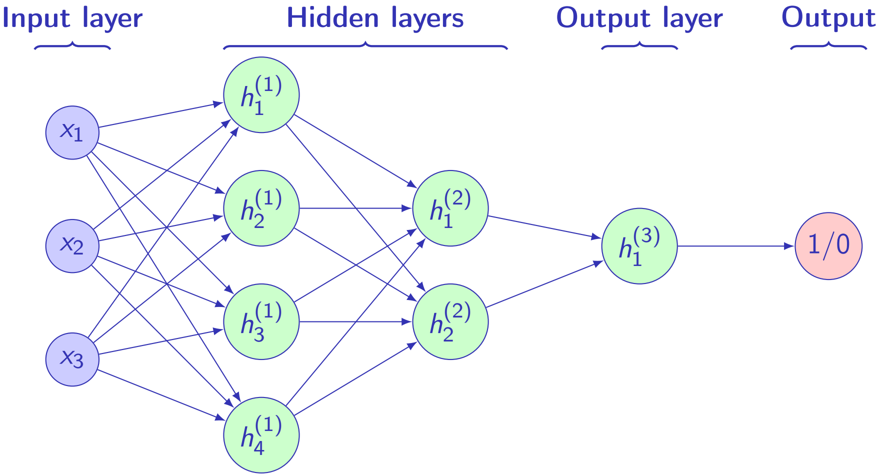
EDAN95
Applied Machine Learning
Lecture 5: Convolutional Networks

Pierre Nugues

Pierre.Nugues@cs.lth.se
http://cs.lth.se/pierre_nugues/

November 16, 2020

Classifying Images with a Multilayer Perceptron



Each input node corresponds to a pixel of the input image.

Code Example

Jupyter Notebook from Chollet's book:
`2.1-a-first-look-at-a-neural-network.ipynb`

- Although efficient on digits, multilayer perceptron or logistic regression cannot capture optimally patterns in images.
- It is now replaced by networks that embed a pattern-extraction mechanism.

The Origins: The Convolution

The product of a function and a moving window, called the kernel.
Mathematical definition:

$$(f * g)(x) = \int_{-\infty}^{+\infty} f(x-t)g(t)\,\mathrm{d}t,$$

where f is the function and g, the convolution kernel
Notice that one of the function, here f, is reversed and shifted to guarantee commutativity
In the discrete case, we have:

$$(f * g)(i) = \sum_{j=-\infty}^{\infty} f(i-j)g(j)$$

It can be extended to two dimensions.

Convolution in Image Processing

Convolution is used extensively in pattern recognition to implement spatial filtering.

In image processing, f is an image and g a small window, most frequently its dimensions are: (3, 3) or (5, 5).

For a kernel of dimensions (M, N), normally odd numbers, we have:

$$(f * g)(x,y) = \sum_{i=-M/2}^{M/2} \sum_{j=-N/2}^{N/2} f(x-i, y-j) g(i,j),$$

where g is centered at 0.

Example of a Convolution

A blurring kernel, normally normalized by its sum (1/9):

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \mathbf{219} & 253 & 247 \\ 0 & 0 & 190 & 0 \\ 0 & 0 & 0 & 93 \\ 0 & 0 & 221 & 253 \\ 136 & 212 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \mathbf{662} & & 0 \\ 0 & & & 0 \\ 0 & & \mathbf{757} & 0 \\ 0 & & & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$219 + 253 + 190 = 662$

Borders can either be padded or ignored. In the latter case, the kernel must always fit in the image and the output image has a reduced size.

(Complete the matrix...)

Spatial Filters

The kernels enable us to create filters, for instance to smooth or sharpen the image.
The Sobel operator is a popular edge detector. It corresponds to the gradient norm of the input image and sharpens the edges.
We compute the x and y derivatives using two kernels:

$$\mathrm{G}_x = \mathrm{I} * \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}; \mathrm{G}_y = \mathrm{I} * \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$\mathrm{G} = \sqrt{\mathrm{G}_x^2 + \mathrm{G}_y^2}$$

We can also compute the gradient angle:

$$\tan\theta = \frac{\mathrm{G}_y}{\mathrm{G}_x}; \theta = \arctan\frac{\mathrm{G}_y}{\mathrm{G}_x}$$

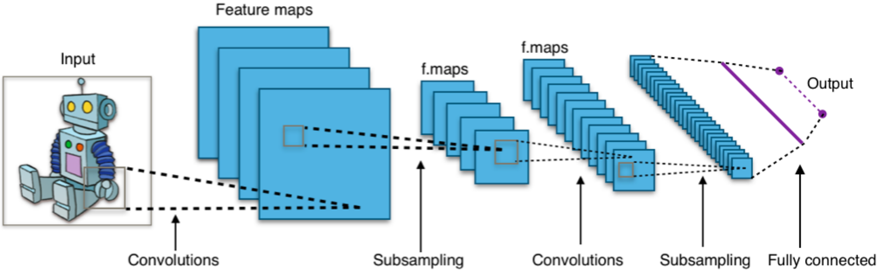
Code Example

Jupyter Notebook: `2.1-convolution.ipynb`

Architecture of a Convolutional Neural Network (CNN)

A pipeline of convolution and subsampling operations:

1. In the figure, the network will learn four kernels in the first layer;
2. in the second layer, it will subsample the images, keep one pixel every four pixels, for instance.



