudo apt-get install libopenblas-dev liblapack-dev gfortran python-numpy python-scipy
sudo pip install matplotlib scikit-learn scikit-image protobuf jupyter
sudo pip install tensorflow
sudo pip install Theano
sudo pip install keras
```

Step 2: Import the libraries and dataset

```
from __future__ import print_function
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD
from keras.utils import np_utils
np.random.seed(1671) # for reproducibility
```

```
# data: shuffled and split between train and test sets
#
(X_train, y_train), (X_test, y_test) = mnist.load_data()
#X_train is 60000 rows of 28x28 values --> reshaped in 60000 x 784
RESHAPED = 784
#
X_train = X_train.reshape(60000, RESHAPED)
X_test = X_test.reshape(10000, RESHAPED)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
# normalize
#
X_train /= 255
X_test /= 255
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
```

```
# network and training
NB_EPOCH = 200
BATCH_SIZE = 128
VERBOSE = 1
NB_CLASSES = 10 # number of outputs = number of digits
OPTIMIZER = SGD() # SGD optimizer, explained later in this chapter
N_HIDDEN = 128
VALIDATION_SPLIT=0.2 # how much TRAIN is reserved for VALIDATION
```

```
class vectors to binary class matrices
np_utils.to_categorical(y_train, NB_CLASSES)
np_utils.to_categorical(y_test, NB_CLASSES)
```

Step 3: Define networks and run the model

```
# final stage is softmax
model = Sequential()
model.add(Dense(NB_CLASSES, input_shape=(RESHAPED,)))
model.add(Activation('softmax'))
model.summary()
```

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

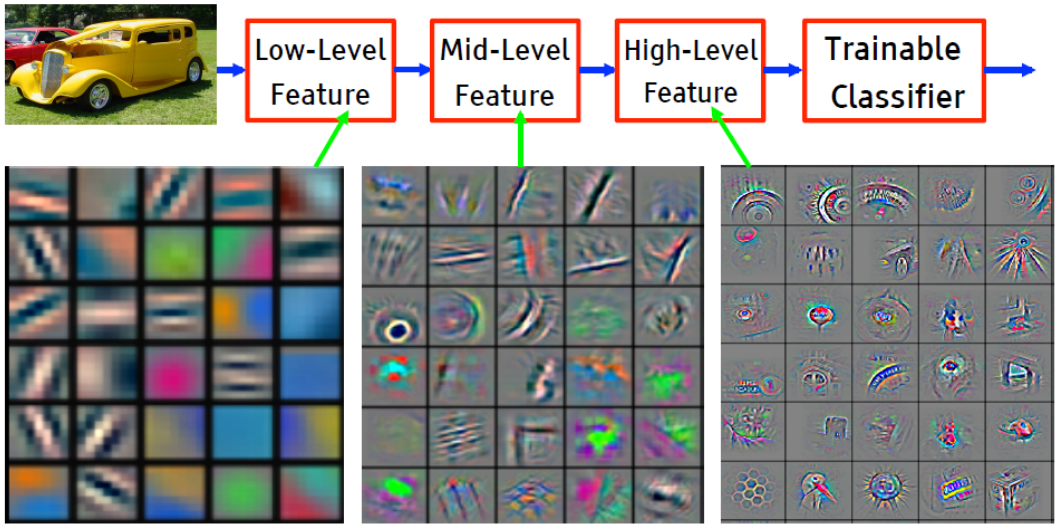
```
history = model.fit(X_train, Y_train,
batch_size=BATCH_SIZE, epochs=NB_EPOCH,
verbose=VERBOSE, validation_split=VALIDATION_SPLIT)
```

```
score = model.evaluate(X_test, Y_test, verbose=VERBOSE)
print("Test score:", score[0])
print('Test accuracy:', score[1])
```

