Convolutional Neural Network for Track Seed Filtering at the CMS High-Level Trigger

Felice Pantaleo$^1$, Adriano Di Florio$^2$, Antonio Carta$^3$, Maurizio Pierini$^1$

$^1$CERN, Experimental Physics Department
$^2$University of Bari
$^3$University of Pisa

felice@cern.ch
Two-stages event selection strategy

Trigger System
- Reduce input rate (40 MHz) to a data rate (~1 kHz) that can be stored, reconstructed and analyzed Offline maximizing the physics reach of the experiment

Level 1 Trigger
- coarse readout of the Calorimeters and Muon detectors
- implemented in custom electronics, ASICs and FPGAs
- output rate limited to 100 kHz by the readout electronics

High Level Trigger
- readout of the whole detector with full granularity
- based on the CMSSW software, running on 26,000 Xeon cores
- organized in O(2500) modules, O(400) trigger paths, O(10) streams
- output rate limited to an average of ~1 kHz by the Offline resources
Tracking at the CMS High-Level Trigger

• Today the CMS online farm consists of ~26k Intel Xeon cores
  – The current approach: one event per logical core
• Pixel Tracks cannot be reconstructed for all the events at the HLT
• This will be even more difficult at higher pile-up
  – Combinatorial time in pixel seeding $O(\text{pileup}!)$ in worst case
  – More memory/event
Online:
• Pixel-only tracks used for fast tracking and vertexing

Offline:
• Pixel tracks are used as seeds for the Kalman filter in the strip detector
The already complex online and offline track reconstruction has to deal not only with a much more crowded environment but also with data coming from a more complex detector.
The already complex online and offline track reconstruction has to deal not only with a much more crowded environment but also with data coming from a more complex detector.
### Tracking at HLT

- **Pixel hits are used for pixel tracks, vertices, seeding**
- **HLT Iterative tracking:**

<table>
<thead>
<tr>
<th>Iteration name</th>
<th>Phase0 Seeds</th>
<th>Phase1 Seeds</th>
<th>Target Tracks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel Tracks</td>
<td>triplets</td>
<td>quadruplets</td>
<td></td>
</tr>
<tr>
<td>Iter0</td>
<td>Pixel Tracks</td>
<td>Pixel Tracks</td>
<td>Prompt, high $p_T$</td>
</tr>
<tr>
<td>Iter1</td>
<td>triplets</td>
<td>quadruplets</td>
<td>Prompt, low $p_T$</td>
</tr>
<tr>
<td>Iter2</td>
<td>doublets</td>
<td>triplets</td>
<td>High $p_T$, recovery</td>
</tr>
</tbody>
</table>
Doublets generation

• Open a search window in the innermost layer
  – Size depends on the minimum value of transverse momentum which is allowed, transverse impact parameter, longitudinal impact parameter
CA-based HitChain Maker

• The CA is a track seeding algorithm designed for parallel architectures
• It requires a list of layers and their pairings
  – A graph of all the possible connections between layers is created
  – Doublets aka Cells are created for each pair of layers (compatible with a region hypothesis)

See talk by my evil twin on this afternoon, Track 1
There is some more information...
• **Doublets generation**: bottleneck due to huge combinatorial background.
• E.g. $\sim 10^5 - 10^6$ doublets produced @ PU35 for $TT\bar{B}ar_{13TeV}$ event
• Typical binary classification problem: keep true doublets & reject fake doublets
• Hit is a 15x15 pixel pad/image
• Cluster centered
• Pattern recognition problem: suitable for a Convolutional Neural Network approach
Training dataset

• ttbar +PU35:
  – 500k doublets per event
  – Heavily unbalanced: 200-300 fakes per good doublet,
  – 3000 true doublets per event

Association RECO - MC
1. get list of all matched reconstructed tracks track hits
2. get list of all doublets produced
3. true doublets = doublets formed by hits from the same sim matched track
450 ADC pixels [2x15x15 pads]

\text{inPixLab} = ["inPix1", "inPix2", \ldots, "inPix224", "inPix225"]

\text{outPixLab} = ["outPix1", "outPix2", \ldots, "outPix224", "outPix225"]

63 features defined for each doublet [true or fake] that may be used as additional features to the pixel pad

\text{headLab} = ["run", "evt", "detSeqIn", "detSeqOut", "inX", "inY", "inZ", "outX", "outY", "outZ", "inPhi", "inR", "outPhi", "outR", "detCounterIn", "detCounterOut", "isBarrelIn", "isBarrelOut", "layerIn", "ladderIn", "moduleIn", "sideIn", "diskIn", "panelIn", "bladeIn", "layerOut", "ladderOut", "moduleOut", "sideOut", "diskOut", "panelOut", "bladeOut", "isBigIn", "isBigOut", "isEdgIn", "isEdgOut", "isBadIn", "isBadOut", "isFlippedIn", "isFlippedOut", "iCSize", "pixInX", "pixInY", "inClusterADC", "iZeroADC", "iCSize", "iCSizeX", "iCSizeY", "iOverFlowX", "iOverFlowY", "oCSize", "pixOutX", "pixOutY", "outClusterADC", "oZeroADC", "oCSize", "oCSizeX", "oCSizeY", "oOverFlowX", "oOverFlowY", "diffADC"]