Credits: Wikipedia

Subsampling

Subsampling is generally carried out using the max-pooling operation:

1. We partition the image into squared tiles of size (2, 2) or (3, 3);
2. We reduce each tile to one value: the max value in the tile.

For example, using a tile of size (2, 2), we have:

$$\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 0 & \mathbf{219} & \mathbf{253} & 247 \\ 0 & 0 & \mathbf{190} & 0 \\ 0 & 0 & 0 & 93 \\ 0 & 0 & 221 & \mathbf{253} \\ 136 & \mathbf{212} & 0 & 0 \end{array} \rightarrow \begin{array}{cc} 219 & 253 \\ 0 & 190 \\ 212 & 253 \end{array}$$

We have reduced the image size from (6, 4) to (3, 2).
In addition, it makes the images invariant to small rotations and translations

A Small Convolutional Network

Encoding a CNN is straightforward in Keras:

1. We declare a sequential model
2. We add the layers

From Chollet, Listing 5.1, a small network with three convolutional layers and two subsampling layers:

```
from keras import layers
from keras import models

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

Understanding the Parameters

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 64)          36928
=================================================================
```

First layer: 32 kernels of size (3, 3) and an intercept:
$(3 \times 3 + 1) \times 32 = 320$.
The output consists of 32 images, where the size is reduced by two so that the kernel fits in the image
The downsampling reduces the images to (13, 13)
The third layer has $(3 \times 3 \times 32 + 1) \times 64 = 18496$ parameters

Adding a Classifier

The output the the convolutions consists of 64 (3, 3) images. We need to flatten them.
The rest is just a classifier with 10 outputs:

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

Code Example

Jupyter Notebook: Chollet 5.1 `https://github.com/fchollet/deep-learning-with-python-notebooks`

A Larger Dataset: The cats and dogs from Kaggle

We will follow Chollet again and review how to classify images of cats and dogs drawn from a Kaggle competition.
We will examine three strategies:

1. Raw data set, but smaller than the original one
2. Dataset augmented with image distortions
3. Using a pretrained network

In the next lab, you will apply the same procedure to another dataset from Kaggle and categorize five types of flowers

Raw Dataset

To process a raw dataset, we create a pipeline of convolutional layers and of max-pooling.

1. As for other networks, the activation is the relu function except the last layer, which uses a logistic function.
2. As a general rule, the output images, feature maps, are smaller, but their number increases.
3. As architecture, Chollet proposed 4 convolutional layers and 4 subsampling layers, and a final classifier.
4. The images are supplied by a generator.
5. As with all datasets, you must rescale your data

Generator

A construct to build sequences (iterators) with a minimal memory footprint
Compare:

```
# A list
a = [i for i in range(10000000)]
print(sys.getsizeof(a))
```

8 times the number of items.
And

```
# A generator
b = (i for i in range(10000000))
print(sys.getsizeof(b))
```

A very small size
But you can traverse it only once

Code Example

Jupyter Notebook: `2.2-generators.ipynb`

Image Processing

Keras has a builtin module to read images from a folder, process them, for instance, rescale them, and supply them to the network using a generator (Chollet, Listing 5.7).

```
from keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir, # This is the target directory
        target_size=(150, 150), # All images will be resized to 150x150
        batch_size=20,
        class_mode='binary')  # we use binary_crossentropy, we need binary labels

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

The `train_generator` and `validation_generator` loop endlessly.

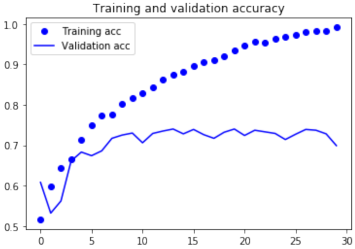
Fitting Function

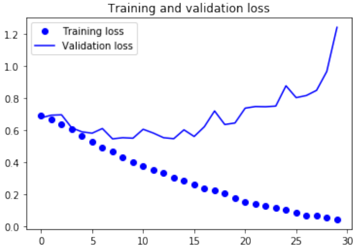
The training procedure must see all the samples:

