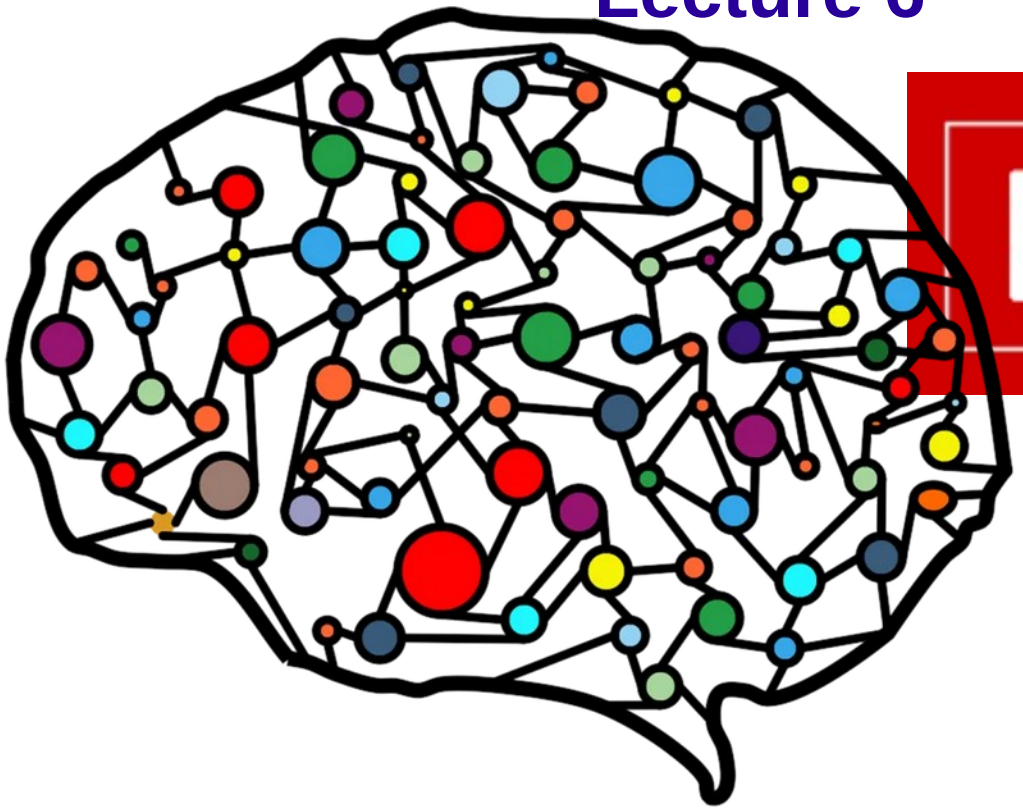
Machine learning

Lecture 6



- How to build a Deep Neural Networks – Keras tutorial
- Convolutional Deep Neural Network

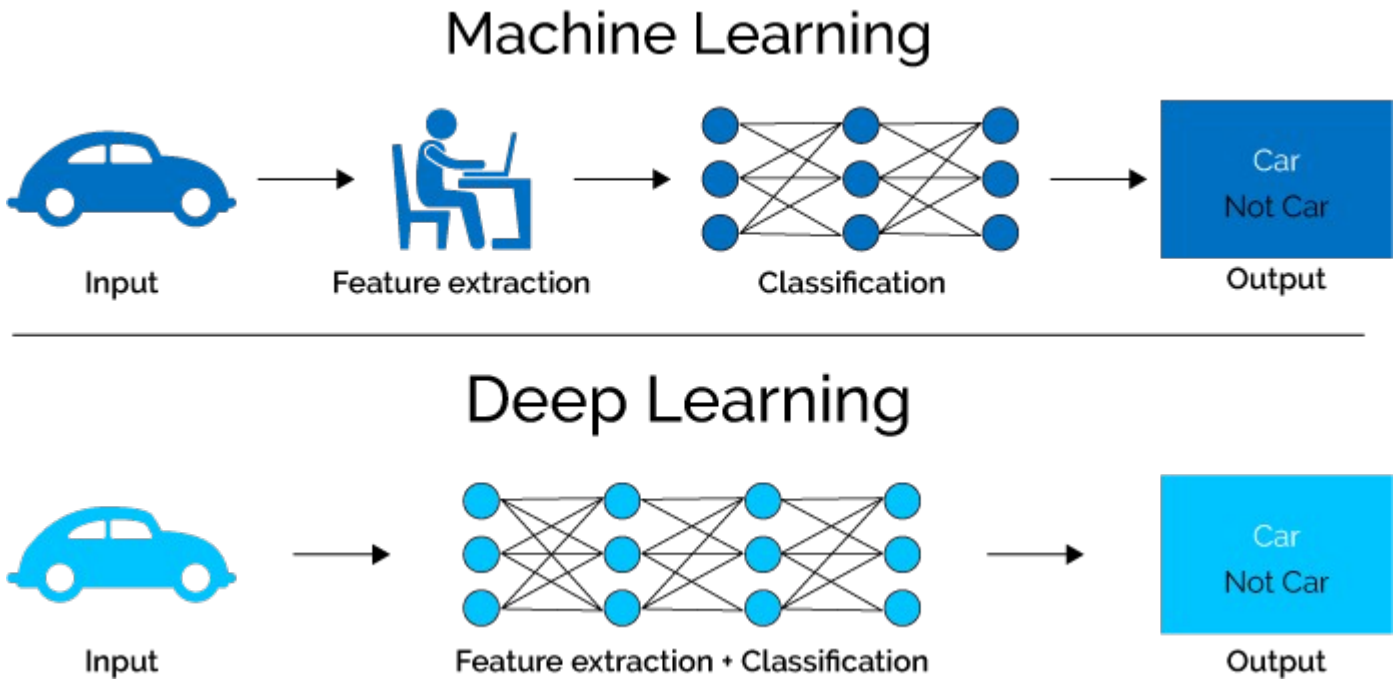
Marcin Wolter

IFJ PAN

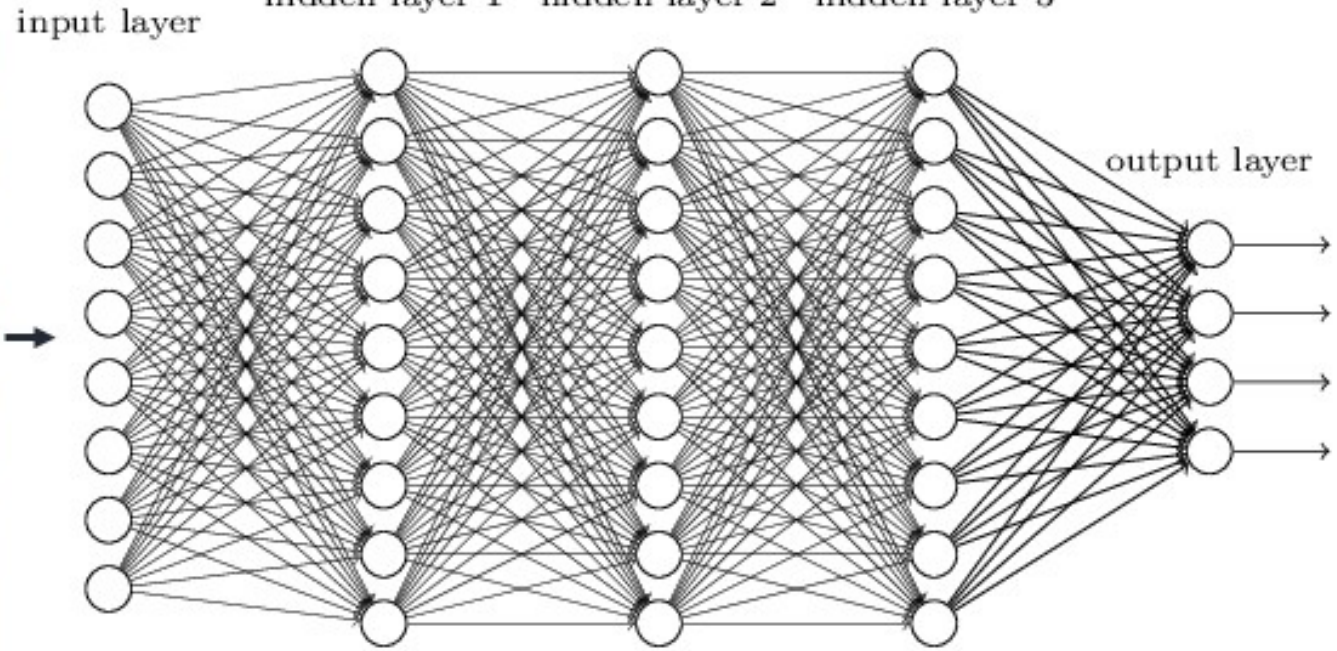
17 January 2020

Machine Learning and Deep Learning

- **Traditional ML (BDT, NN etc)** – the scientist finds good, well discriminating variables (~10), called “features”, and performs classification using them as inputs for the ML algorithm.
- **Deep Learning** – thousands or millions of input variables (like pixels of a photo), the features are *automagically* extracted during training.