24 labels defined only for MC matched doublets

\text{tailLab} = ["idTrack", "px", "py", "pz", "pt", "mT", "eT", "mSqr", "rapidity", "etaTrack", "phi", "pdgId", "charge", "noTrackerHits", "noTrackerLayers", "dZ", "dXY", "Xvertex", "Yvertex", "Zvertex", "bunCross", "isCosmic", "chargeMatch", "sigMatch"]

Normalization with incident angle
Two approaches

• Detector tuning
• Feature map
• Taking into account the modules’ flipping **inward** or **outward** (Lorentz angle changes)
• Dividing the datasets in chunks for **detector pairs**: only doublets from specific detector pair (e.g. Barrel-Barrel).
Detector tuning

• **Detector approach:** selecting data only from detector couples doublets

• **Concatenates:**
  - **CNN architecture:** stack of convolutional layers (4) and max pooling (2)
  - **“DENSE” architecture:** dense layers (2) fed with the 1-dim reduced images + doublets infos (inX, inY, inZ, ...)

• **Dropouts** to prevent overfitting

• **Train & val datasets balanced** (0.5)

---

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>hit_shape_input (InputLayer)</td>
<td>(None, 15, 15, 8)</td>
<td>0</td>
</tr>
<tr>
<td>dropout_1 (Dropout)</td>
<td>(None, 15, 15, 8)</td>
<td>0</td>
</tr>
<tr>
<td>conv1 (Conv2D)</td>
<td>(None, 15, 15, 32)</td>
<td>6432</td>
</tr>
<tr>
<td>conv2 (Conv2D)</td>
<td>(None, 15, 15, 32)</td>
<td>9248</td>
</tr>
<tr>
<td>pool1 (MaxPooling2D)</td>
<td>(None, 8, 8, 32)</td>
<td>0</td>
</tr>
<tr>
<td>conv3 (Conv2D)</td>
<td>(None, 8, 8, 64)</td>
<td>18496</td>
</tr>
<tr>
<td>conv4 (Conv2D)</td>
<td>(None, 8, 8, 64)</td>
<td>36928</td>
</tr>
<tr>
<td>pool2 (MaxPooling2D)</td>
<td>(None, 4, 4, 64)</td>
<td>0</td>
</tr>
<tr>
<td>flatten_1 (Flatten)</td>
<td>(None, 1024)</td>
<td>0</td>
</tr>
<tr>
<td>info_input (InputLayer)</td>
<td>(None, 59)</td>
<td>0</td>
</tr>
<tr>
<td>concatenate_1 (Concatenate)</td>
<td>(None, 1083)</td>
<td>0</td>
</tr>
<tr>
<td>batch_normalization_1 (BatchNorm)</td>
<td>(None, 1083)</td>
<td>4332</td>
</tr>
<tr>
<td>dense1 (Dense)</td>
<td>(None, 128)</td>
<td>138752</td>
</tr>
<tr>
<td>dropout_2 (Dropout)</td>
<td>(None, 128)</td>
<td>0</td>
</tr>
<tr>
<td>dense2 (Dense)</td>
<td>(None, 64)</td>
<td>8256</td>
</tr>
<tr>
<td>dropout_3 (Dropout)</td>
<td>(None, 64)</td>
<td>0</td>
</tr>
<tr>
<td>output (Dense)</td>
<td>(None, 2)</td>
<td>130</td>
</tr>
</tbody>
</table>

Total params: 222,574.0
Trainable params: 220,498.0
Non-trainable params: 2,166.0
Accuracies and loss function for train and validation sets on GTX1080
- Train: 0.912
- validation: 0.913
- test: 0.909
**fMap approach**

- Feature layer map (fMap model) approach: each layer (inner & outer) has its own channel
- **Concatenates:**
  - **CNN architecture:** stack of convolutional layers (4) and max pooling (2)
  - **“DENSE” architecture:** dense layers (2) fed with the 1-dim reduced images + doublets infos (inX,inY,inZ … )
- **Dropouts** to prevent overfitting
- **Train & val datasets balanced (0.5)**
• Accuracies and loss function for train and validation sets on GTX1080
  – Train: 0.909
  – validation: 0.911
  – test: 0.906

Efficiency (tpr) @ fake rejection
  - tpr @ rej 50%: 0.998996700259
  - tpr @ rej 75%: 0.990524391331
  - tpr @ rej 90%: 0.922210826719
  - tpr @ rej 99%: 0.338669401587
Conclusion

- CNN techniques for mitigating combinatorial explosion look very promising
- Exploring the integration in the CMS reconstruction Framework
- Verification of the effect on the downstream track reconstruction ongoing
- Exploration of different hardware architecture for fast inference ongoing
Backup