```
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100, # 20 samples per batch x 100 = 2000
      epochs=100,
      validation_data=validation_generator,
      validation_steps=50)  # 20 samples per batch x 50 = 1000
```

Code Example

Jupyter Notebook: Chollet 5.2 `https://github.com/fchollet/deep-learning-with-python-notebooks`





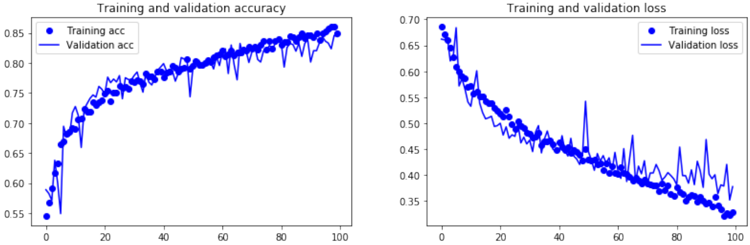
Data Augmentation

Small datasets are prone to overfit: The model fits perfectly the training data, but cannot generalize to other kinds of data.
To avoid overfit, we can "augment" data in the training set with small transformations, for example a rotation.
To make it easier, `ImageDataGenerator()` has a set of predefined random transformations:

```
datagen = ImageDataGenerator(
      rotation_range=40, # A random rotation
      width_shift_range=0.2, # A random shift
      height_shift_range=0.2,
      shear_range=0.2, # shearing
      zoom_range=0.2, # random zoom
      horizontal_flip=True,
      fill_mode='nearest')
```

Code Example

Jupyter Notebook: Chollet 5.2 `https://github.com/fchollet/deep-learning-with-python-notebooks`

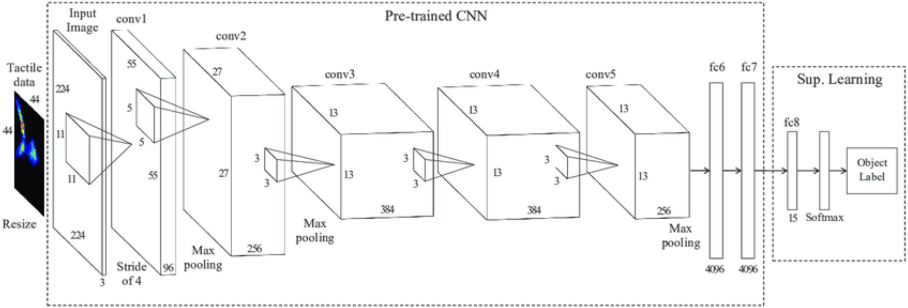


Pretrained Convnet

In our datasets, we have a few thousands images and a handful of classes.
Some groups trained models on millions of images and thousands of classes, and they were kind enough to make their models available.
As for simpler CNN, models learn more and more symbolic patterns as we proceed in the pipeline.
In practice, it can be difficult to train a large network on many images with many classes.
But we can reuse large existing models to extract the patterns.

Architecture of a Pretrained Convnet

This architecture consists of a pretrained convolutional base on which we plug a trainable classifier.



[1]

[1] Juan Manuel Gandarias, Jesus Gomez-de-Gabriel, Alfonso J. García-Cerezo, Enhancing Perception with Tactile Object Recognition in Adaptive Grippers for Human-Robot Interaction, Sensors 18(3):692, February 2018

Training Strategies with a Pretrained Convnet

We either:

1. Use the pretrained convolutional base as input to a classifier. We then train this classifier;
2. Build a new network that consists of the frozen pretrained part as a base and extend it with a top that we train;
3. Train parts of the pipeline that consists of the pretrained convolutional base and a top carrying out the classification.

Extending the Convolutional Base

The second option is very easy with Keras: We just add the base (Chollet, Listing 5.20):

```
from keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(150, 150, 3))

model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

For available pretrained convnets, see `https://keras.io/applications/#documentation-for-individual-models`.

Extracting Features from the Convolutional Base

For the first option, we need to read the prediction from the base (Chollet, Listing 5.17):

```
datagen = ImageDataGenerator(rescale=1./255)
batch_size = 20

def extract_features(directory, sample_count):
    features = np.zeros(shape=(sample_count, 4, 4, 512))
    labels = np.zeros(shape=(sample_count))
    generator = datagen.flow_from_directory(
        directory,
        target_size=(150, 150),
        batch_size=batch_size,
        class_mode='binary')
    i = 0
    for inputs_batch, labels_batch in generator:
        features_batch = conv_base.predict(inputs_batch)
        features[i * batch_size : (i + 1) * batch_size] = features_batch
        labels[i * batch_size : (i + 1) * batch_size] = labels_batch
        i += 1
        if i * batch_size >= sample_count:
            # Note that since generators yield data indefinitely in a loop,
            # we must ‘break‘ after every image has been seen once.
            break
```

Extracting Features from the Convolutional Base

We need to read the prediction from the base (Chollet, Listing 5.18):

```
train_features, train_labels = extract_features(train_dir, 2000)
validation_features, validation_labels = extract_features(validation_dir, 1000)
test_features, test_labels = extract_features(test_dir, 1000)

history = model.fit(train_features, train_labels,
                    epochs=30,
                    batch_size=20,
                    validation_data=(validation_features, validation_labels))
```

Code Example

Jupyter Notebook: Chollet 5.3 `https://github.com/fchollet/deep-learning-with-python-notebooks`