```
Using TensorFlow backend.
_________________________________________________________________
Layer (type)                 Output Shape              Param #   Connected to
=================================================================
dense_1 (Dense)              (None, 10)                7850      dense_input_1[0][0]
_________________________________________________________________
activation_1 (Activation)    (None, 10)                0         dense_1[0][0]
=================================================================
Total params: 7850
_________________________________________________________________
Train on 48000 samples, validate on 12000 samples
Epoch 1/200
48000/48000 [==============================] - 0s - loss: 1.4102 - acc: 0.6554 - val_loss: 0.9073 - val_acc: 0.8244
Epoch 2/200
48000/48000 [==============================] - 0s - loss: 0.8006 - acc: 0.8279 - val_loss: 0.6625 - val_acc: 0.8567
Epoch 3/200
48000/48000 [==============================] - 0s - loss: 0.6467 - acc: 0.8495 - val_loss: 0.5650 - val_acc: 0.8704
Epoch 4/200
48000/48000 [==============================] - 0s - loss: 0.5728 - acc: 0.8600 - val_loss: 0.5112 - val_acc: 0.8778
Epoch 5/200
48000/48000 [==============================] - 0s - loss: 0.5280 - acc: 0.8677 - val_loss: 0.4767 - val_acc: 0.8822
```

Why Multiple Layers?
The World is Compositional

- Hierarchy of representations with increasing level of abstraction
- Each stage is a kind of trainable feature transform
- Image recognition: Pixel → edge → texton → motif → part → object
- Text: Character → word → word group → clause → sentence → story
- Speech: Sample → spectral band → sound → … → phone → phoneme → word



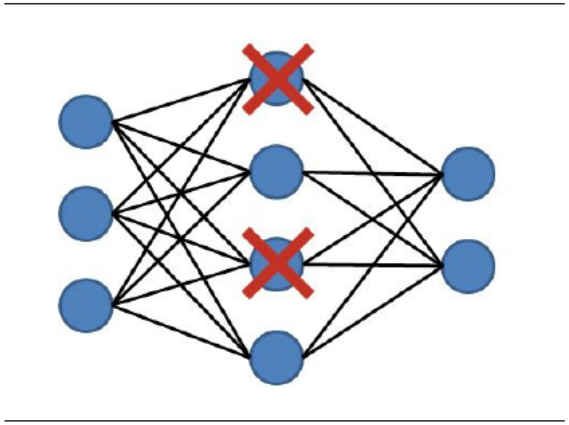
Improve the model with **hidden layers**

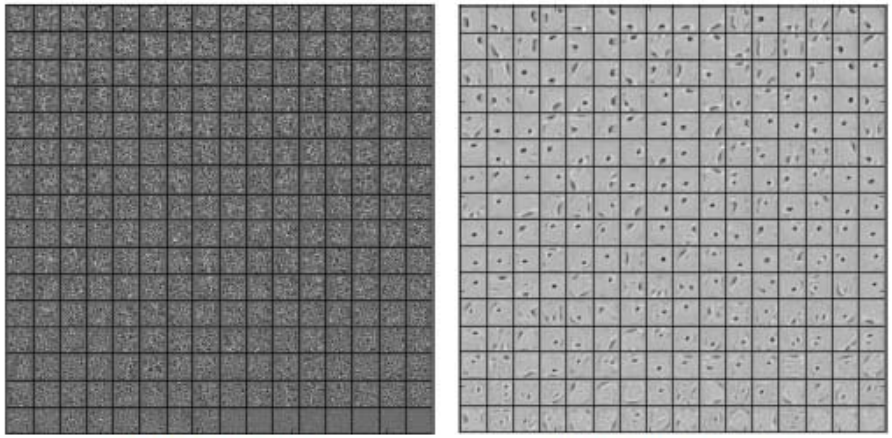
```
from __future__ import print_function
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD
from keras.utils import np_utils
np.random.seed(1671) # for reproducibility
# network and training
NB_EPOCH = 20
BATCH_SIZE = 128
VERBOSE = 1
NB_CLASSES = 10 # number of outputs = number of digits
OPTIMIZER = SGD() # optimizer, explained later in this chapter
N_HIDDEN = 128
VALIDATION_SPLIT=0.2 # how much TRAIN is reserved for VALIDATION
# data: shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
#X_train is 60000 rows of 28x28 values --> reshaped in 60000 x 784
RESHAPED = 784
#
X_train = X_train.reshape(60000, RESHAPED)
X_test = X_test.reshape(10000, RESHAPED)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
# normalize
X_train /= 255
X_test /= 255
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
# convert class vectors to binary class matrices
Y_train = np_utils.to_categorical(y_train, NB_CLASSES)
Y_test = np_utils.to_categorical(y_test, NB_CLASSES)
```

```
# M_HIDDEN hidden layers
# 10 outputs
# final stage is softmax
model = Sequential()
model.add(Dense(N_HIDDEN, input_shape=(RESHAPED,)))
model.add(Activation('relu'))
model.add(Dense(N_HIDDEN))
model.add(Activation('relu'))
model.add(Dense(NB_CLASSES))
model.add(Activation('softmax'))
model.summary()
model.compile(loss='categorical_crossentropy',
optimizer=OPTIMIZER,
metrics=['accuracy'])
history = model.fit(X_train, Y_train,
batch_size=BATCH_SIZE, epochs=NB_EPOCH,
verbose=VERBOSE, validation_split=VALIDATION_SPLIT)
score = model.evaluate(X_test, Y_test, verbose=VERBOSE)
print("Test score:", score[0])
```

What is dropout?

During the learning phase, the connections with the next layer can be limited to a subset of neurons to reduce the weights to be updated, this learning optimization technique is called **dropout**. The dropout is therefore a technique used to decrease the overfitting within a network with many layers and/or neurons.





(a) Without dropout

(b) Dropout with $p = 0.5$.

Further improvement with **dropout**