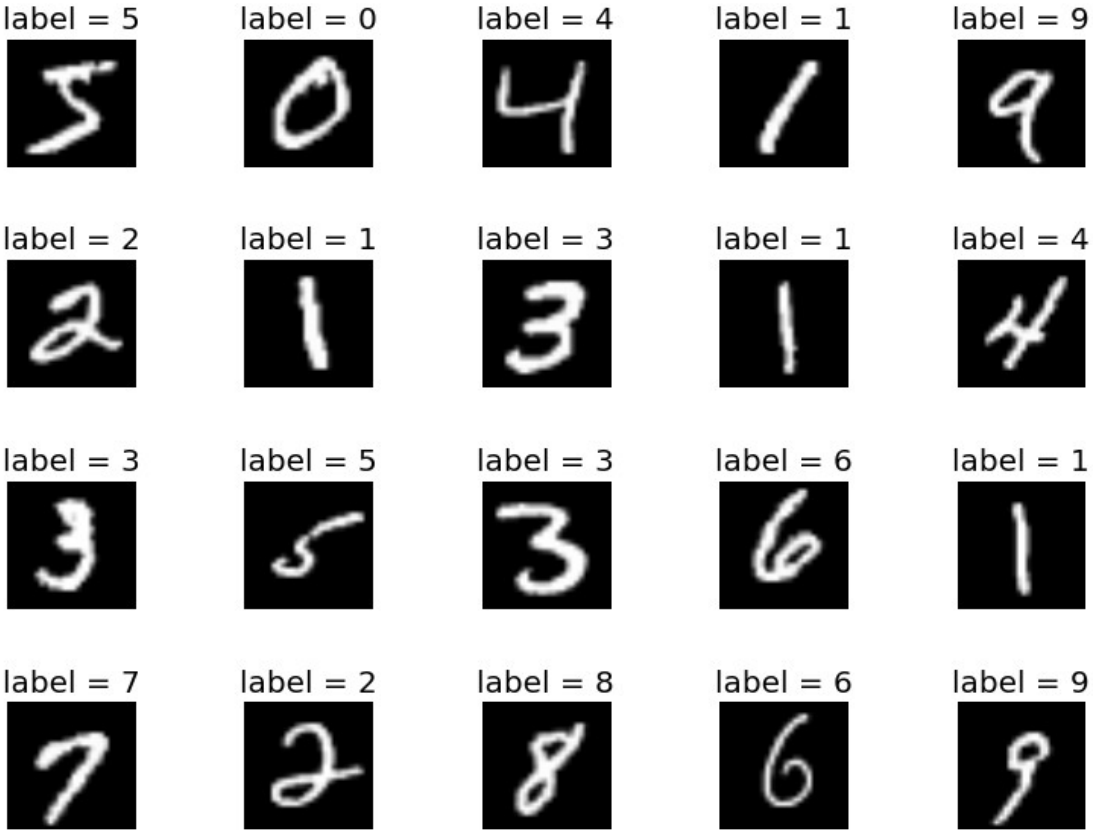


Deep neural networks learn hierarchical feature representations



A tutorial – how to design a Keras DNN

- Task – build a simple network to recognize hand-written digits:



28 x 28 pixels

60000 train samples
10000 test samples
Model: "sequential_3"

Layer (type)	Output Shape	Param #
dense_9 (Dense)	(None, 512)	401920
dropout_7 (Dropout)	(None, 512)	0
dense_10 (Dense)	(None, 512)	262656
dropout_8 (Dropout)	(None, 512)	0
dense_11 (Dense)	(None, 512)	262656
dropout_9 (Dropout)	(None, 512)	0
dense_12 (Dense)	(None, 10)	5130

Total params: 932,362
Trainable params: 932,362
Non-trainable params: 0

Init

The first step is to define the functions and classes we intend to use in this tutorial. We will use the NumPy library to load our dataset and we will use two classes from the Keras library to define our model.

The imports required are listed below.

```
import matplotlib.pyplot as plt  # matplotlib plotting
import numpy as np

import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop, Adam
```

Load Data

We can now load our dataset:

```
# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

MNIST database of handwritten digits

Dataset of 60,000 28x28 grayscale images of the 10 digits, along with a test set of 10,000 images.
Usage:

from keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()

Returns:
2 tuples:
x_train, x_test: uint8 array of grayscale image data with shape (num_samples, 28, 28).
y_train, y_test: uint8 array of digit labels (integers in range 0-9) with shape (num_samples,).

MNIST dataset

6	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0

784 numbers

We make now a numpy array of shape (6000, 784) out of a python tuple

```
# reshape dataset
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)

# convert to float32
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

#normalize to one
x_train /= 255
x_test /= 255
```

Prepare data
convert to categorical

We want to use NN with 10 outputs (each corresponding to one digit) to get a probability for each digit. So we convert the y_train from a single number to vector:

- 7 → (0, 0, 0, 0, 0, 0, 0, 1, 0)
- 0 → (1, 0, 0, 0, 0, 0, 0, 0, 0)
- 9 → (0, 0, 0, 0, 0, 0, 0, 0, 1)

```
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

num_classes = 10
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

Define Keras Model

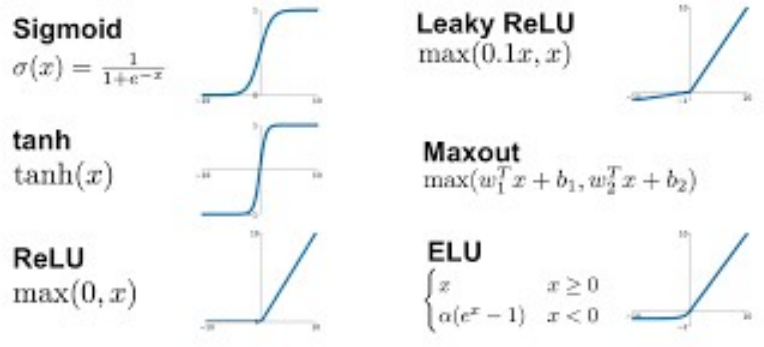
Models in Keras are defined as a sequence of layers.

We create a Sequential model and add layers one at a time until we are happy with our network architecture.

The first thing to get right is to ensure the input layer has the right number of input features. This can be specified when creating the first layer with 512 nodes and with the input_dim argument and setting it to 784 for the 784 input variables.

```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
```

The activation function is relu (Rectified Linear):



Define Keras Model

Adding next layers. How do we know the number of layers and their types?

This is a very hard question. There are heuristics that we can use and often the best network structure is found through a process of trial and error experimentation. Generally, you need a network large enough to capture the structure of the problem.

In this example, we will use a fully-connected network structure with three hidden layers.

Fully connected layers are defined using the Dense class. We can specify the number of neurons or nodes in the layer as the first argument, and specify the activation function using the activation argument.

We will use the rectified linear unit activation function referred to as ReLU on the first three layers:

```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='relu'))
```

Define Keras Model

```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.summary()
```

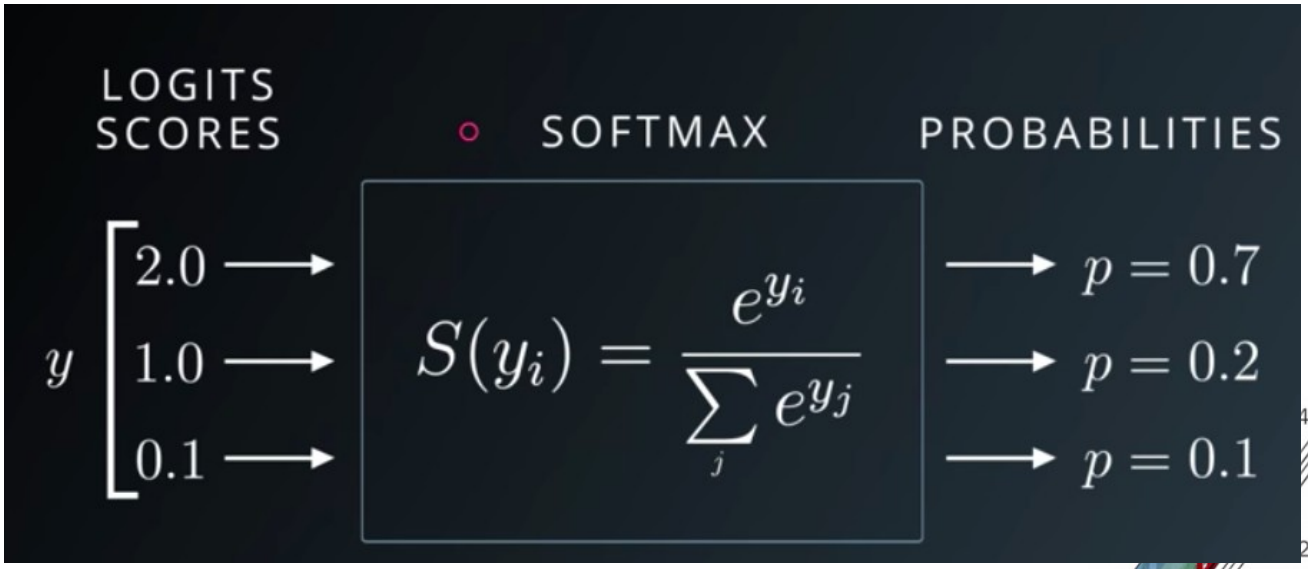
Adding output layer with num_classes=10 nodes and softmax (see next slide) activation function. We use a sigmoid on the output layer to ensure our network output is between 0 and 1 and easy to map to either a probability of class 1 or snap to a hard classification of either class with a default threshold of 0.5.

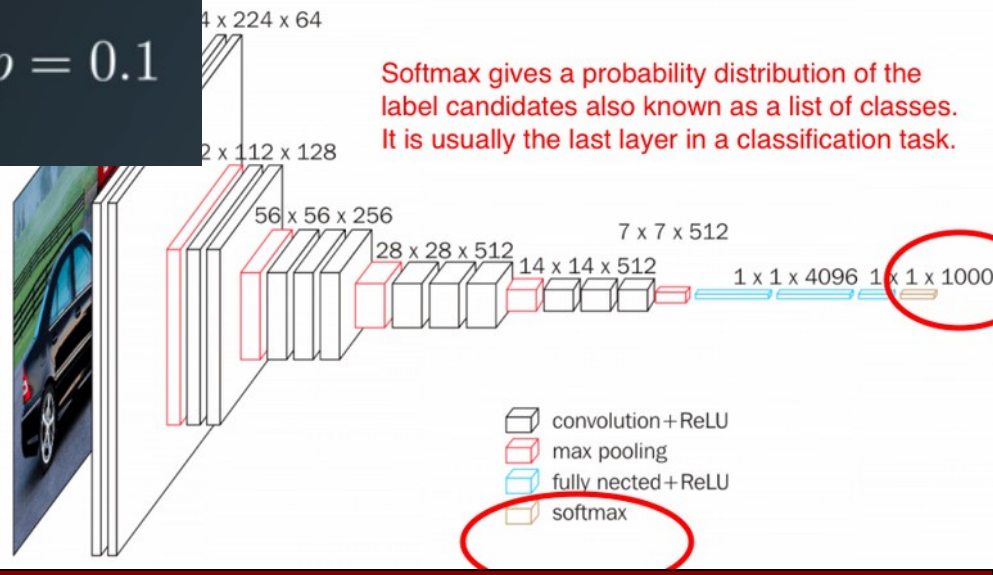
Between the layers we add a Dropout layer to avoid overtraining:
Dropout consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting.
http://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

`model.summary()` - print the network structure

Softmax activation function

Softmax function, a wonderful activation function that turns numbers aka logits into probabilities that sum to one. Softmax function outputs a vector that represents the probability distributions of a list of potential outcomes. It's also a core element used in deep learning classification tasks.





