Using BigDL with Siamese CNN for Detecting Duplicate Entries In Real Estate Listings Databases

Olga Ermolin, MLS Listings, Inc
Sergey Ermolin, Data Analytics Technologies, Intel Corp
Agenda

- Introductions: Intel and MLSListings
- Business Use-Case
- Data Governance in US Real Estate sector
- Image Processing Pipeline
- Deep Learning Implementation
  - Architecture
  - BigDL code
  - TensorFlow code
- Results and Discussion
BigDL is an open-source distributed deep learning library for Apache Spark* that can run directly on top of existing Spark or Apache Hadoop* clusters.

**Ideal for DL Models TRAINING and INFEERENCE**

- **Dataframe**
  - ML Pipelines: SQL, SparkR, Streaming, MLlib, GraphX, BigDL

**High Performance Deep Learning for Apache Spark**

On CPU Infrastructure

No need to deploy costly accelerators, duplicate data, or suffer through scaling headaches!

**Feature Parity**

with TensorFlow*, Caffe* and Torch*

**Lower TCO and Improved ease of use**

with existing infrastructure

**Deep Learning on Big Data Platform**, Enabling **Efficient Scale-Out**

*Powered by Intel® Math Kernel Library (Intel® MKL) and multi-threaded programming**

software.intel.com/bigdl
MLSListings Inc, Sunnyvale, California

About Us

Located in the heart of Silicon Valley, our internal teams of Engineers, IT Professionals, and Product Managers are complemented by Professional Services staff who provide strategic business planning and implementation.
Overview – problem statement

- 600+ MLS in USA
- 7 just in San Francisco Bay Area in California
- Overlapping boarders
- To maximize exposure, agents enter listings in several MLSs, especially where borders are overlapping
- This results in Duplicate listings. Need for de-duping.
- Websites powered by aggregated databases have supplicates
- De-Duping is important
How can de-duping be done?

- De-duping..
- Can be done based on transactional data.
  - Same address - but variations of spelling
  - Same parcel number
  - Same Listing Agent
  - Same date when came to market
  - Same price

- What if necessary data is not available or unreliable?
  - Monterey, CA
De-duping based on image similarity

- One way to do it - by image analysis using **transfer learning** to identify duplicate images.
- More often than not, images belonging to the same property are not identical:
  - Watermarked by entry system
  - Cropped due to entry system’s requirements
  - Rotated due to user error
  - Taken from different angle
- To train a model to recognize duplicates, we need to create a large labeled training set.
Solution Architecture – sneak preview

- “Siamese” Architecture
  - First publications ~ 2005
  - Matured solution ~2015
  - Identical (“Siamese”) CNNs are used side-by-side.

- Images are converted to feature vectors
- L1 or L2 difference between output vectors
- Compare with a-priory binary labels
- Train the network
<10% of real-life images are duplicates

For training need more images, balanced dataset (50%)

The code prepares a labeled training set of images for supervised learning. It crops, scales, watermarks, and rotates images creating “duplicates”

- Input: as set of unique property images
- Outputs:
  - Directory of images (originals plus modified “duplicates”)
  - Annotation *.csv file: filename pairs plus a True/False flag indicating dup or not
Utilizing Python Pillow library and itertools

Take input of N images. For each create 3 modified copies:
- Watermarked, with randomized location of the WM
- Cropped copy, randomize the size (between 85-95%)
- Rotated, randomized the angle
- Create a file listing all possible pairs of images with a flag indicating if they are duplicate or not.
# Image pipeline script - output