```
from __future__ import print_function
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.optimizers import SGD
from keras.utils import np_utils
np.random.seed(1671) # for reproducibility
# network and training
NB_EPOCH = 250
BATCH_SIZE = 128
VERBOSE = 1
NB_CLASSES = 10 # number of outputs = number of digits
OPTIMIZER = SGD() # optimizer, explained later in this chapter
N_HIDDEN = 128
VALIDATION_SPLIT=0.2 # how much TRAIN is reserved for VALIDATION
DROPOUT = 0.3
# data: shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
#X_train is 60000 rows of 28x28 values --> reshaped in 60000 x 784
RESHAPED = 784
#
X_train = X_train.reshape(60000, RESHAPED)
X_test = X_test.reshape(10000, RESHAPED)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
# normalize
X_train /= 255
X_test /= 255
# convert class vectors to binary class matrices
Y_train = np_utils.to_categorical(y_train, NB_CLASSES)
Y_test = np_utils.to_categorical(y_test, NB_CLASSES)
```

```
# M_HIDDEN hidden layers 10 outputs
model = Sequential()
model.add(Dense(N_HIDDEN, input_shape=(RESHAPED,)))
model.add(Activation('relu'))
model.add(Dropout(DROPOUT))
model.add(Dense(N_HIDDEN))
model.add(Activation('relu'))
model.add(Dropout(DROPOUT))
model.add(Dense(NB_CLASSES))
model.add(Activation('softmax'))
model.summary()
model.compile(loss='categorical_crossentropy',
optimizer=OPTIMIZER,
metrics=['accuracy'])
history = model.fit(X_train, Y_train,
batch_size=BATCH_SIZE, epochs=NB_EPOCH,
verbose=VERBOSE, validation_split=VALIDATION_SPLIT)
score = model.evaluate(X_test, Y_test, verbose=VERBOSE)
print("Test score:", score[0])
print('Test accuracy:', score[1])
```

Other Improvements

- Use different optimizers in Keras
  - Stochastic gradient descent (SGD)
  - RMSprop and Adam

```
from keras.optimizers import RMSprop, Adam
...
OPTIMIZER = RMSprop() # optimizer,
```

```
OPTIMIZER = Adam() # optimizer
```

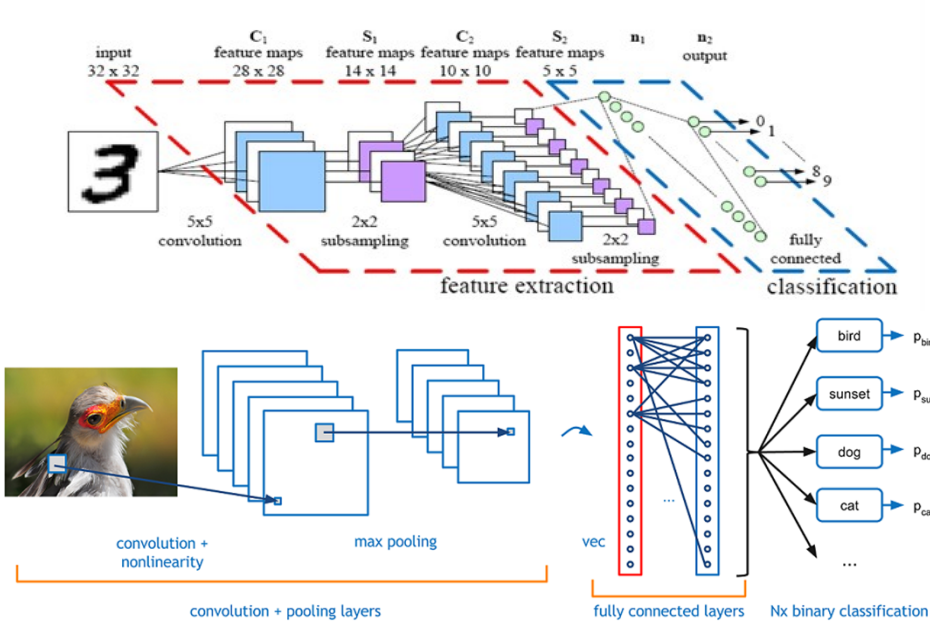
Model/Accuracy	Training	Validation	Test
Simple	92.36%	92.37%	92.22%
Two hidden (128)	94.50%	94.63%	94.41%
Dropout (30%)	98.10%	97.73%	97.7% (200 epochs)
RMSprop	97.97%	97.59%	97.84% (20 epochs)
Adam	98.28%	98.03%	97.93% (20 epochs)

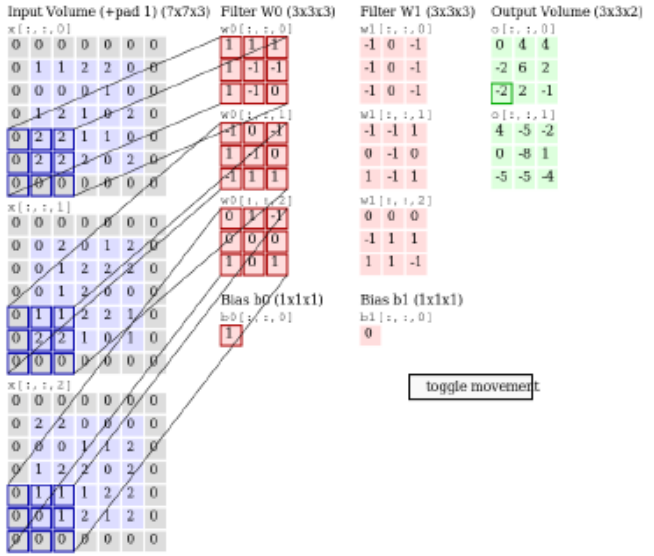
- Increasing the number of epochs.
- Controlling the optimizer learning rate.
- Increasing the number of internal hidden neurons.
- Increasing the size of batch computation.
- Adopting regularization for avoiding overfitting.

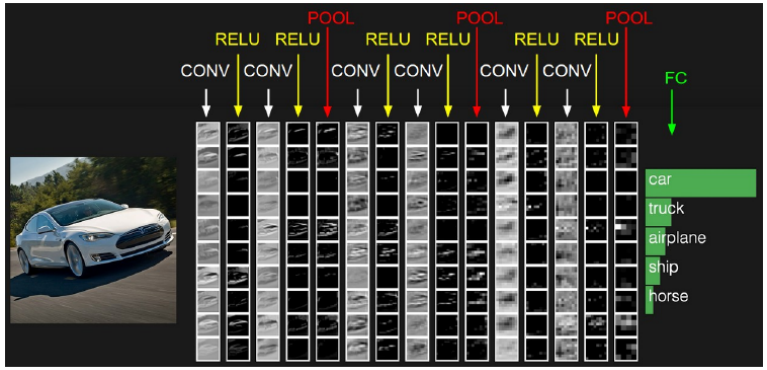
```
from keras import regularizers model.add(Dense(64, input_dim=64, kernel_regularizer=regularizers.l2(0.01)))
```

Convolutional Neural Network (CNN)

- Key intuitions beyond ConvNets
  - Local receptive fields
  - Shared weights
  - Pooling



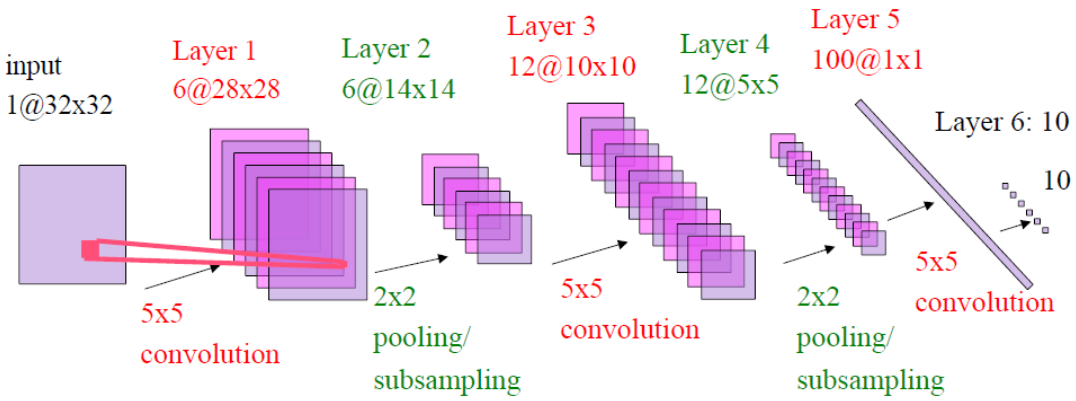




Example of CNN: LeNet