Train the network

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

When compiling, we must specify some additional properties required when training the network. Training a network means finding the best set of weights to map inputs to outputs in our dataset.
We must specify the loss function to use to evaluate a set of weights, the optimizer is used to search through different weights for the network and any optional metrics we would like to collect and report during training.

This loss is for a categorical classification problems and is defined in Keras as "*categorical_crossentropy*". You can learn more about choosing loss functions based on your problem here:
https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/

We will define the optimizer as the efficient stochastic gradient descent algorithm "*RMSprop*". We could also use "adam", which is a popular version of gradient descent because it automatically tunes itself and gives good results in a wide range of problems.

Fit Keras Model

```
batch_size = 128
epochs = 10

history = model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
```

We can train or fit our model on our loaded data by calling the fit() function on the model. Training occurs over epochs and each epoch is split into batches.

Epoch: One pass through all of the rows in the training dataset.
Batch: One or more samples considered by the model within an epoch before weights are updated.
One epoch is comprised of one or more batches, based on the chosen batch size and the model is fit for many epochs.
For this problem, we will run for a small number of epochs (10) and use a batch size of 128.

These configurations can be chosen experimentally by trial and error. We want to train the model enough so that it learns a good (or good enough) mapping of rows of input data to the output classification. The model will always have some error, but the amount of error will level out after some point for a given model configuration. This is called model convergence.

Evaluate Keras Model

```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test accuracy:', score[1])
```

We have trained our neural network on the entire dataset and we can evaluate the performance of the network on another “test” dataset.
You can evaluate your model on a dataset using the evaluate() function.
This will generate a prediction for each input and output pair and collect scores, including the average loss and any metrics you have configured, such as accuracy.
The evaluate() function will return a list with two values. The first will be the loss of the model on the dataset and the second will be the accuracy of the model on the dataset.

Tie It All Together

https://github.com/marcinwolter/MachineLearnin2019/blob/master/mnist_mlp_minimal.ipynb

Model: "sequential_2"

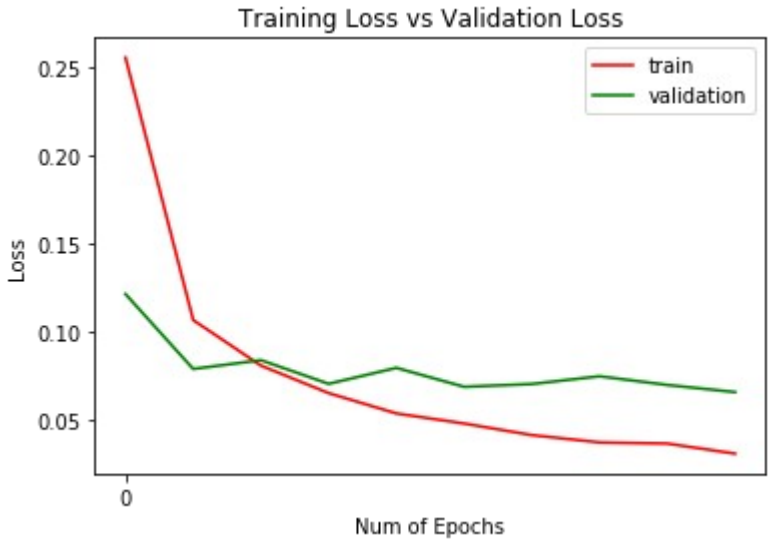
Layer (type)	Output Shape	Param #
dense_5 (Dense)	(None, 512)	401920
dropout_4 (Dropout)	(None, 512)	0
dense_6 (Dense)	(None, 512)	262656
dropout_5 (Dropout)	(None, 512)	0
dense_7 (Dense)	(None, 512)	262656
dropout_6 (Dropout)	(None, 512)	0
dense_8 (Dense)	(None, 10)	5130

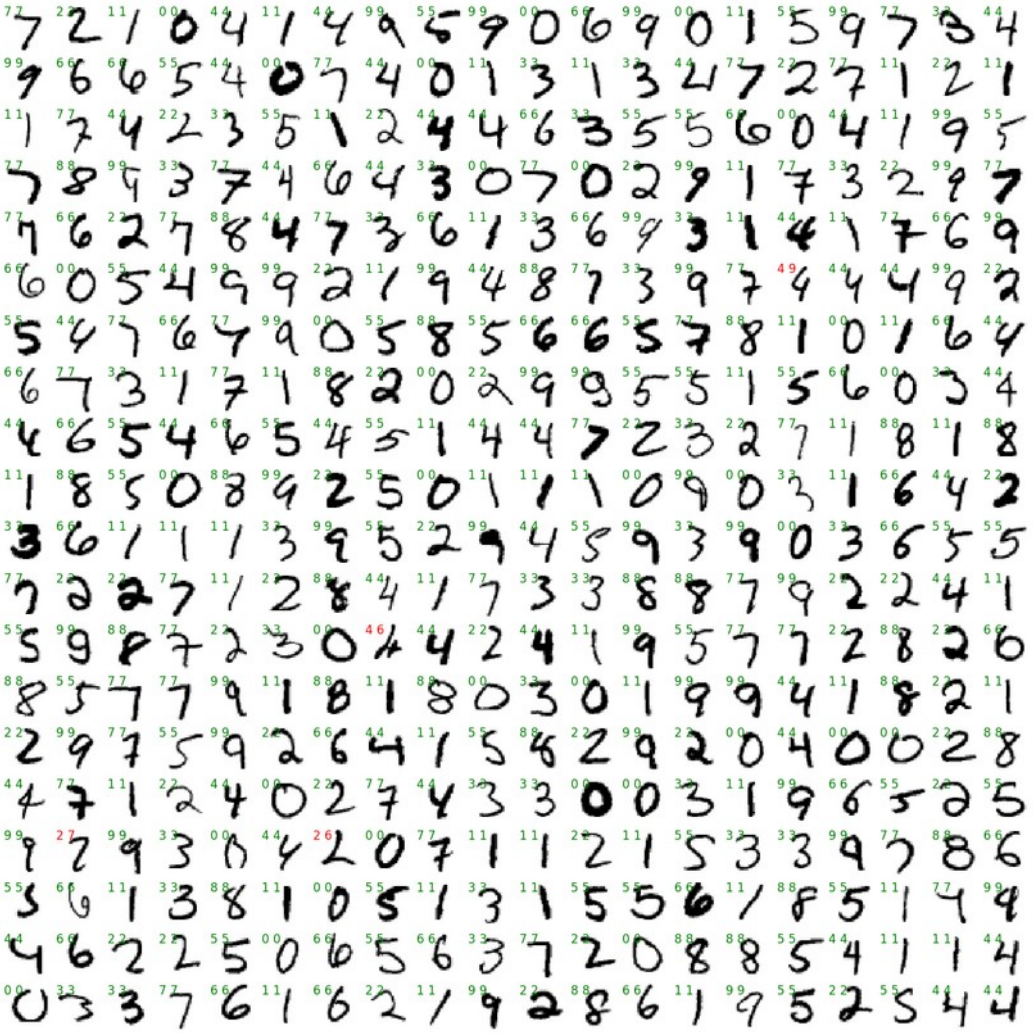
Total params: 932,362
Trainable params: 932,362
Non-trainable params: 0

Program with more features

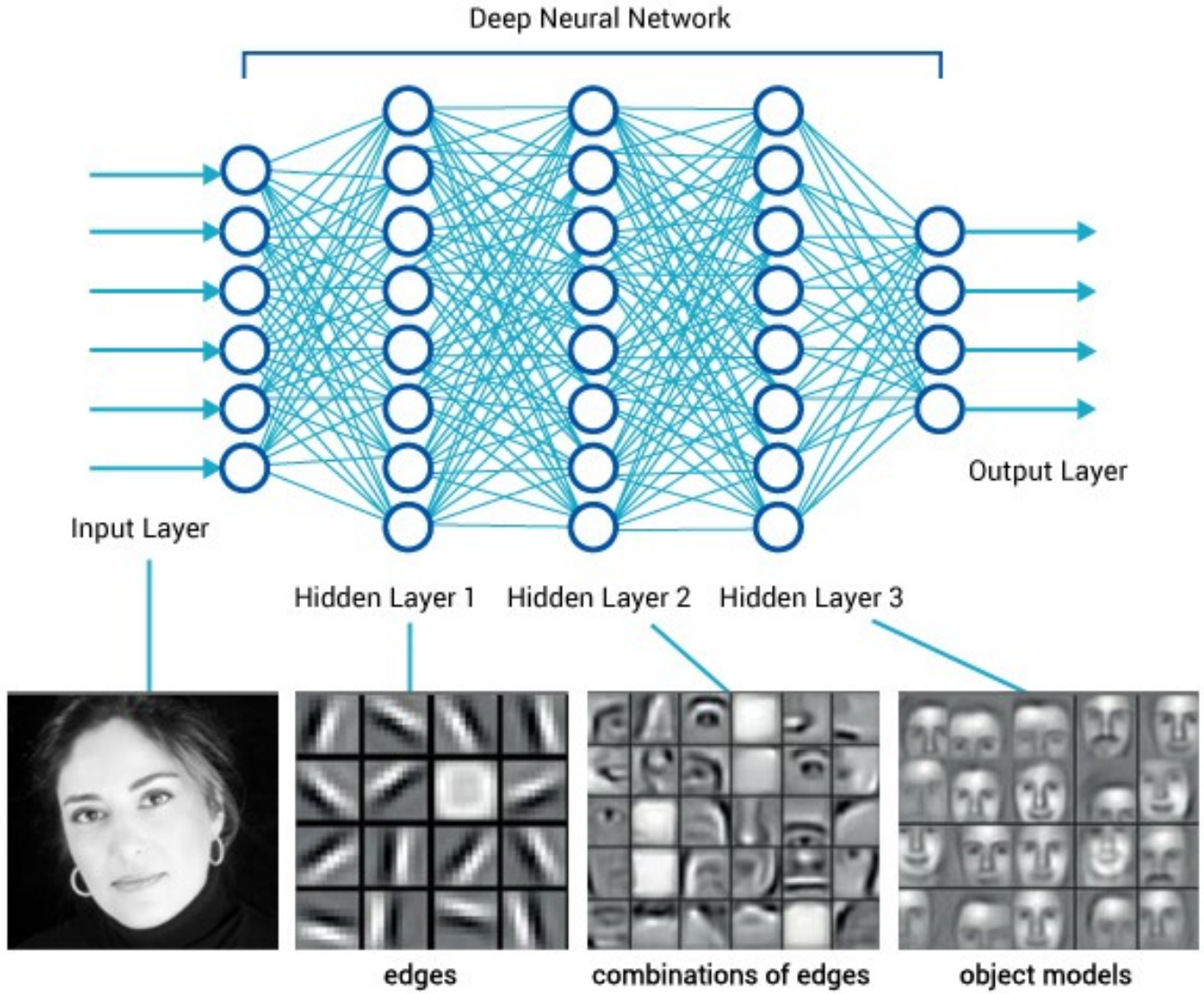
- https://github.com/marcinwolter/MachineLearnin2019/blob/master/mnist_mlp.ipynb

