An Introduction to Convolutional Neural Networks

Alessandro Giusti
Dalle Molle Institute for Artificial Intelligence
Lugano, Switzerland
Sources & Resources

- Andrej Karpathy, CS231n
  http://cs231n.github.io/convolutional-networks/
- Ian Goodfellow et al., Deep Learning
  http://www.deeplearningbook.org/
- F. Chollet, Deep Learning with Python
  https://www.manning.com/books/deep-learning-with-python
- Jonathan Hui, CNN Tutorial
  https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/
- Tim Demetters, Understanding Convolutions
  http://timdettmers.com/2015/03/26/convolution-deep-learning/

Some images in this presentation are extracted from the sources listed above
What is a convolution?
What is a convolution?
How does a convolution look like?

1 input map  1 3x3 kernel  1 output map

Response map, quantifying the presence of the filter’s pattern at different locations
What about multiple maps?

1 input map → 1 3x3 kernel → 1 output map
1 input map  2 3x3 kernels  2 output maps
3 input maps  3x2 3x3 kernels  2 output maps
3 input maps    3x2 3x3 kernels    2 output maps
3 input maps  3x2 3x3 kernels  2 output maps
3 input maps  3x2 3x3 kernels  2 output maps

Quiz: how many parameters does this layer have?
3 input maps  3x2 3x3 kernels  2 output maps
= 54 ...
3 input maps 3x2 3x3 kernels 2 output maps
= 54 ... + 2 biases
3 input maps     3x2 3x3 kernels   2 output maps
= 54 ... + 2 biases
= 56 trainable parameters (weights)
Details: padding

How many $3x3$ patches are fully contained in a $5x5$ map?
Details: padding

9: the output map is 3x3
Details: padding

• This is known as “valid” padding mode (default)
• An alternative pads the input map with zeros to yield a same-sized map
Details: striding

- Stride 1x1 is most frequently used: shift 1 pixel at a time → patches are heavily overlapping
- Stride 2x2 skips one patch horizontally and vertically
Why convolutional layers?

• Sparse connectivity
• Parameter sharing
• Translation invariance
Sparse connectivity

Fully connected

3x1 convolutional
Sparse connectivity

Fully connected

3x1 convolutional
Receptive fields

Fully connected

3x1 convolutional
Receptive fields

Deeper neurons depend on wider patches of the input

3x1 convolutional

3x1 convolutional
Parameter sharing

Fully connected
5x5 = 25 weights
(+ 5 bias)

3x1 convolutional
3 weights!
(+ 1 bias)

Quiz: how many parameters does this layer have?
Translational invariance
Max pooling layers ... on many maps?

Quiz: how many parameters does this layer have?
Max pooling downsamples activation maps
Exercise

Input:

Conv 3x3
Conv 3x3
Conv 3x3
Conv 3x3
Conv 3x3
Conv 3x3

map

Conv 3x3
MP 2x2
Conv 3x3
MP 2x2
Conv 3x3
MP 2x2
Receptive fields

Input

Conv 3x3  Conv 3x3  Conv 3x3  Conv 3x3  Conv 3x3  Conv 3x3

map

Conv 3x3  MP 2x2  Conv 3x3  MP 2x2  Conv 3x3  MP 2x2
Receptive fields

Input

Conv 3x3
Conv 3x3
Conv 3x3
Conv 3x3
Conv 3x3
Conv 3x3

map

Conv 3x3
Conv 3x3
MP 2x2
Conv 3x3
MP 2x2
Conv 3x3
MP 2x2

How large is the receptive field of the black neuron?
Receptive fields

How large is the receptive field of the black neuron?