```
from keras import backend as K
from keras.models import Sequential
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 Dense
from keras.datasets import mnist
from keras.utils import np_utils
from keras.optimizers import SGD, RMSprop, Adam
import numpy as np
import matplotlib.pyplot as plt
```

```
#define the ConvNet
class LeNet:
   @staticmethod
   def build(input_shape, classes):
        model = Sequential()
        # CONV => RELU => POOL
        model.add(Conv2D(20, kernel_size=5, padding="same",input_shape=input_shape))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        # CONV => RELU => POOL
        model.add(Conv2D(50, kernel_size=5, border_mode="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        # Flatten => RELU layers
        model.add(Flatten())
        model.add(Dense(500))
        model.add(Activation("relu"))
        # a softmax classifier
        model.add(Dense(classes))
        model.add(Activation("softmax"))
        return model
```



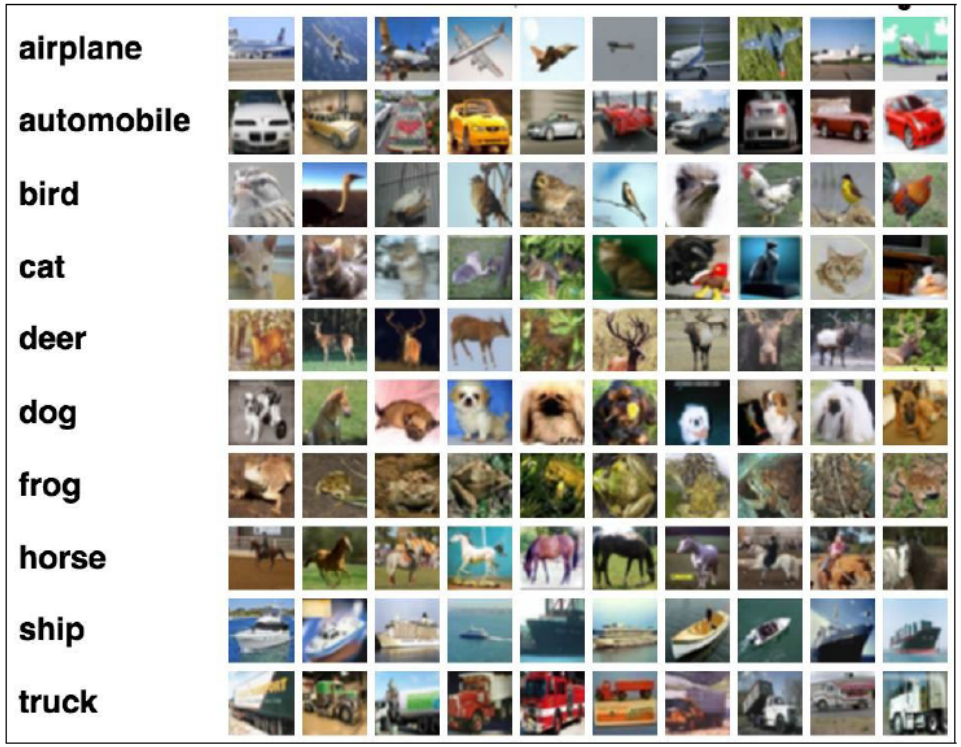
Example of CNN: LeNet