<table>
<thead>
<tr>
<th>File name</th>
<th>File name</th>
<th>Dup Y/N</th>
</tr>
</thead>
<tbody>
<tr>
<td>File_1</td>
<td>File_2</td>
<td>0</td>
</tr>
<tr>
<td>File_1</td>
<td>File_1_WM</td>
<td>1</td>
</tr>
<tr>
<td>File_2</td>
<td>File_1_ROT</td>
<td>0</td>
</tr>
</tbody>
</table>
for file in filelist:
    # catch possible exceptions per each file, so that we can continue to the next one
    try:
        current_file_path = MyImage(r"", file)
        current_file_path.rotate()
        current_file_path.crop()
        current_file_path.watermark()
    except Exception:
        # just catch it and move on to the next image
        pass

from PIL import Image, ImageDraw, ImageFont
import os
import glob
import random
import csv
import itertools
import time

def rotate(self):
    try:
        temp_image = self.image.rotate(self._rand_rotate_angle(), expand=True)
        # saving modified image with _rot extension of the file name
        image_filename_rot = self.filename_ext_list[0] + '_ROT.' + self.filename_ext_list[1]
        temp_image.save(image_filename_rot)
        temp_image.show()
        self.list_newfiles.append(image_filename_rot)
        # update dictionary value
        MyImage.Image_dict[self.filename] = self.list_newfiles
        MyImage.Image_new_count += 1
    except IOError:
        message = "Rotated file could not be created --> " + self.dir + self.filename
        MyImage.Image_log_file.writelnlines(message)

def crop(self):
    try:
        temp_image = self.image.crop((self._rand_crop_size()))
        # saving modified image with _crop extension of the file name
        image_filename_crop = self.filename_ext_list[0] + '_CROP.' + self.filename_ext_list[1]
        temp_image.save(image_filename_crop)
        temp_image.show()
        self.list_newfiles.append(image_filename_crop)
        # update dictionary value
        MyImage.Image_dict[self.filename] = self.list_newfiles
        MyImage.Image_new_count += 1
    except IOError:
        message = "Cropped file could not be created --> " + self.dir + self.filename
        MyImage.Image_log_file.writelnlines(message)

def watermark(self):
    try:
        # similar to above methods, creates a new watermarked image from original
        temp_image = self.image.copy()
        draw = ImageDraw.Draw(temp_image)
        text = "WATERMARK"
        draw.text((x, y), text, fill=(255, 255, 255, 255), font=font)
    except IOError:
Deep Learning Implementation

- Two identical CNN networks
- **Same weights** at every training steps
- Each network translates an input image into a single-dimensional feature vector
- L1 or L2 vector distance metrics
- …. Followed by an FC layer to collapse the vector_diff into a single-value
- …. Followed by a non-linearity criterion.
- Training: compare against an a-priory label. Forward pass and backwards pass through each.
- Inference: generate “same”/”different” label.
- Transfer learning – we don’t start from random ‘w’
Deep Learning Implementation
BigDL Distributed Deep Learning Framework

- BigDL makes Deep Learning without GPUs POSSIBLE
- BigDL embraces distributed and fault-tolerant Deep Learning
- BigDL code runs unchanged on a single machine or on a cluster.
- BigDL leverages tried-and-true Apache Spark framework
- BigDL has the same features and capabilities as TensorFlow, Keras, PyTorch
- BigDL can import/export models with TF, Keras, Torch
- High performance via:
  - Written in Apache Spark’s native language, Scala (+Python API)
  - Intel’s latest-and-greatest MLK library (comes bundled with BigDL)
- Open-sourced - bigdl-project.github.io
Implementation - BigDL

```python
def get_vgg():
    vgg0 = Net.load_bigdl("/home/lizhichao/bin/data/model
    vgg1 = vgg0.new_graph(outputs=["drop7"])
    shared_vgg = Sequential()
    shared_vgg.add(InputLayer(input_shape=image_shape))
    shared_vgg.add(vgg1.toKeras())
    shared_vgg.add(Flatten())  # output [None, 4096]
    shared_vgg.add(Reshape([-1, 1, 1]))  # output [None,
    time = TimeDistributed(layer=shared_vgg)(input)
    vgg_features = Reshape([2, -1])(time)  # back to [-1, 2, 4096]
    diff = Lambda(function=ll)(vgg_features)
    fc = Dense(1)(diff)
    output = Activation("sigmoid")(fc)
    model = Model(input, output)
    model.compile(optimizer=SGD(learningrate=0.001),
                   loss=CECriterion(), metrics=None)
    model.fit(x=images, y=labels, batch_size=12, nb_epoch=30)
    out_data = model.forward(mock_x)
```