13x13

22x22
Why convnets work

Convnets learn a **hierarchy** of **translation-invariant** spatial pattern detectors
What are layers looking for?
Data from a convnet trained on ImageNet
Shallow layers respond to fine, low-level patterns
Intermediate layers ...
Deep layers respond to complex, high-level patterns
Detail: backprop with max pooling

The gradient is only routed through the input pixel that contributes to the output value; e.g.:
Gradient of \( \bullet \) with respect to \( \bullet \) = 0

![Diagram of max pooling with 2x2 filters and stride 2]
A typical architecture

As we move to deeper layers:
• spatial resolution is reduced
• the number of maps increases

We search for higher-level patterns, and don’t care too much about their exact location. There are more high-level patterns than low-level details!
A typical architecture

Extract high-level features from pixel data

Classify

Convolution layers

Fully connected layers

2x2
1024

256x1
1024x1
4096x1
We will manipulate 4D tensors

Images are represented in 4D tensors:
Tensorflow convention:
(samples, height, width, channels)
The software stack

- Keras
- TensorFlow / Theano / CNTK / ...
- CUDA / cuDNN
- BLAS, Eigen
- GPU
- CPU
What is Keras?

• A model-level library, providing high-level building blocks for developing deep-learning models.
• Doesn’t handle low-level operations such as tensor manipulation and differentiation.
• Relies on backends (such as Tensorflow)
• Allows full access to the backend
Why Keras?

Pros:
• Higher level $\rightarrow$ fewer lines of code
• Modular backend $\rightarrow$ not tied to tensorflow
• Way to go if you focus on applications

Cons:
• Not as flexible
• Need more flexibility? Access the backend directly!
More about ConvNets

Alessandro Giusti
Dalle Molle Institute for Artificial Intelligence
Lugano, Switzerland
1. UNDERFITTING & OVERFITTING ON OUR ROCK PAPER SCISSORS NET
def makeModel(nb_filters):
    model = Sequential()
    model.add(Conv2D(nb_filters, kernel_size, input_shape=(patchsize,patchsize,3), padding = "same"))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size = pool_size))
    model.add(Conv2D(nb_filters*2, kernel_size, padding = "same"))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size = pool_size))
    model.add(Conv2D(nb_filters*4, kernel_size, padding = "same"))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size = pool_size))
    model.add(AveragePooling2D(pool_size = pool_size))
    model.add(Flatten())
    model.add(Dense(128)) # generate a fully connected layer wiht 128 outputs (arbitrary value)
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(3)) # output layer
    model.add(Activation('softmax'))
# Train many models

```python
for filters in [1,2,4,8,16,32,48,64,96]:
    modelid = "filters{:03d}_timestamp{}".format(filters,time.strftime("%Y%m%d%H%M%S"))

    callbacks_list = [
        keras.callbacks.EarlyStopping(
            monitor='val_acc',
            patience=50),
        keras.callbacks.ModelCheckpoint(
            filepath='model_checkpoint_best_{}.h5'.format(modelid),
            monitor='val loss',
            save_best_only=True),
        keras.callbacks.TensorBoard(
            log_dir='./logs/' + modelid,
            histogram_freq=0, write_graph=False, write_images=False)
    ]

    model = makeModel(filters)
    print(model.summary())
    print(model.count_params())

    history=model.fit_generator(
        generator(dataset_tr, batch_size, patchsize),
        steps_per_epoch=50,
        epochs=5000,
        verbose=1,
        validation_data=(X_test,y_test),
        callbacks=callbacks_list)
```
<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv2d_1 (Conv2D)</td>
<td>(None, 32, 32, 1)</td>
<td>28</td>
</tr>
<tr>
<td>activation_1 (Activation)</td>
<td>(None, 32, 32, 1)</td>
<td>0</td>
</tr>
<tr>
<td>max_pooling2d_1 (MaxPooling2)</td>
<td>(None, 16, 16, 1)</td>
<td>0</td>
</tr>
<tr>
<td>conv2d_2 (Conv2D)</td>
<td>(None, 16, 16, 2)</td>
<td>20</td>
</tr>
<tr>
<td>activation_2 (Activation)</td>
<td>(None, 16, 16, 2)</td>
<td>0</td>
</tr>
<tr>
<td>max_pooling2d_2 (MaxPooling2)</td>
<td>(None, 8, 8, 2)</td>
<td>0</td>
</tr>
<tr>
<td>conv2d_3 (Conv2D)</td>
<td>(None, 8, 8, 4)</td>
<td>76</td>
</tr>
<tr>
<td>activation_3 (Activation)</td>
<td>(None, 8, 8, 4)</td>
<td>0</td>
</tr>
<tr>
<td>max_pooling2d_3 (MaxPooling2)</td>
<td>(None, 4, 4, 4)</td>
<td>0</td>
</tr>
<tr>
<td>average_pooling2d_1 (Average)</td>
<td>(None, 2, 2, 4)</td>
<td>0</td>
</tr>
<tr>
<td>flatten_1 (Flatten)</td>
<td>(None, 16)</td>
<td>0</td>
</tr>
<tr>
<td>dense_1 (Dense)</td>
<td>(None, 128)</td>
<td>2176</td>
</tr>
<tr>
<td>activation_4 (Activation)</td>
<td>(None, 128)</td>
<td>0</td>
</tr>
<tr>
<td>dropout_1 (Dropout)</td>
<td>(None, 128)</td>
<td>0</td>
</tr>
<tr>
<td>dense_2 (Dense)</td>
<td>(None, 3)</td>
<td>387</td>
</tr>
<tr>
<td>activation_5 (Activation)</td>
<td>(None, 3)</td>
<td>0</td>
</tr>
</tbody>
</table>

