Automatic crack detection - with deep learning

Sigrid Keydana, Trivadis
2017/22/09
Crack? No crack?
One step back: What’s deep learning?
What is a neural network?

Biological neuron and artificial neuron

Source: Stergiou, C. and Siganos, D. Artificial neurons
Prototype of a neuron: the perceptron

Source: Stergiou, C. and Siganos, D. Artificial neurons
Going deep

Source: Stergiou, C. and Siganos, D. Artificial neurons
Why go deep? A bit of background

Easy? Difficult?

- walk
- talk
- play chess
- solve matrix computations
Representación importa

Source: Goodfellow et al. 2016, Deep Learning
Just feed the network the right features?

What are the correct pixel values for a “bike” feature?

- race bike, mountain bike, e-bike?
- pixels in the shadow may be much darker
- what if bike is mostly obscured by rider standing in front?
Let the network pick the features

... a layer at a time

Source: Goodfellow et al. 2016, *Deep Learning*
How does a deep network learn?
Training a deep neural network

We need:

- a way to quantify our current (e.g., classification) error
- a way to reduce error on subsequent iterations
- a way to propagate our improvement logic from the output layer all the way back through the network!
The loss (or cost) function indicates the cost incurred from false prediction / misclassification.

Probably the best-known loss functions in machine learning are **mean squared error**:

\[
\frac{1}{n} \sum_{i=1}^{n} (y_i - y_i^2)
\]

and **cross entropy**:

\[
\sum_{j} t_j \log y_j
\]
Learning from errors: Gradient Descent

Source: Goodfellow et al. 2016, *Deep Learning*
Propagate back errors ... backpropagation!

- basically, just the chain rule: \( \frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx} \)
- chained over several layers:

Source: https://colah.github.io/posts/2015-08-Backprop/
Example domain: Convolutional Neural Networks for Computer Vision
Why computer vision is hard

Figure 1. The **deformable and truncated cat**. Cats exhibit (almost) unconstrained variations in shape and layout. The cat examples shown here are detected by our Distinctive Part Model, but missed by the template based method of [11].

Source: Parkhi et al. *Cats and Dogs*
Tasks in computer vision

Source: Stanford CS231n Convolutional Neural Networks Lecture Notes
Convolutional Neural Networks (CNNs)

Source: http://cs231n.github.io/convolutional-networks/
The Convolution Operation

Back to our cracks!
Let’s build our own CNN, in Keras (using R!)

4 steps

- build model
- prepare data
- train model
- test model
model <- keras_model.Sequential()

model %>%
  layer_conv_2d(
    filter = 32, kernel_size = c(3,3), padding = "same", input_shape = c(target_height, target_width, 3) )
%>%
  layer_activation("relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%

layer_conv_2d(filter = 32, kernel_size = c(3,3)) %>%
layer_activation("relu") %>%
layer_max_pooling_2d(pool_size = c(2,2)) %>%

layer_conv_2d(filter = 64, kernel_size = c(3,3), padding = "same") %>%
layer_activation("relu") %>%
layer_max_pooling_2d(pool_size = c(2,2)) %>%

layer_flatten() %>%
layer_dense(64) %>%
layer_activation("relu") %>%
layer_dropout(0.5) %>%
layer_dense(2) %>%
layer_activation("softmax")

opt <- optimizer_rmsprop(lr = 0.001, decay = 1e-6)

model %>% compile(
  loss = "binary_crossentropy",
  optimizer = opt,
  metrics = "accuracy"
)
How about the data?

- in this case study, we have very little data at our disposition
- can use data augmentation to artificially increase training set size

```r
train_datagen <- image_data_generator(
  rescale = 1/255,
  rotation_range = 80,
  width_shift_range = 0.2,
  height_shift_range = 0.2,
  horizontal_flip = TRUE,
  vertical_flip = TRUE,
  shear_range = 0.2,
  zoom_range = 0.2,
  fill_mode = "wrap"
)
```
Train model

- Ready to resume in a few hours?
- Let's load the trained model instead

```r
model_name <- "model_filter323264_kernel3_epochs20_lr001.h5"
model <- load_model_hdf5(model_name)
```
Test model

- Accuracy (train/test): 0.70 / 0.86
- Recall (train/test): 0.50 / 0.88
- Precision (train/test): 0.870 / 0.85
Let’s look at some predictions (1)

Crack (top row - easy)

class probabilities(crack/no crack): 0.92 / 0.08
Let’s look at some predictions (2)

No crack (top row - easy)

class probabilities(crack/no crack): 0.35 / 0.65
Let’s look at some predictions (3)

Crack (middle row - medium)

class probabilities(crack/no crack): 0.59 / 0.41
Let’s look at some predictions (4)

No crack (middle row - *medium*)

class probabilities(crack/no crack): 0.42 / 0.58
Let’s look at some predictions (5)

Crack (bottom row - *difficult*)

```
class probabilities(crack/no crack): 0.32 / 0.68
```
Let’s look at some predictions (6)

No crack (bottom row - difficult)

class probabilities(crack/no crack): 0.63 / 0.37
How about using a pre-trained model?

- frameworks often already come with models pre-trained on ImageNet (e.g., ResNet, VGG16, InceptionV3...)

- usual workflow
  - instantiate all layers below top-level densely connected layer, and set them to non-trainable
  - put own densely connected layer on top and train the unified model
  - possibly unfreeze a few of the convolutional blocks near the top and try fine-tuning them

```r
base_model <- application_vgg16(weights = 'imagenet', include_top = FALSE)
for (layer in base_model$ layers)
  layer$ trainable <- FALSE
# add our own fully connected layer (with dropout!)
```
Pre-trained model: accuracy

- Accuracy (train/test): 1.0 / 0.83
- Recall (train/test): 1.0 / 0.88
- Precision (train/test): 1.0 / 0.8

... this means?
We need more data!

- for this example, only 7 actual images were used to train the model
- in the real world, for a task like this, we expect *much* more data to be available
- already with this tiny amount of data and a small network trained from scratch, performance is pretty good!
Conclusion

- just one of many possible things you can do with deep learning
- more and more “traditional” machine learning problems are being addressed with DL all the time, with increasing success
- watch out for increasing applications to unsupervised problems!
Finally

- neural networks are less of a black box than one might think
- open source frameworks like Keras, PyTorch or TensorFlow make it easy to try DL for yourself
- how could deep learning apply to your problem domain?
THANK YOU!