```
# network and training
NB_EPOCH = 20
BATCH_SIZE = 128
VERBOSE = 1
OPTIMIZER = Adam()
VALIDATION_SPLIT=0.2
IMG_ROWS, IMG_COLS = 28, 28 # input image dimensions
NB_CLASSES = 10 # number of outputs = number of digits
INPUT_SHAPE = (1, IMG_ROWS, IMG_COLS)
# data: shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
k.set_image_dim_ordering("th")
# consider them as float and normalize
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
# we need a 60K x [1 x 28 x 28] shape as input to the CONVNET
```

```
X_train = X_train[:, np.newaxis, :, :]
X_test = X_test[:, np.newaxis, :, :]
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
# convert class vectors to binary class matrices
y_train = np_utils.to_categorical(y_train, NB_CLASSES)
y_test = np_utils.to_categorical(y_test, NB_CLASSES)
# initialize the optimizer and model
model = LeNet.build(input_shape=INPUT_SHAPE, classes=NB_CLASSES)
model.compile(loss="categorical_crossentropy", optimizer=OPTIMIZER,
metrics=["accuracy"])
history = model.fit(X_train, y_train,
batch_size=BATCH_SIZE, epochs=NB_EPOCH,
verbose=VERBOSE, validation_split=VALIDATION_SPLIT)
score = model.evaluate(X_test, y_test, verbose=VERBOSE)
print("Test score:", score[0])
print('Test accuracy:', score[1])
# list all data in history
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Recognizing CIFAR-10 images with deep learning

- The CIFAR-10 dataset contains 60,000 color images of 32 x 32 pixels in 3 channels divided into 10 classes.
- Each class contains 6,000 images. The training set contains 50,000 images and test sets provides 10,000 images.
- The goal is to recognize previously unseen images and assign them to one of the 10 classes.



Recognizing CIFAR-10 images with deep learning

```
from keras.datasets import cifar10
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.optimizers import SGD, Adam, RMSprop
import matplotlib.pyplot as plt

# CIFAR_10 is a set of 60K images 32x32 pixels on 3 channels
IMG_CHANNELS = 3
IMG_ROWS = 32
IMG_COLS = 32

#constant
BATCH_SIZE = 128
NB_EPOCH = 20
NB_CLASSES = 10
VERBOSE = 1
VALIDATION_SPLIT = 0.2
```

```
OPTIM = RMSprop()

#load dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
print('X_train shape:', X_train.shape)
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
```

```
# convert to categorical
Y_train = np_utils.to_categorical(y_train, NB_CLASSES)
Y_test = np_utils.to_categorical(y_test, NB_CLASSES)

# float and normalization
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
```

```
# network
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
input_shape=(IMG_ROWS, IMG_COLS, IMG_CHANNELS)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

```
model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(NB_CLASSES))
model.add(Activation('softmax'))
model.summary()
```

Recognizing CIFAR-10 images with deep learning

```
# train
model.compile(loss='categorical_crossentropy', optimizer=OPTIM,
metrics=['accuracy'])
model.fit(X_train, Y_train, batch_size=BATCH_SIZE,
epochs=NB_EPOCH, validation_split=VALIDATION_SPLIT,
verbose=VERBOSE)
score = model.evaluate(X_test, Y_test,
batch_size=BATCH_SIZE, verbose=VERBOSE)
print("Test score:", score[0])
print('Test accuracy:', score[1])
```

```
#save model
model_json = model.to_json()
open('cifar10_architecture.json', 'w').write(model_json)
And the weights learned by our deep network on the training set
model.save_weights('cifar10_weights.h5', overwrite=True)
```