Total params: 2,687
Trainable params: 2,687
Non-trainable params: 0
<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv2d_4 (Conv2D)</td>
<td>(None, 32, 32, 16)</td>
<td>448</td>
</tr>
<tr>
<td>activation_6 (Activation)</td>
<td>(None, 32, 32, 16)</td>
<td>0</td>
</tr>
<tr>
<td>max_pooling2d_4 (MaxPooling2)</td>
<td>(None, 16, 16, 16)</td>
<td>0</td>
</tr>
<tr>
<td>conv2d_5 (Conv2D)</td>
<td>(None, 16, 16, 32)</td>
<td>4640</td>
</tr>
<tr>
<td>activation_7 (Activation)</td>
<td>(None, 16, 16, 32)</td>
<td>0</td>
</tr>
<tr>
<td>max_pooling2d_5 (MaxPooling2)</td>
<td>(None, 8, 8, 32)</td>
<td>0</td>
</tr>
<tr>
<td>conv2d_6 (Conv2D)</td>
<td>(None, 8, 8, 64)</td>
<td>18496</td>
</tr>
<tr>
<td>activation_8 (Activation)</td>
<td>(None, 8, 8, 64)</td>
<td>0</td>
</tr>
<tr>
<td>max_pooling2d_6 (MaxPooling2)</td>
<td>(None, 4, 4, 64)</td>
<td>0</td>
</tr>
<tr>
<td>average_pooling2d_2 (Average)</td>
<td>(None, 2, 2, 64)</td>
<td>0</td>
</tr>
<tr>
<td>flatten_2 (Flatten)</td>
<td>(None, 256)</td>
<td>0</td>
</tr>
<tr>
<td>dense_3 (Dense)</td>
<td>(None, 128)</td>
<td>32896</td>
</tr>
<tr>
<td>activation_9 (Activation)</td>
<td>(None, 128)</td>
<td>0</td>
</tr>
<tr>
<td>dropout_2 (Dropout)</td>
<td>(None, 128)</td>
<td>0</td>
</tr>
<tr>
<td>dense_4 (Dense)</td>
<td>(None, 3)</td>
<td>387</td>
</tr>
<tr>
<td>activation_10 (Activation)</td>
<td>(None, 3)</td>
<td>0</td>
</tr>
</tbody>
</table>