- Visualization of results
- Plotting the Neural Network structure

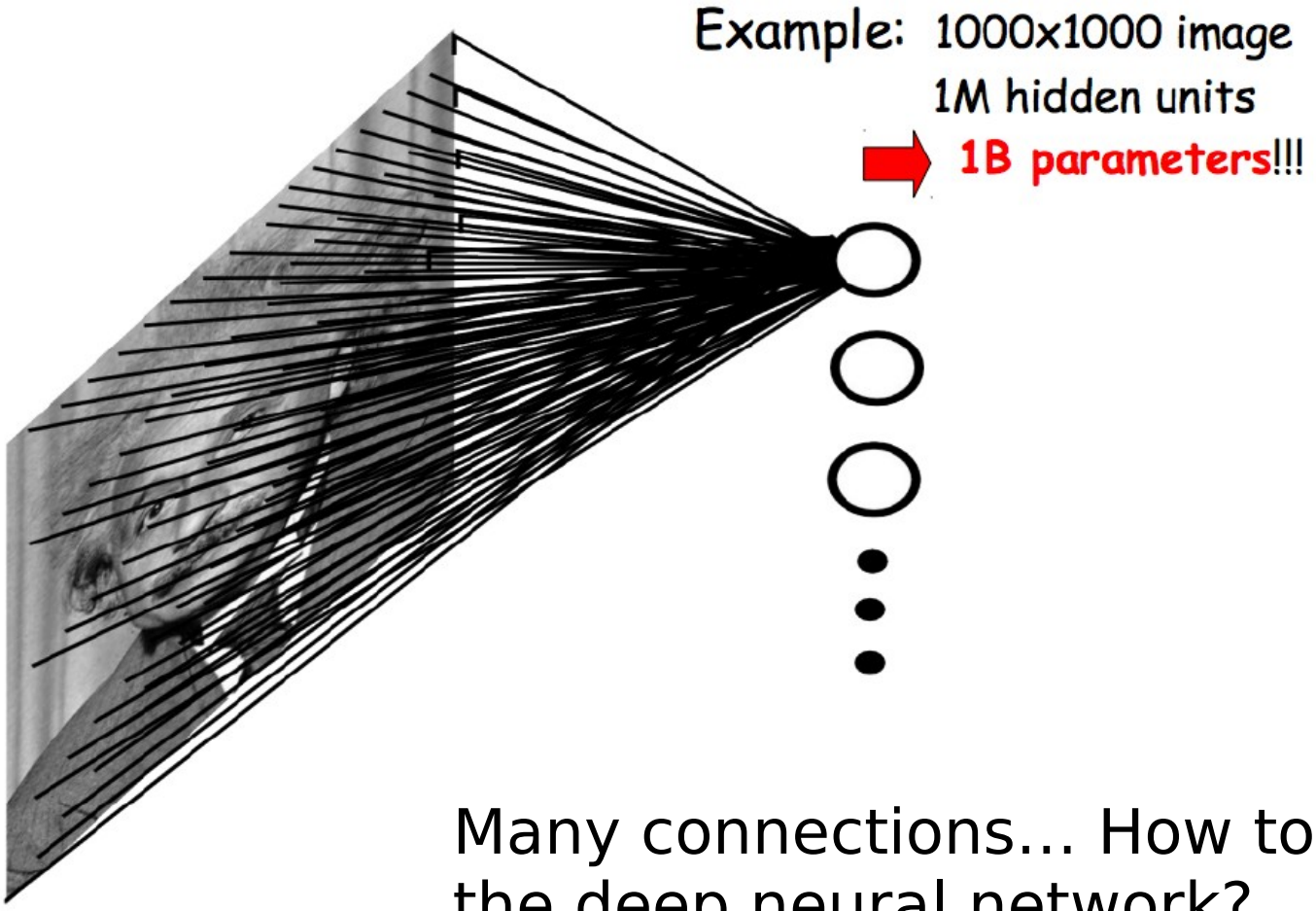




Deep Neural Network works like that...

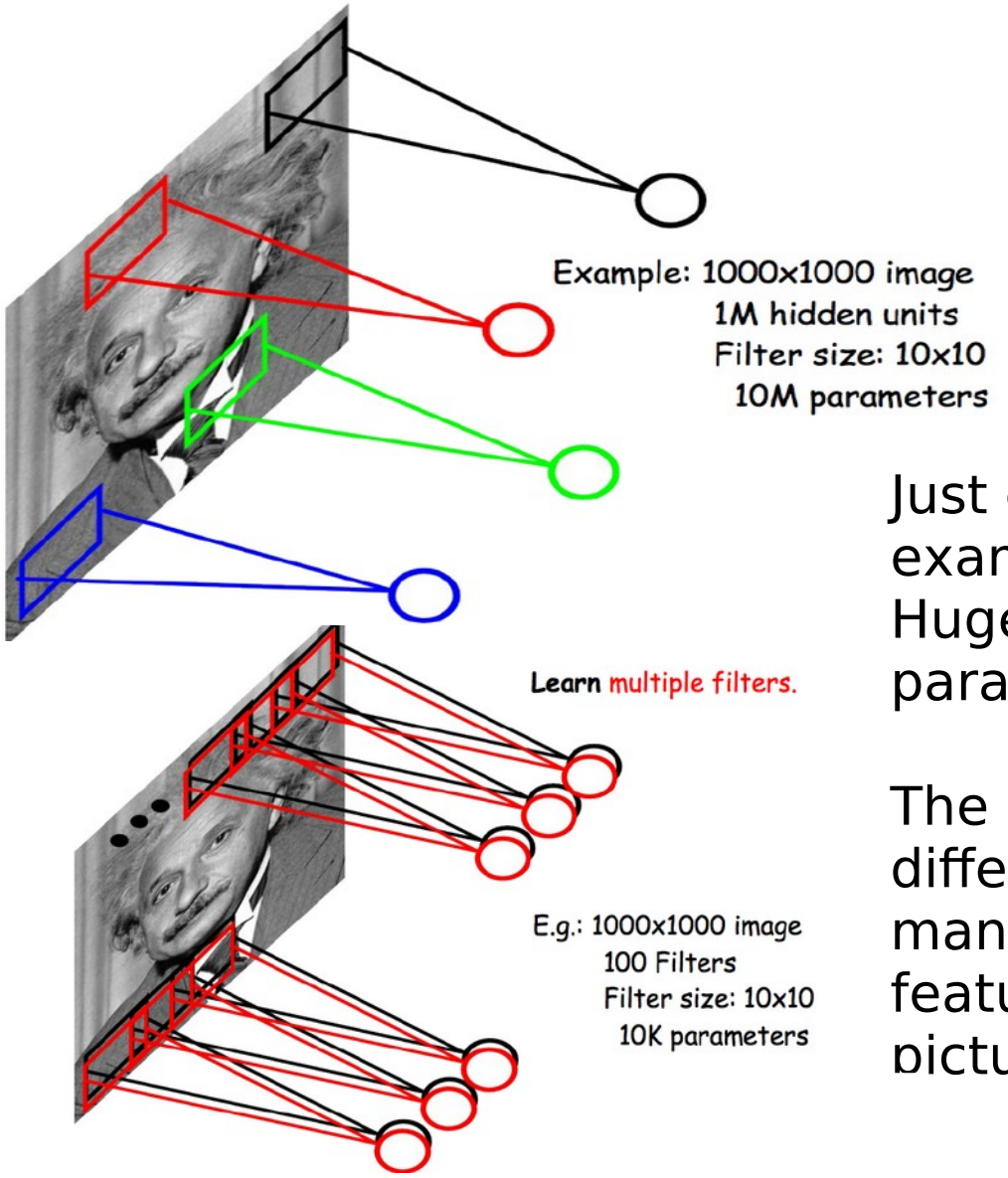


Convolutional NN
Pattern recognition



Many connections… How to simplify the deep neural network?