```
gulli-macbookpro:code gulli$ python keras_CIFAR10_simple.py
Using TensorFlow backend.
('X_train shape:', (50000, 3, 32, 32))
(50000, 'train samples')
(10000, 'test samples')
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
convolution2d_1 (Convolution2D)  (None, 32, 32, 32)    896         convolution2d_input_1[0][0]
____________________________________________________________________________________________________
activation_1 (Activation)        (None, 32, 32, 32)    0           convolution2d_1[0][0]
____________________________________________________________________________________________________
maxpooling2d_1 (MaxPooling2D)    (None, 32, 16, 16)    0           activation_1[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 32, 16, 16)    0           maxpooling2d_1[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 8192)          0           dropout_1[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 512)           4194816     flatten_1[0][0]
____________________________________________________________________________________________________
activation_2 (Activation)        (None, 512)           0           dense_1[0][0]
____________________________________________________________________________________________________
dropout_2 (Dropout)              (None, 512)           0           activation_2[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 10)            5130        dropout_2[0][0]
____________________________________________________________________________________________________
activation_3 (Activation)        (None, 10)            0           dense_2[0][0]
====================================================================================================
Total params: 4200842
____________________________________________________________________________________________________
Train on 40000 samples, validate on 10000 samples
Epoch 1/20
40000/40000 [==============================] - 114s - loss: 1.7380 - acc: 0.3855 - val_loss: 1.5353 - val_acc: 0.4376
Epoch 2/20
40000/40000 [==============================] - 114s - loss: 1.3847 - acc: 0.5081 - val_loss: 1.2392 - val_acc: 0.5629
Epoch 3/20
40000/40000 [==============================] - 116s - loss: 1.2481 - acc: 0.5566 - val_loss: 1.2737 - val_acc: 0.5446
Epoch 4/20
40000/40000 [==============================] - 114s - loss: 1.1590 - acc: 0.5913 - val_loss: 1.1919 - val_acc: 0.5722
Epoch 5/20
40000/40000 [==============================] - 116s - loss: 1.0904 - acc: 0.6138 - val_loss: 1.0860 - val_acc: 0.6257
Epoch 6/20
40000/40000 [==============================] - 115s - loss: 1.0202 - acc: 0.6391 - val_loss: 1.0771 - val_acc: 0.6245
Epoch 7/20
40000/40000 [==============================] - 115s - loss: 0.9828 - acc: 0.6523 - val_loss: 1.0491 - val_acc: 0.6375
Epoch 8/20
40000/40000 [==============================] - 114s - loss: 0.9328 - acc: 0.6739 - val_loss: 1.0344 - val_acc: 0.6453
Epoch 9/20
40000/40000 [==============================] - 114s - loss: 0.8978 - acc: 0.6858 - val_loss: 1.0789 - val_acc: 0.6384
Epoch 10/20
40000/40000 [==============================] - 115s - loss: 0.8556 - acc: 0.7004 - val_loss: 1.0072 - val_acc: 0.6538
Epoch 11/20
40000/40000 [==============================] - 114s - loss: 0.8215 - acc: 0.7142 - val_loss: 1.1334 - val_acc: 0.6450
Epoch 12/20
40000/40000 [==============================] - 115s - loss: 0.7938 - acc: 0.7256 - val_loss: 1.0761 - val_acc: 0.6464
Epoch 13/20
40000/40000 [==============================] - 118s - loss: 0.7631 - acc: 0.7337 - val_loss: 1.0204 - val_acc: 0.6587
Epoch 14/20
40000/40000 [==============================] - 121s - loss: 0.7301 - acc: 0.7433 - val_loss: 0.9647 - val_acc: 0.6853
Epoch 15/20
40000/40000 [==============================] - 114s - loss: 0.7094 - acc: 0.7529 - val_loss: 1.0852 - val_acc: 0.6604
Epoch 16/20
40000/40000 [==============================] - 114s - loss: 0.6872 - acc: 0.7608 - val_loss: 1.0144 - val_acc: 0.6680
Epoch 17/20
40000/40000 [==============================] - 115s - loss: 0.6642 - acc: 0.7682 - val_loss: 0.9787 - val_acc: 0.6781
Epoch 18/20
40000/40000 [==============================] - 114s - loss: 0.6524 - acc: 0.7758 - val_loss: 1.0035 - val_acc: 0.6803
Epoch 19/20
40000/40000 [==============================] - 114s - loss: 0.6302 - acc: 0.7834 - val_loss: 1.1088 - val_acc: 0.6571
Epoch 20/20
40000/40000 [==============================] - 113s - loss: 0.6081 - acc: 0.7902 - val_loss: 1.0744 - val_acc: 0.6672
Testing...
10000/10000 [==============================] - 13s
('\nTest score:', 1.0762448620796203)
('Test accuracy:', 0.66490000000000005)
['acc', 'loss', 'val_acc', 'val_loss']
```

Improving the CIFAR-10 performance with deeper network

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
input_shape=(IMG_ROWS, IMG_COLS, IMG_CHANNELS)))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, 3, 3))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(NB_CLASSES))
model.add(Activation('softmax'))
```

```
gulli-macbookpro:code gulli$ python keras_CIFAR10_V1.py
Using TensorFlow backend.
('X_train shape:', (50000, 3, 32, 32))
(50000, 'train samples')
(10000, 'test samples')
```