Total params: 56,867  
Trainable params: 56,867  
Non-trainable params: 0
<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv2d_7 (Conv2D)</td>
<td>(None, 32, 32, 128)</td>
<td>3584</td>
</tr>
<tr>
<td>activation_11 (Activation)</td>
<td>(None, 32, 32, 128)</td>
<td>0</td>
</tr>
<tr>
<td>max_pooling2d_7 (MaxPooling2)</td>
<td>(None, 16, 16, 128)</td>
<td>0</td>
</tr>
<tr>
<td>conv2d_8 (Conv2D)</td>
<td>(None, 16, 16, 256)</td>
<td>295168</td>
</tr>
<tr>
<td>activation_12 (Activation)</td>
<td>(None, 16, 16, 256)</td>
<td>0</td>
</tr>
<tr>
<td>max_pooling2d_8 (MaxPooling2)</td>
<td>(None, 8, 8, 256)</td>
<td>0</td>
</tr>
<tr>
<td>conv2d_9 (Conv2D)</td>
<td>(None, 8, 8, 512)</td>
<td>1180160</td>
</tr>
<tr>
<td>activation_13 (Activation)</td>
<td>(None, 8, 8, 512)</td>
<td>0</td>
</tr>
<tr>
<td>max_pooling2d_9 (MaxPooling2)</td>
<td>(None, 4, 4, 512)</td>
<td>0</td>
</tr>
<tr>
<td>average_pooling2d_3 (Average)</td>
<td>(None, 2, 2, 512)</td>
<td>0</td>
</tr>
<tr>
<td>flatten_3 (Flatten)</td>
<td>(None, 2048)</td>
<td>0</td>
</tr>
<tr>
<td>dense_5 (Dense)</td>
<td>(None, 128)</td>
<td>262272</td>
</tr>
<tr>
<td>activation_14 (Activation)</td>
<td>(None, 128)</td>
<td>0</td>
</tr>
<tr>
<td>dropout_3 (Dropout)</td>
<td>(None, 128)</td>
<td>0</td>
</tr>
<tr>
<td>dense_6 (Dense)</td>
<td>(None, 3)</td>
<td>387</td>
</tr>
<tr>
<td>activation_15 (Activation)</td>
<td>(None, 3)</td>
<td>0</td>
</tr>
</tbody>
</table>

Total params: 1,741,571
Trainable params: 1,741,571
Non-trainable params: 0
- Define the problem at hand and the data on which you’ll train. Collect this data, or annotate it with labels if need be.
- Choose how you’ll measure success on your problem. Which metrics will you monitor on your validation data?
- Determine your evaluation protocol: hold-out validation? K-fold validation? Which portion of the data should you use for validation?
- Develop a first model that does better than a basic baseline: a model with statistical power.
- Develop a model that overfits.
- Regularize your model and tune its hyperparameters, based on performance on the validation data. A lot of machine-learning research tends to focus only on this step—but keep the big picture in mind.
2. VISUALIZATION TECHNIQUES
Visualizing the weights of the net

We want to see this
Visualizing the weights of the net

11x11x3 filters (visualized in RGB) in the first convolutional layers
Visualizing the activations of intermediate layers

We want to see this
Visualizing the activations of intermediate layers
Visualizing the input that maximally activates some neurons

We want to compute (and see) the input that maximally activates this guy
Step 1

Compute the gradient of this with respect to the input.
Step 2

Nudge the input accordingly: our guy will increase its activation
Goto step 1
Shallow layers respond to fine, low-level patterns
Intermediate layers ...
Deep layers respond to complex, high-level patterns
Stand on the shoulder of giants

**USING PRETRAINED WEIGHTS**
Using pretrained weights

Step 1

Prediction

Trained classifier

Trained convolutional base

Input
Option 1

Input

Conv MP Conv MP Conv MP Flatten Dense Dense Outputs
Option 1

Then, train a new classifier that uses these features as input.
Option 3

Input

Freeze
- Conv
- MP
- Conv
- MP

Finetune
- Conv
- MP
- Flatten
- Dense
- Dense

Train only

Outputs
A MILE-HIGH OVERVIEW OF FULLY
CONVOLUTIONAL NETWORKS FOR
SEGMENTATION
Overall idea
“Convolutionalization” of a dense layer
SOME POSSIBLE PROJECTS
Deep Learning on vibration data for detecting fence violations
Deep Learning on wearable sensor data for robot control

Landing a Drone with Pointing Gestures

Boris Gromov, Luca M. Gambardella, Alessandro Giusti

Dalle Molle Institute for Artificial Intelligence (IDSIA)
Lugano, Switzerland
Learning to predict errors in weather forecasts