Convolutional NN

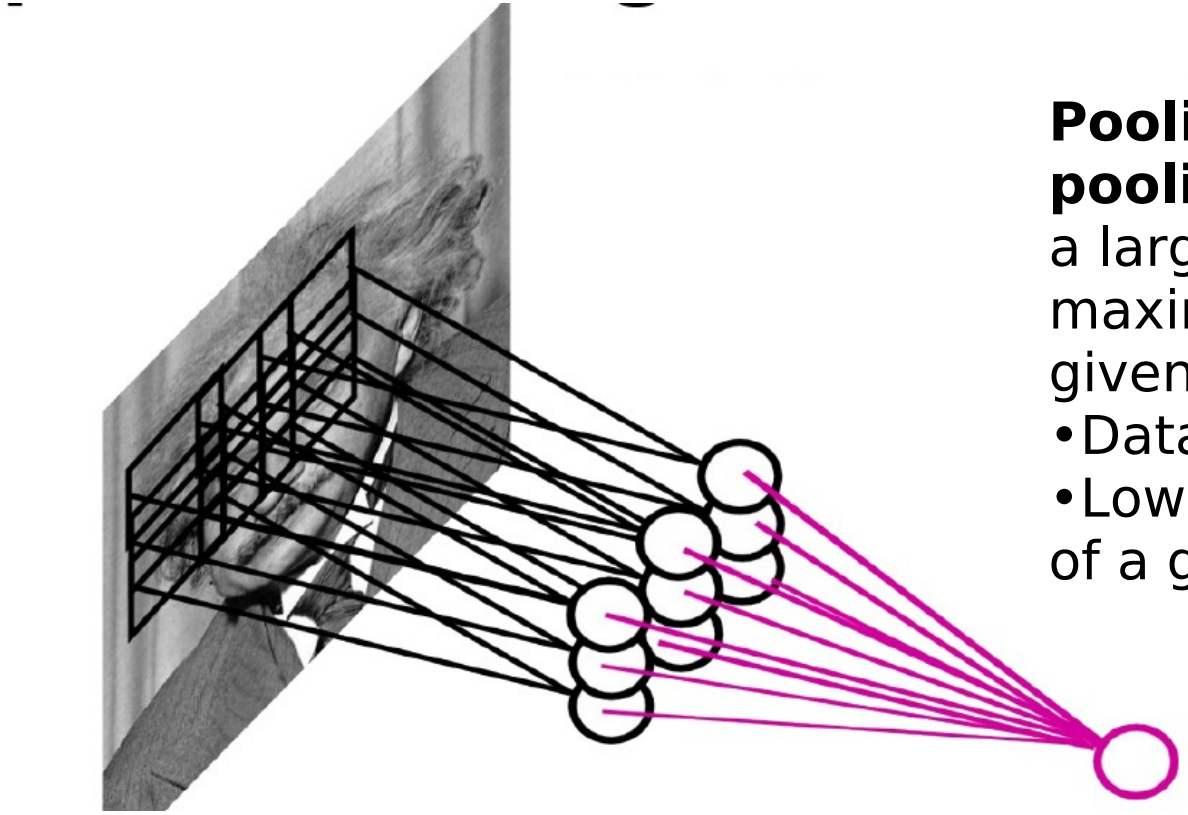


Just connect only local areas, for example 10x10 pixels.
Huge reduction of the number of parameters!

The same features might be found in different places => so we could train many filters, each recognizing another feature, and move them over the picture.

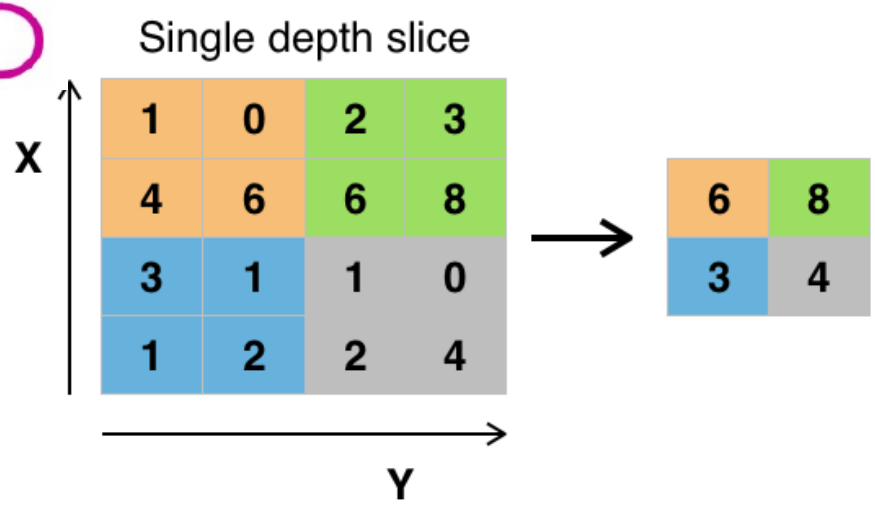
LeCun et al. "Gradient-based learning applied to document recognition" IEEE 1998

Pooling



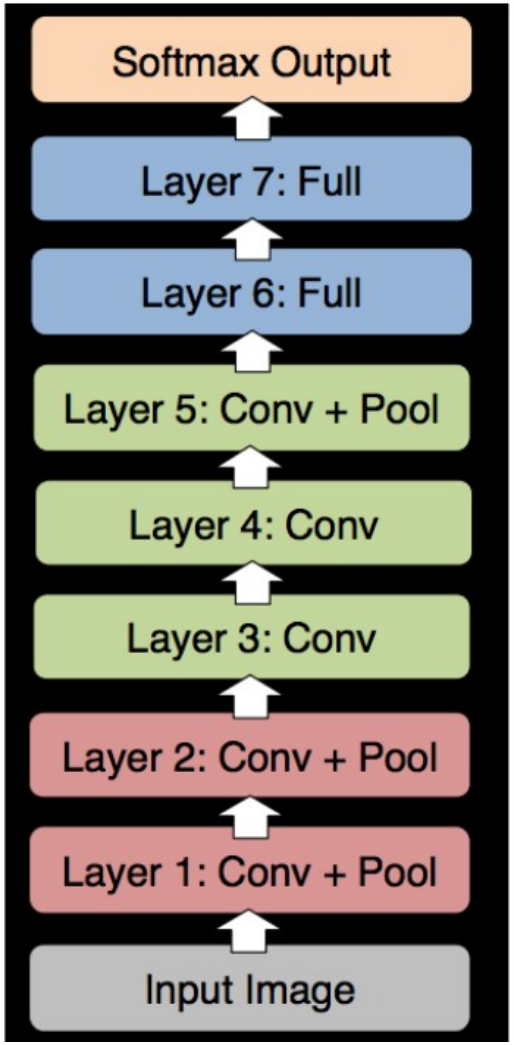
Pooling – (in most cases **max pooling**) the group of outputs for a larger input area is replaced by a maximum (or average) for this given area:

- Data reduction,
- Lower sensitivity for the position of a given feature.



Architecture of Alex Krizhevsky et al.

- 8 layers total.
- Trained on Imagenet Dataset (1000 categories, 1.2M training images, 150k test images)
- 18.2% top-5 error
  - Winner of the ILSVRC-2012 challenge.

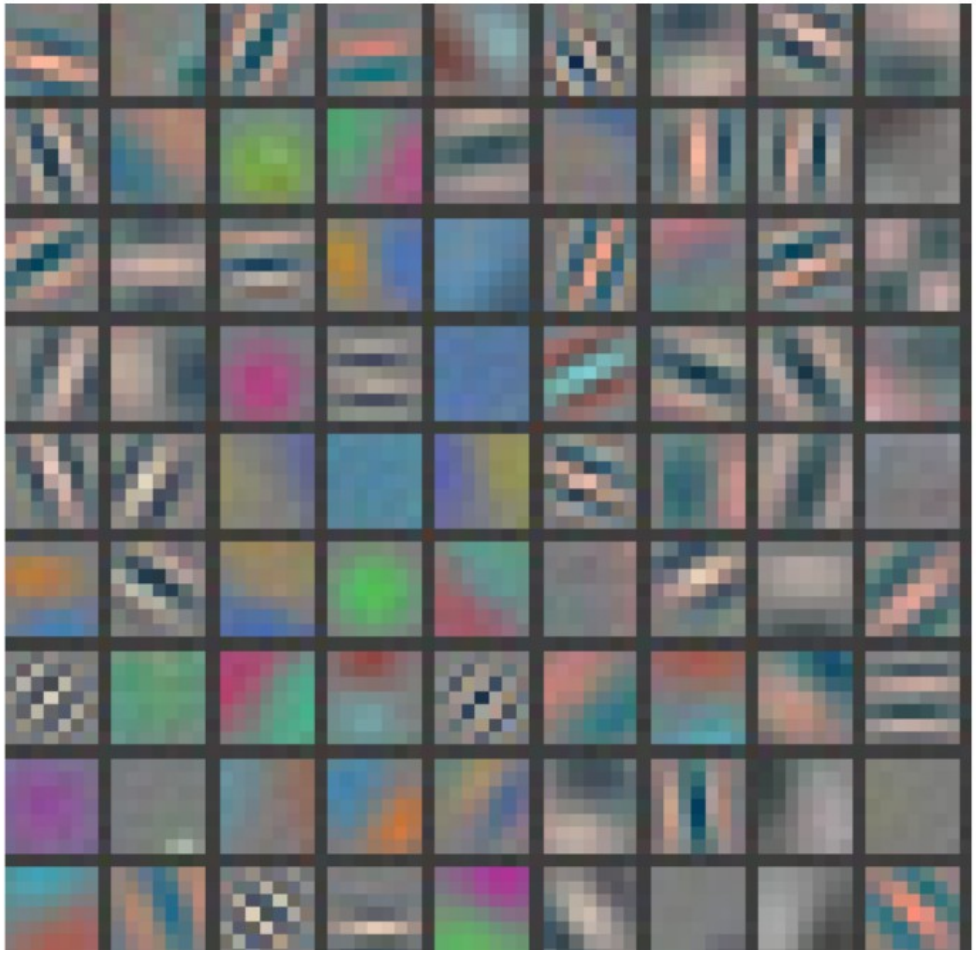


Slide: R. Fergus

First layer filters

Showing 81 filters of 11x11x3.

Capture low-level features like oriented edges, blobs.