Layer (type)	Output Shape	Param #	Connected to
convolution2d_1 (Convolution2D)	(None, 32, 32, 32)	896	convolution2d_input_1[0][0]
activation_1 (Activation)	(None, 32, 32, 32)	0	convolution2d_1[0][0]
convolution2d_2 (Convolution2D)	(None, 32, 32, 32)	9248	activation_1[0][0]
activation_2 (Activation)	(None, 32, 32, 32)	0	convolution2d_2[0][0]
maxpooling2d_1 (MaxPooling2D)	(None, 32, 16, 16)	0	activation_2[0][0]
dropout_1 (Dropout)	(None, 32, 16, 16)	0	maxpooling2d_1[0][0]
convolution2d_3 (Convolution2D)	(None, 64, 16, 16)	18496	dropout_1[0][0]
activation_3 (Activation)	(None, 64, 16, 16)	0	convolution2d_3[0][0]
convolution2d_4 (Convolution2D)	(None, 64, 14, 14)	36928	activation_3[0][0]
activation_4 (Activation)	(None, 64, 14, 14)	0	convolution2d_4[0][0]
maxpooling2d_2 (MaxPooling2D)	(None, 64, 7, 7)	0	activation_4[0][0]
dropout_2 (Dropout)	(None, 64, 7, 7)	0	maxpooling2d_2[0][0]
flatten_1 (Flatten)	(None, 3136)	0	dropout_2[0][0]
dense_1 (Dense)	(None, 512)	1606144	flatten_1[0][0]
activation_5 (Activation)	(None, 512)	0	dense_1[0][0]
dropout_3 (Dropout)	(None, 512)	0	activation_5[0][0]
dense_2 (Dense)	(None, 10)	5130	dropout_3[0][0]
activation_6 (Activation)	(None, 10)	0	dense_2[0][0]

```
Total params: 1676842
____________________________________________________________________________________________________
Train on 40000 samples, validate on 10000 samples
Epoch 1/40
40000/40000 [==============================] - 430s - loss: 1.8179 - acc: 0.3443 - val_loss: 1.5250 - val_acc: 0.4551
Epoch 2/40
40000/40000 [==============================] - 382s - loss: 1.3506 - acc: 0.5182 - val_loss: 1.1998 - val_acc: 0.5714
```

Improving the CIFAR-10 performance with data augmentation

```
from keras.preprocessing.image import ImageDataGenerator
from keras.datasets import cifar10
import numpy as np
NUM_TO_AUGMENT=5

#load dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# augumenting
print("Augmenting training set images...")
datagen = ImageDataGenerator(
rotation_range=40,
width_shift_range=0.2,
height_shift_range=0.2,
zoom_range=0.2,
horizontal_flip=True,
fill_mode='nearest')
```

```
xtas, ytas = [], []
for i in range(X_train.shape[0]):
num_aug = 0
x = X_train[i] # (3, 32, 32)
x = x.reshape((1,) + x.shape) # (1, 3, 32, 32)
for x_aug in datagen.flow(x, batch_size=1,
save_to_dir='preview', save_prefix='cifar', save_format='jpeg'):
if num_aug >= NUM_TO_AUGMENT:
break
xtas.append(x_aug[0])
num_aug += 1
```

```
#fit the dataget
datagen.fit(X_train)

# train
history = model.fit_generator(datagen.flow(X_train, Y_train,
batch_size=BATCH_SIZE), samples_per_epoch=X_train.shape[0],
epochs=NB_EPOCH, verbose=VERBOSE)
score = model.evaluate(X_test, Y_test,
batch_size=BATCH_SIZE, verbose=VERBOSE)
print("Test score:", score[0])
print('Test accuracy:', score[1])
```

Very deep convolutional networks for large-scale image recognition

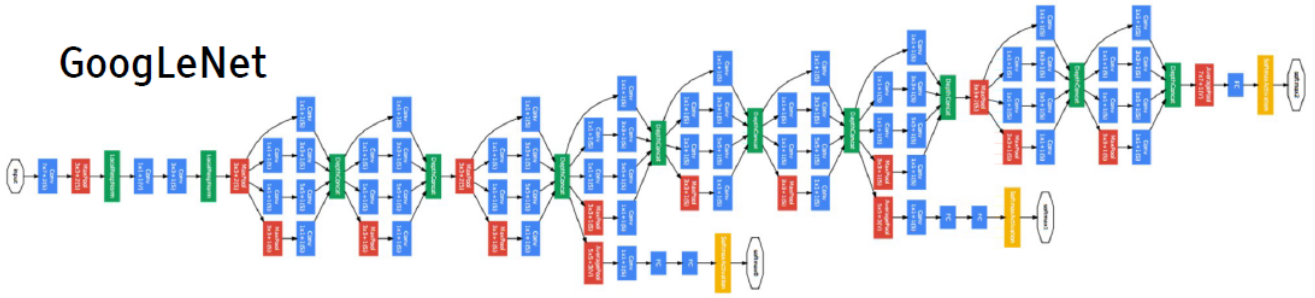
```
from keras.models import Sequential
from keras.layers.core import Flatten, Dense, Dropout
from keras.layers.convolutional import Conv2D, MaxPooling2D, ZeroPadding2D
from keras.optimizers import SGD
import cv2, numpy as np

# define a VGG16 network
def VGG_16(weights_path=None):
model = Sequential()
model.add(ZeroPadding2D((1,1),input_shape=(3,224,224)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(512, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(512, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(512, (3, 3), activation='relu'))
```

```
model.add(MaxPooling2D((2,2), strides=(2,2)))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(512, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(512, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Conv2D(512, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))
model.add(Flatten())
#top layer of the VGG net
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1000, activation='softmax'))
if weights_path:
model.load_weights(weights_path)
return model
```