Building BigDL Graph

- Import VGG-16.
- Invoke VGG-16 model, tap fc7 layer’s output,
- Cast as Keras-like layers
- Reshape output tensor
- Use L1-difference to compare
- Add a fully-connected layer (1-dim output)
- Define loss function and criterion

Executing BigDL Graph

- Invoking model ‘fit’ training function
- Capture results.

Reference code to be published on BigDL github
Implementation - TensorFlow

```
train_dataset = tf.data.Dataset.from_tensor_slices((train_filenames, train_labels))
train_dataset = train_dataset.map(_parse_function, num_parallel_calls=None)
train_init_op = iterator.make_initializer(train_dataset)
val_init_op = iterator.make_initializer(val_dataset)
test_init_op = iterator.make_initializer(test_dataset)

vgg = tf.contrib.slim.nets.vgg

with slim.arg_scope(vgg.vgg_arg_scope(weight_decay=0.00004, is_training=False), dropout_keep_prob=0.5):
    _, layers = vgg.vgg_16(images, num_classes=num_classes, is_training=False, dropout_keep_prob=0.5)

    out_fc7 = layers['vgg_16/fc7'][:,:1024]
    out_fc7 = tf.reduce_mean(out_fc7, [1,2])

    abs_diff = tf.abs(out_fc7_a - out_fc7_b)
    logits = tf.layers.dense(abs_diff, 1, use_bias=False, name='fc_extra')

    loss = tf.nn.sigmoid_cross_entropy(labels, logits)
    loss = tf.losses.get_total_loss()

with tf.Session(graph=graph) as sess:
    init_fn(sess)  # load the pretrained weights
    sess.run([fc8_init, fc7_init, fc_extra_init])  # initialize the new fc8 layer
    for epoch in range(args.num_epochs):
        sess.run(train_init_op)
        while True:
            try:
                out = sess.run([logits, labels, correct_prediction, loss, fc7_train_op], is_training=True)
                logits_out, labels_out, corr_pred, loss_out, _ = out
```

Building TF Graph

- Form a training dataset.
- Set up iterators to iterate over the dataset.
- Import VGG-16 model.
- Invoke VGG-16 model, tap fc7 layer’s output.
- Split fc7 output tensor into odd/even elements.
- Use L1-difference to compare.
- Add a fully-connected layer (1-dim output).
- Define loss function.

Executing TF Graph

- Train on batched dataset.
- Capture results.

https://github.com/sermolin/Siamese_image_duplicates
Results

Discussion

- Different Norm choice

<table>
<thead>
<tr>
<th>Difference norm used</th>
<th>Accuracy</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 (abs)</td>
<td>0.62</td>
<td>0.59</td>
</tr>
<tr>
<td>L2 (square)</td>
<td>0.75</td>
<td>0.98</td>
</tr>
</tbody>
</table>

- Baseline (KNN): Accuracy=24%.

- Siamese networks are very good at identifying identical images (Precision)

- Siamese network produces quite a bit of false negatives which could be improved with better training dataset.
Emerging Business Use-Case – Fraud prevention

- Data compliance
- Property images could be used for predicting house valuation.
- Upgraded images => Higher price
  - Market price setting
  - Property value appraisal for loan approval (serious business)
  - Insurance risk exposure estimation
- Are the property’s images unique or “borrowed” from elsewhere (duplicates)?
Questions?
Thank you!
WHO IS BUILDING WHAT WITH BIGDL?