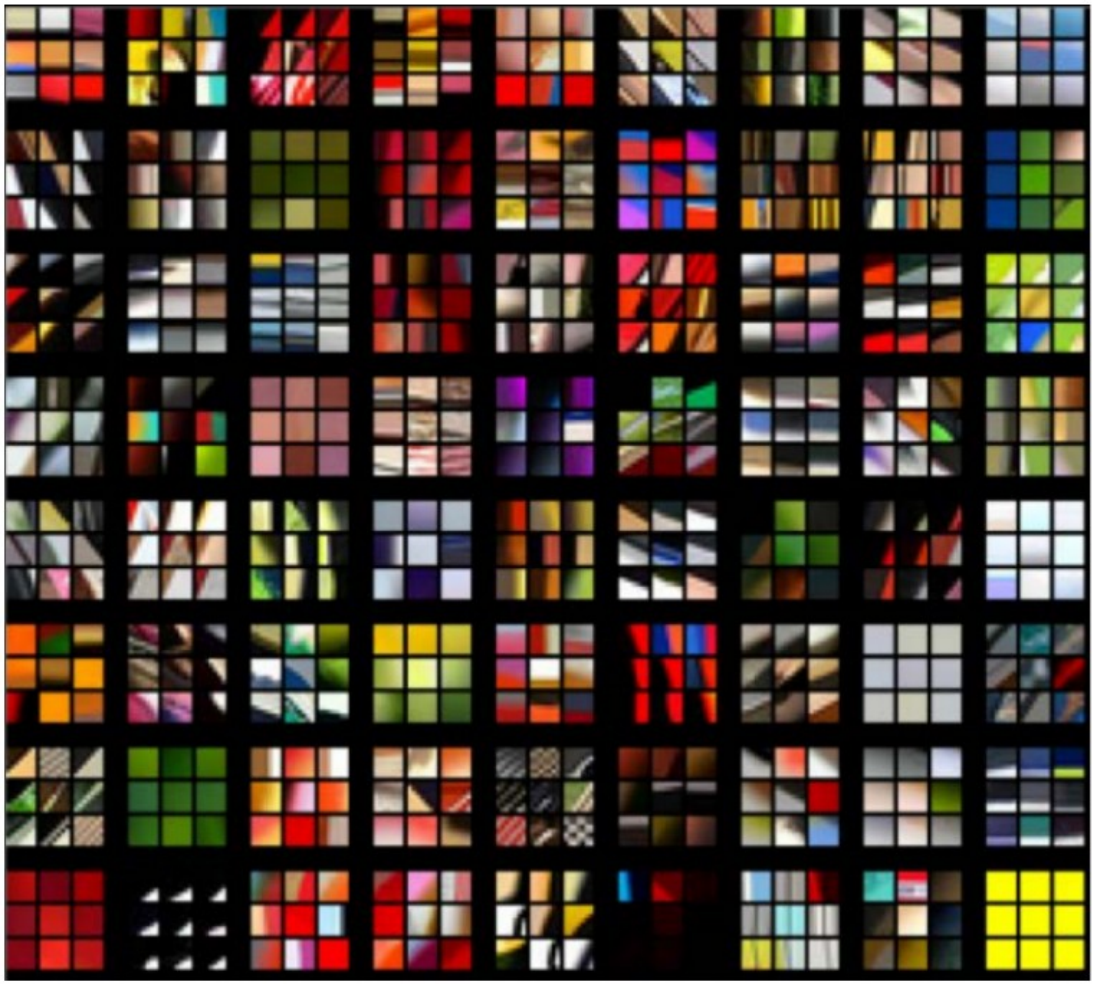


Note these oriented edges are analogous to what SIFT uses to compute the gradients.

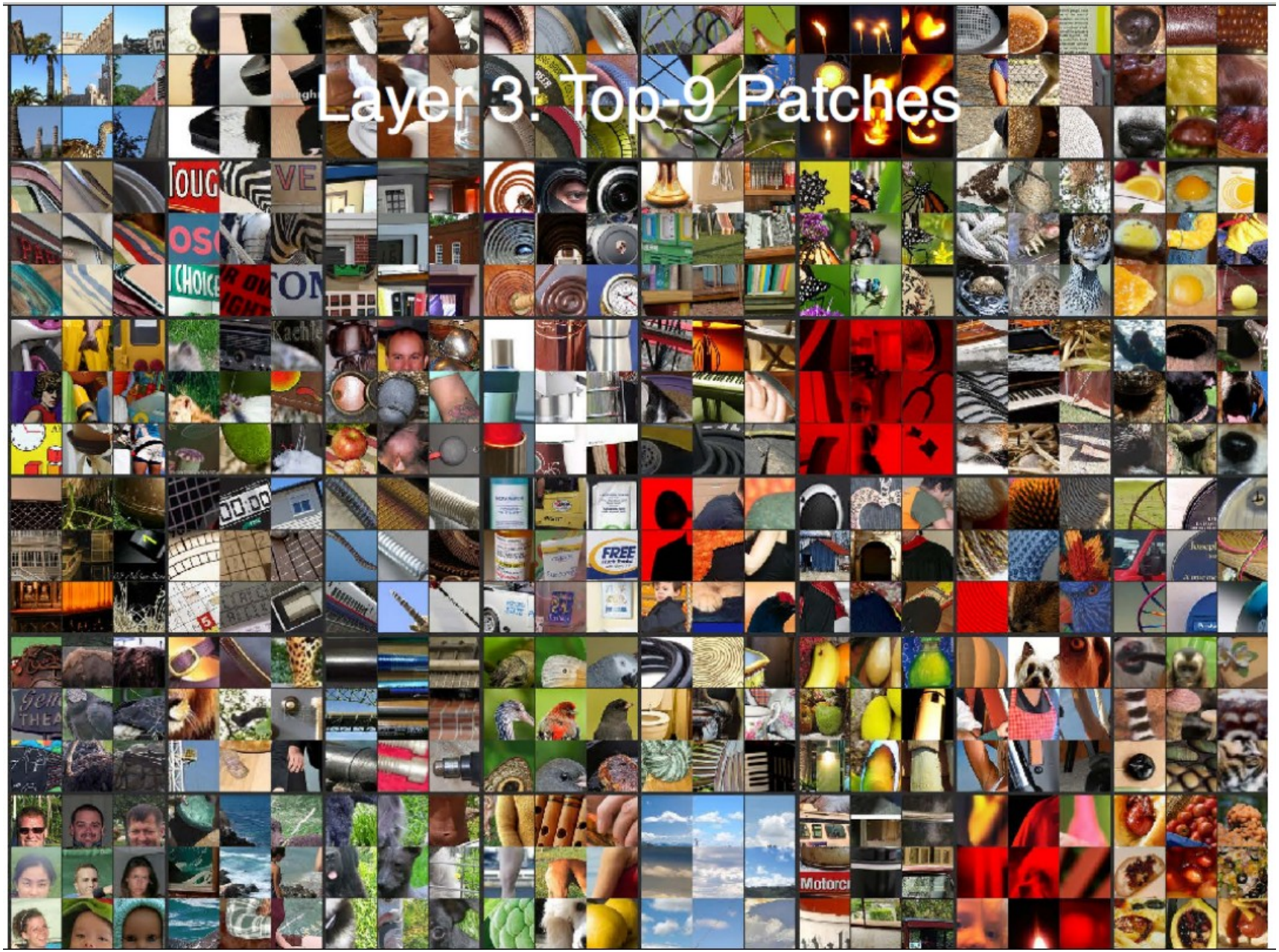
SIFT - scale-invariant feature transform, algorithm published in

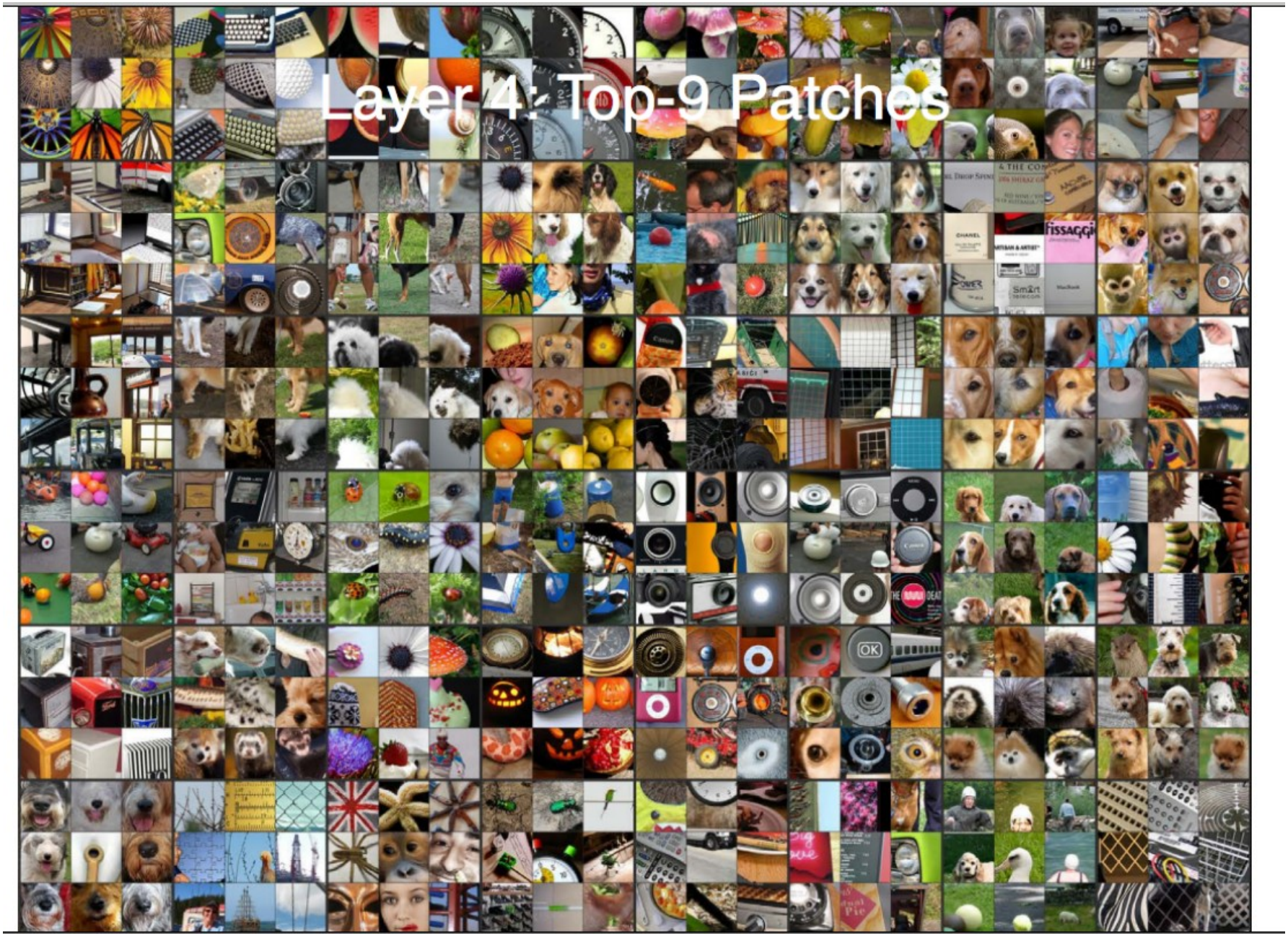
Top 9 patches that activate each filter in layer 1

Each 3x3 block shows the top 9 patches for one filter.