Very Deep ConvNet Architectures

- Small kernels, not much subsampling (fractional subsampling).



Back To Keras: Layers and Layers

Keras has a number of pre-built layers. Notable examples include:

- Regular dense, MLP type

```
keras.layers.core.Dense(output_dim,
                        init='glorot_uniform',
                        activation='linear',
                        weights=None,
                        W_regularizer=None, b_regularizer=None, activity_regularizer=None,
                        W_constraint=None, b_constraint=None,
                        input_dim=None)
```

- Recurrent layers, LSTM, GRU, etc.

```
keras.layers.recurrent.GRU(output_dim,
        init='glorot_uniform', inner_init='orthogonal',
        activation='sigmoid', inner_activation='hard_sigmoid',
        return_sequences=False,
        go_backwards=False,
        stateful=False,
        input_dim=None, input_length=None)
```

Layers and Layers

- 1D Convolutional layers

```
keras.layers.convolutional.Convolution1D(nb_filter, filter_length,
        init='uniform',
        activation='linear',
        weights=None,
        border_mode='valid',
        subsample_length=1,
        W_regularizer=None, b_regularizer=None,
        W_constraint=None, b_constraint=None,
        input_dim=None, input_length=None)
```

- 2D Convolutional layers

```
keras.layers.convolutional.Convolution2D(nb_filter, nb_row, nb_col,
        init='glorot_uniform',
        activation='linear',
        weights=None,
        border_mode='valid',
        subsample=(1, 1),
        W_regularizer=None, b_regularizer=None,
        W_constraint=None,
        dim_ordering='th')
```

Other types of layer include:

- Dropout
- Noise
- Pooling
- Normalization
- Embedding
- And many more...

- Autoencoders can be built with any other type of layer

```
from keras.layers import containers

# input shape: (nb_samples, 32)
encoder = containers.Sequential([Dense(16, input_dim=32), Dense(8)])
decoder = containers.Sequential([Dense(16, input_dim=8), Dense(32)])

autoencoder = Sequential()
autoencoder.add(AutoEncoder(encoder=encoder, decoder=decoder, output_reconstruction=False))
```

Activations

More or less all your favourite activations are available:

- Sigmoid, tanh, ReLu, softplus, hard_sigmoid, linear
- Advanced activations implemented as a layer (after desired neural layer)
- Advanced activations: LeakyReLu, PReLu, ELU, Parametric Softplus, Thresholded linear and Thresholded Relu

Objectives and Optimizers

Objective Functions:

- Error loss: rmse, mse, mae, mape, msle
- Hinge loss: squared_hinge, hinge
- Class loss: binary_crossentropy, categorical_crossentropy

Optimization:

- Provides SGD, Adagrad, Adadelta, Rmsprop and Adam
- All optimizers can be customized via parameters

Parallel Capabilities

- Training time is drastically reduced thanks to Theano's GPU support
- Theano compiles into CUDA, NVIDIA's GPU API
- Currently will only work with NVIDIA cards but Theano is working on OpenCL version
- TensorFlow has similar support
- THEANO_FLAGS=mode=FAST_RUN,device=gpu, floatX=float32 python your_net.py

Architecture/Weight Saving and Loading

- Model architectures can be saved and loaded

```
# save as JSON
json_string = model.to_json()
# save as YAML
yaml_string = model.to_yaml()

# model reconstruction from JSON:
from keras.models import model_from_json
model = model_from_json(json_string)

# model reconstruction from YAML
model = model_from_yaml(yaml_string)
```

- Model parameters (weights) can be saved and loaded

```
model.save_weights('my_model_weights.h5')
model.load_weights('my_model_weights.h5')
```

Callbacks

Allow for function call during training

- Callbacks can be called at different points of training (batch or epoch)
- Existing callbacks: Early Stopping, weight saving after epoch
- Easy to build and implement, called in training function, fit()

In Summary

Pros:

- Easy to implement
- Lots of choice
- Extendible and customizable
- GPU
- High level
- Active community
- keras.io

Cons:

- Lack of generative models
- High level
- Theano overhead

Thanks for your attentions !