CONSUMER
- Gigaspaces
- MLS Listings
- Jobs Search Engine

HEALTH
- UnionPay
- ChinaLife (Insurance)
- Major credit card issuer

FINANCE
- JD.Com

RETAIL
- Steel manufacturing

MANUFACTURING
- Steel Surface defect detection
- Weather forecasting

SCIENTIFIC COMPUTING
- Call center routing,
- Image similarity search,
- smart job search
- Analysis of 3D MRI models for knee degradation
- Fraud detection
- Recommendation,
- Customer/Merchant Propensity
- Image feature extraction (Inference)
WHY BIGDL?

- Does BigDL offer lower TCO? **YES**
- It is the story of efficient enterprise scale-out and resource leveraging:
  - Reuse existing Spark clusters
    - Leverage existing infra of deployment, monitoring, support, etc.
  - Reuse existing Spark data pipelines
    - Incrementally add AI capabilities to existing flow rather than develop brand-new pipelines
  - Compute where your data is (data efficiency).
    - In FinTech, often you can’t even move data outside of “datalake”
  - Written in Scala – leverage expertise of Spark engineers (+Python API)
  - No need to change Spark infrastructure: “spark-submit bigdl_app.jar”
  - *That's How Customers Want It*

bigdl-project.github.io  software.intel.com/bigdl
Customer Testaments (based on their own framework comparison):

- TensorFlow/Caffe runs on specialized HW+interconnect - $$$
- Open MP Implementation of TensorFlow/Caffe-on-Spark conflicts with Spark’s JVM threading – lower performance
- TensorFlow/Spark can only interact with the rest of the analytics pipelines in a very coarse-grained fashion

### Major Spark Deep Learning Frameworks

<table>
<thead>
<tr>
<th>Framework</th>
<th>Programming Interface</th>
<th>Contributors</th>
<th>commits</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigDL</td>
<td>Scala &amp; Python</td>
<td>50</td>
<td>2221</td>
</tr>
<tr>
<td>TensorflowOnSpark</td>
<td>Python</td>
<td>9</td>
<td>257</td>
</tr>
<tr>
<td>Databricks/tensor</td>
<td>Python</td>
<td>9</td>
<td>185</td>
</tr>
<tr>
<td>Databricks/spark-deep-learning</td>
<td>Python</td>
<td>8</td>
<td>51</td>
</tr>
</tbody>
</table>

Statistics collected on Mar 5th, 2018
BIGDL

Jupyter, Zeppelin notebooks and TensorBoard support

SCALARS  IMAGES  AUDIOs  DISTRIBUTIONS  HISTOGRAMS

Exploring the Lorenz System

In this Notebook we explore the Lorenz system of differential equations:
\[
\begin{align*}
\dot{x} &= \sigma(y - x) \\
\dot{y} &= x - y + xy \\
\dot{z} &= -bz + xy
\end{align*}
\]

This is one of the classic systems in non-linear differential equations. It exhibits a range of complex behaviors as the parameters \((\sigma, b, \beta)\) are varied, including what are known as chaotic solutions. The system was originally developed as a simplified mathematical model for atmospheric convection in 1963.

conv2_5x5/weight
Train Lenet on MNIST/lenet/train

Loss

Loss

bigdl-project.github.io  software.intel.com/bigdl
BIGDL DNN FOR APACHE SPARK

- **BigDL** makes Deep Learning without GPUs **POSSIBLE**
- **BigDL** **embraces** distributed and fault-tolerant Deep Learning
- **BigDL** code runs **without changes** on a single machine or on a cluster.
- **BigDL** leverages tried-and-true Apache Spark framework
- **BigDL** has the same **features and capabilities** as TF, Keras, PyTorch, etc
- **BigDL** can **exchange models** with TF, Keras, Torch
- **High performance via:**
  - Written in Apache Spark’s native language, Scala (Python API as well)
  - Intel’s latest-and-greatest MLK library (pre-bundled with BigDL)
- Open-sourced

bigdl-project.github.io  software.intel.com/bigdl