Layer 2: Top-9 Patches


Layer 3: Top-9 Patches


Layer 4: Top-9 Patches


Layer 5: Top-9 Patches

Few properties of Deep Neural Networks

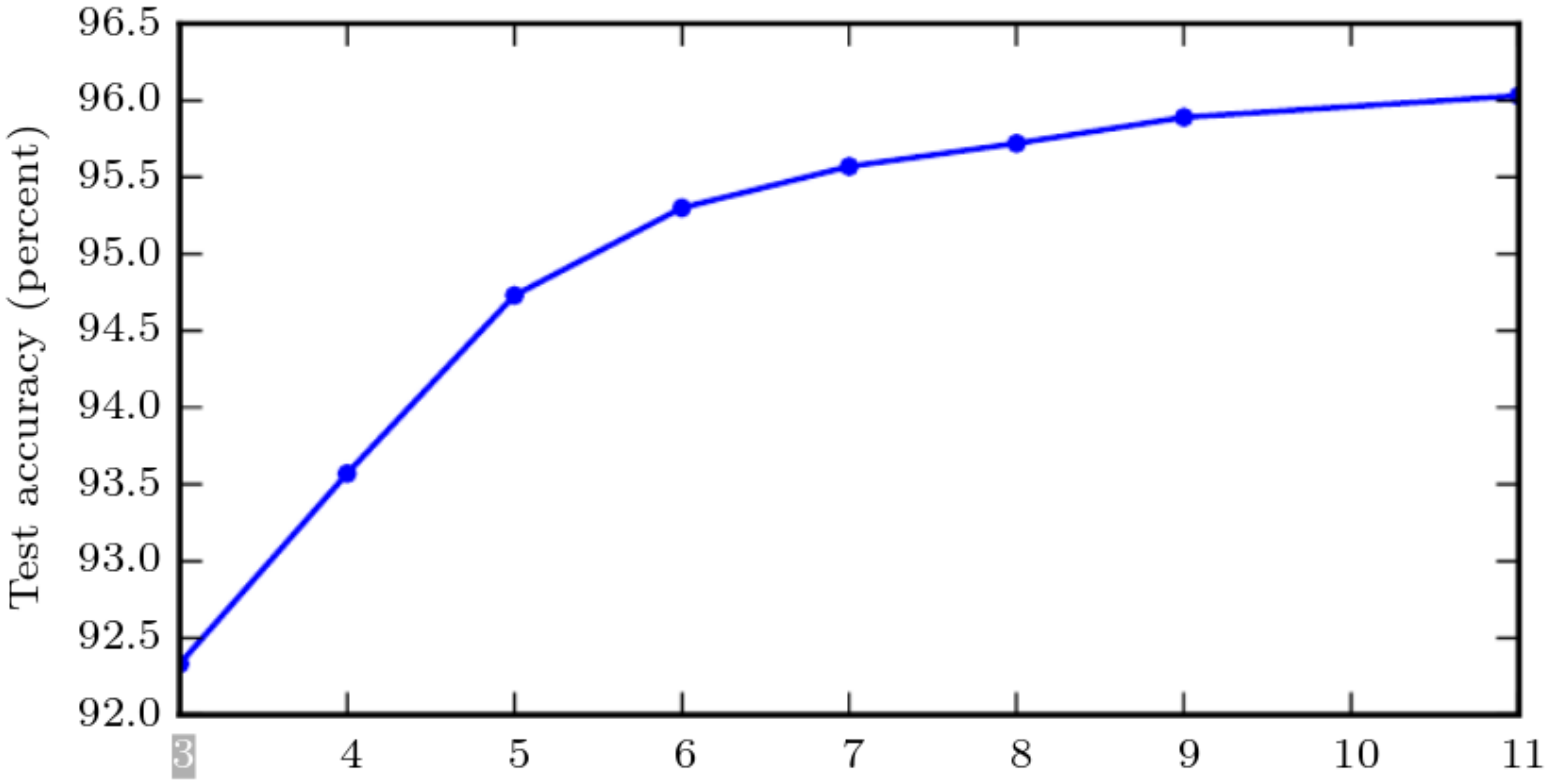


Figure 6.6: Empirical results showing that deeper networks generalize better when used to transcribe multi-digit numbers from photographs of addresses. Data from Goodfellow *et al.* (2014d). The test set accuracy consistently increases with increasing depth. See figure 6.7 for a control experiment demonstrating that other increases to the model size do not yield the same effect.

http://www.deeplearningbook.org

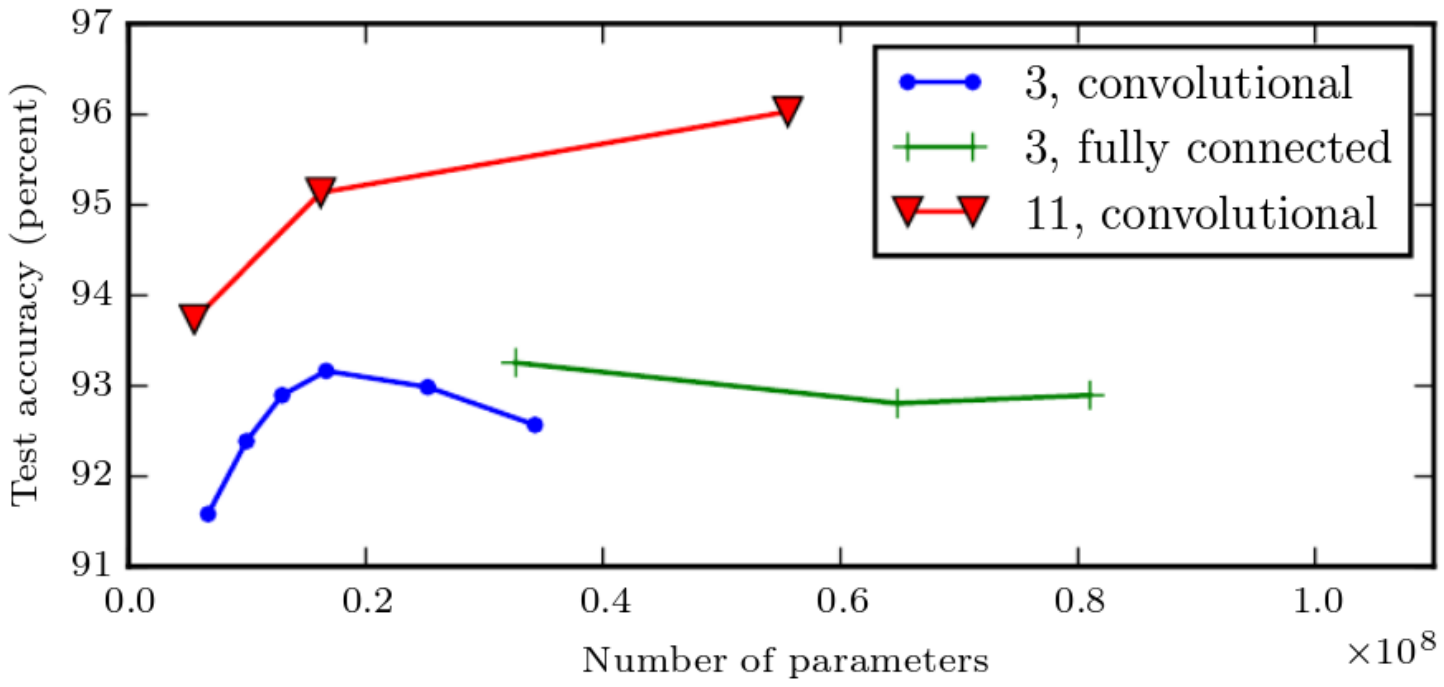


Figure 6.7: Deeper models tend to perform better. This is not merely because the model is larger. This experiment from Goodfellow *et al.* (2014d) shows that increasing the number of parameters in layers of convolutional networks without increasing their depth is not nearly as effective at increasing test set performance. The legend indicates the depth of network used to make each curve and whether the curve represents variation in the size of the convolutional or the fully connected layers. We observe that shallow models in this context overfit at around 20 million parameters while deep ones can benefit from having over 60 million. This suggests that using a deep model expresses a useful preference over the space of functions the model can learn. Specifically, it expresses a belief that the function should consist of many simpler functions composed together. This could result either in learning a representation that is composed in turn of simpler representations (e.g., corners defined in terms of edges) or in learning a program with sequentially dependent steps (e.g., first locate a set of objects, then segment them from each other, then recognize them).

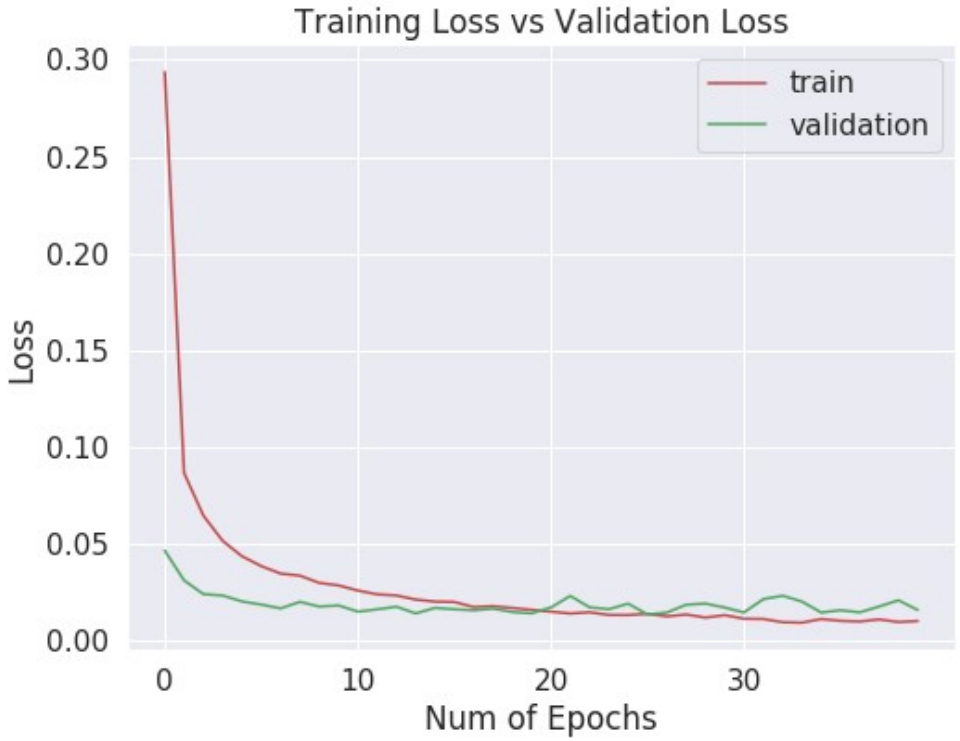
http://www.deeplearningbook.org

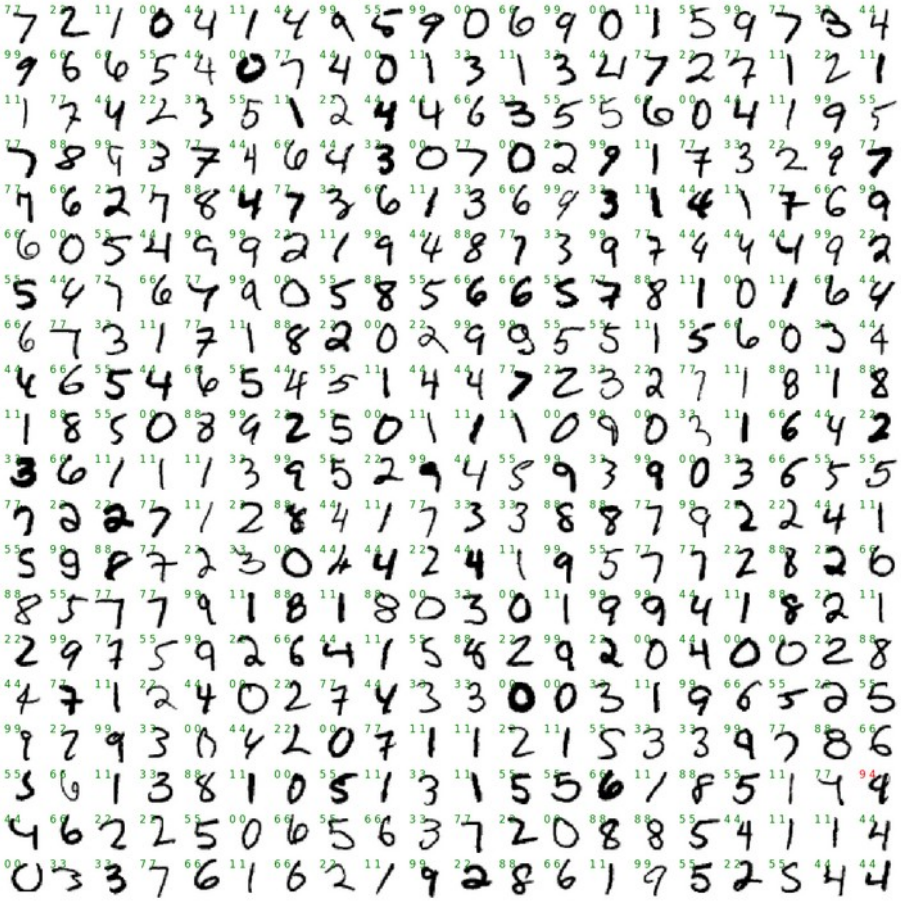
CNN Example

- Back to our digits recognition with CNN…
- Gets much, much better results! Think why?

```
Test loss:     0.0160
Test accuracy: 0.9959
```

https://github.com/marcinwolter/MachineLearnin2019/blob/master/mnist_cnn.ipynb





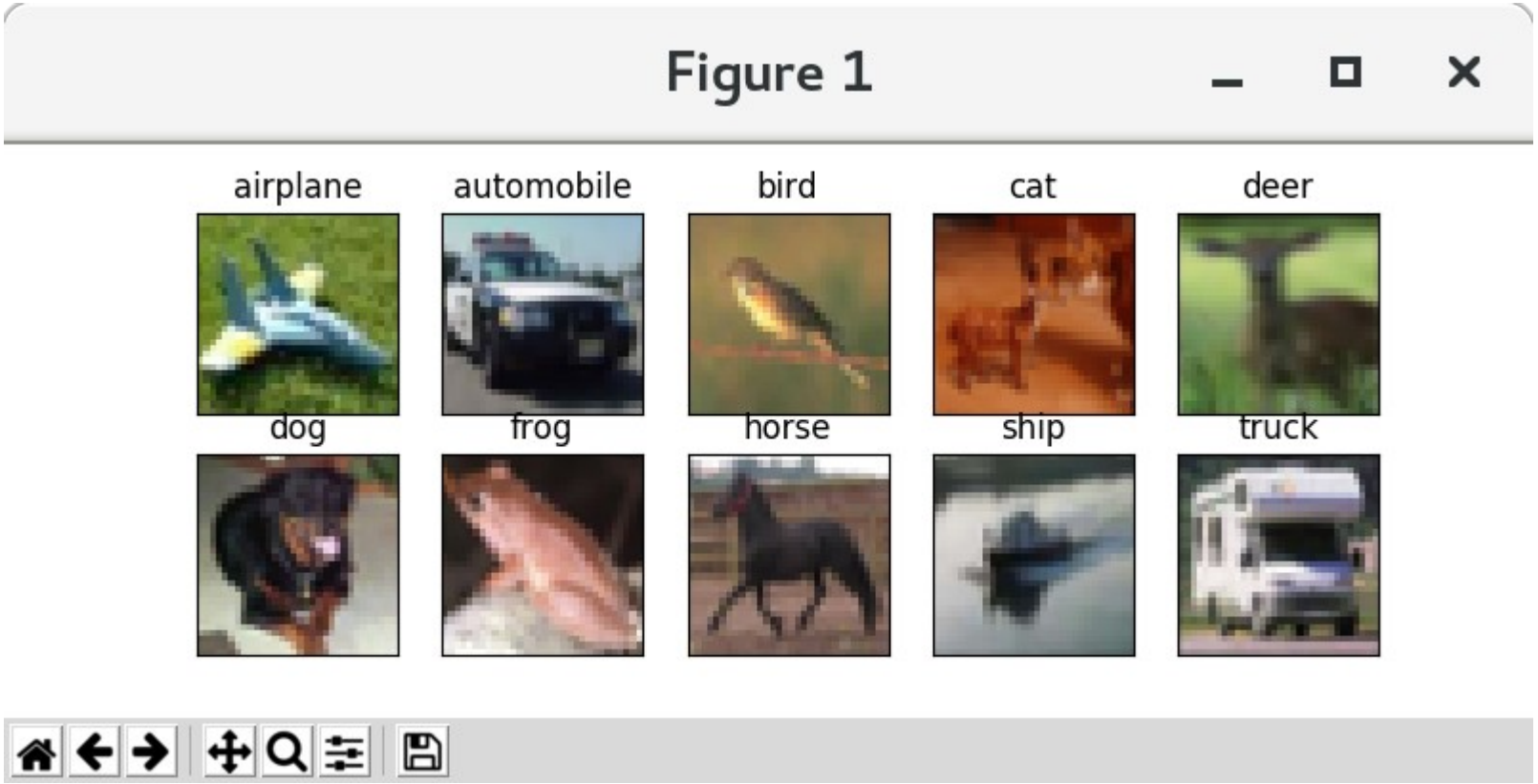
A Deep Neural Network Applet

Applet showing the performance of deep NN:

http://cs.stanford.edu/people/karpathy/convnetjs/

An example – pattern recognition with KERAS and TensorFlow

- CIFAR10 small image classification. Dataset of 50,000 32x32 color training images, labeled over 10 categories, and 10,000 test images.



https://github.com/marcinwolter/DNN_examples/blob/master/cifar_classifier_gpu_aug.ipynb

Deep Neural Network

```
=================================================================
Total params: 1,250,858
Trainable params: 1,250,858
Non-trainable params: 0
_________________________________________________________________
           OPERATION           DATA DIMENSIONS   WEIGHTS(N)   WEIGHTS(%)

               Input   #####      3   32   32
              Conv2D    \|/  -------------------       896     0.0%
                relu   #####     32   32   32
              Conv2D    \|/  -------------------      9248     0.0%
                relu   #####     32   30   30
         MaxPooling2D   Y max -------------------         0     0.0%
                        #####     32   15   15
             Dropout    | || -------------------         0     0.0%
                        #####     32   15   15
              Conv2D    \|/  -------------------     18496     1.0%
                relu   #####     64   15   15
              Conv2D    \|/  -------------------     36928     2.0%
                relu   #####     64   13   13
         MaxPooling2D   Y max -------------------         0     0.0%
                        #####     64    6    6
             Dropout    | || -------------------         0     0.0%
                        #####     64    6    6
             Flatten   ||||| -------------------         0     0.0%
                        #####         2304
               Dense   XXXXX -------------------   1180160    94.0%
                relu   #####          512
             Dropout    | || -------------------         0     0.0%
                        #####          512
               Dense   XXXXX -------------------      5130     0.0%
             softmax   #####           10
Train on 50000 samples, validate on 10000 samples
```

Results


Recognized as
Really was
cat cat
ship ship
airplane ship
airplane airplane
frog frog
frog frog
automobile automobile
frog frog
cat cat
truck automobile
airplane airplane
truck truck
dog dog
horse horse
truck truck
frog ship
dog dog
horse horse
ship ship
frog frog
horse horse
airplane airplane
deer deer
truck truck
deer dog
frog bird
deer deer
airplane airplane
truck truck
frog frog
frog frog
bird dog
bird deer
dog dog
truck truck
cat bird
horse deer
automobile automobile
truck truck
dog dog
deer deer
frog frog
dog dog
frog frog
airplane airplane
truck truck
cat cat
cat truck
horse horse
frog frog
truck truck
ship ship
frog airplane
cat cat
ship ship
ship ship
horse horse
cat horse
dog deer
cat frog
horse horse
cat cat
frog frog
airplane cat
frog frog
bird bird
automobile automobile
bird bird
dog cat
horse horse
bird bird
frog frog
ship ship
ship ship
airplane airplane
bird bird
airplane truck
cat cat
cat cat
ship ship
ship ship
automobile automobile
automobile automobile
horse horse
bird bird
horse dog
bird bird
ship horse
ship ship
truck truck
airplane airplane
cat cat
ship ship
frog frog
deer deer
frog frog
frog frog
bird airplane
bird airplane
dog horse

Summary

Can you design a Deep Neural Network in Keras?