The HEP.TrkX Project: Deep Neural Networks for HEP Tracking

Steve Farrell
on behalf of the HEP.TrkX project

Connecting the Dots / Intelligent Trackers Workshop
March 9, 2017

Berkeley Lab — Ernest Orlando Lawrence Berkeley National Laboratory
Caltech
Fermilab

Introduction

- **Current tracking algorithms have been used very successfully in HEP/LHC experiments**
  - Good efficiency and modeling with acceptable throughput/latency
- **However, they don't scale so well to HL-LHC conditions**
  - Thousands of charged particles, $O(10^5)$ 3D spacepoints, while algorithms scale worse than quadratic
- Thus, it's worthwhile to try and think "outside the box"; i.e., consider ***Deep Learning algorithms***
  - Relatively unexplored area of research
  - Might see major improvements… who knows?

The HEP.TrkX project

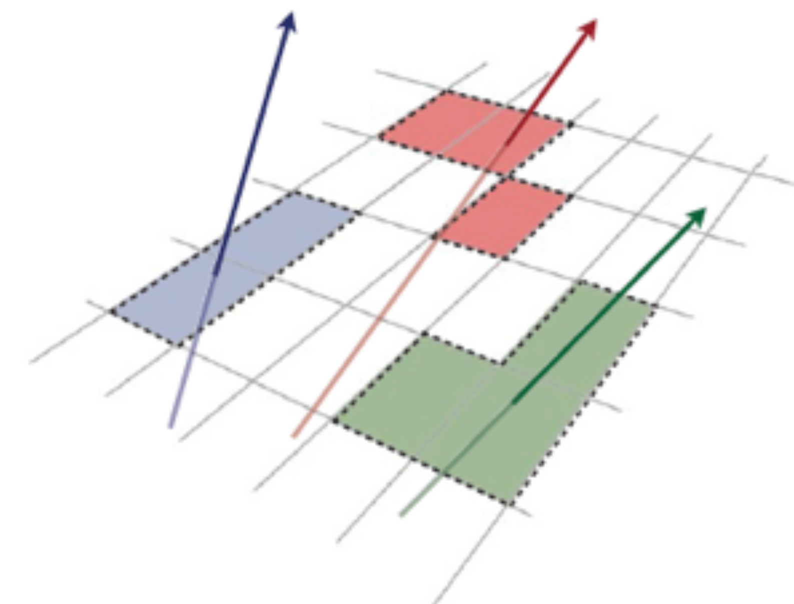
- **A 1-year pilot project to develop ML algorithms for HEP tracking**
  - Funded by DOE ASCR and COMP HEP, part of HEP CCE
  - Collaboration between ATLAS, CMS, LAr folks from LBL, Caltech, and FNAL

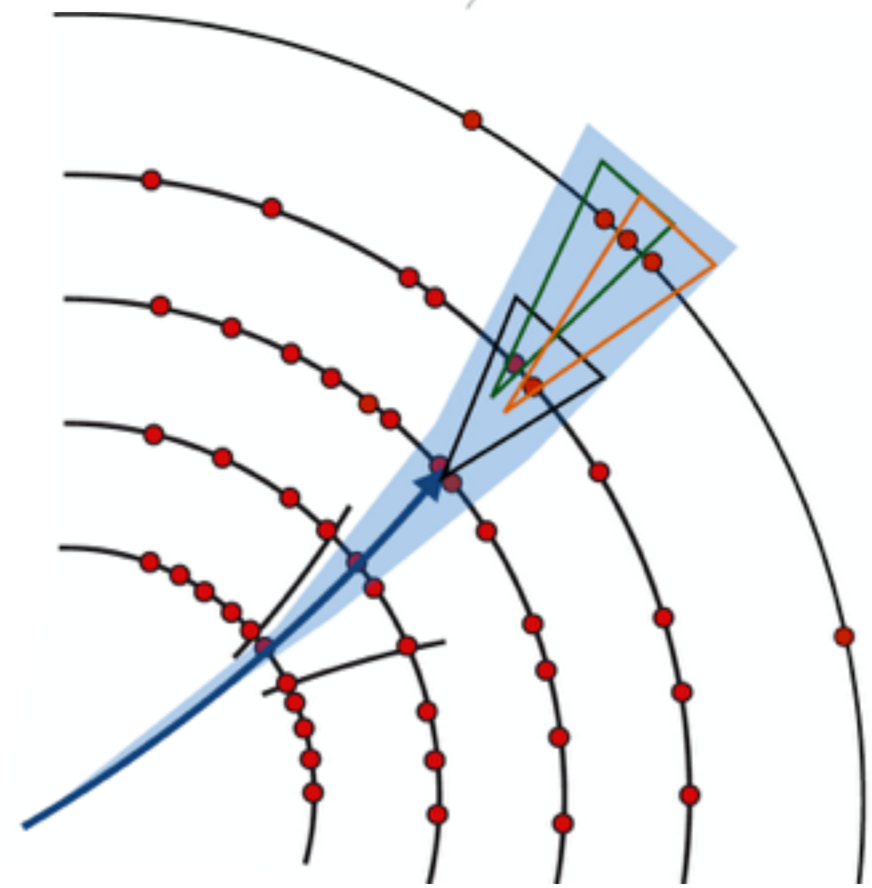
    LBL: Me, Mayur Mudigonda, Prabhat, Paolo
    Caltech: Dustin Anderson, Jean-Roch Vlimant, Josh Bendavid, Maria Spiropoulou, Stephan Zheng
    FNAL: Aristeidis Tsaris, Giuseppe Cerati, Jim Kowalkowski, Lindsey Gray, Panagiotis Spentzouris

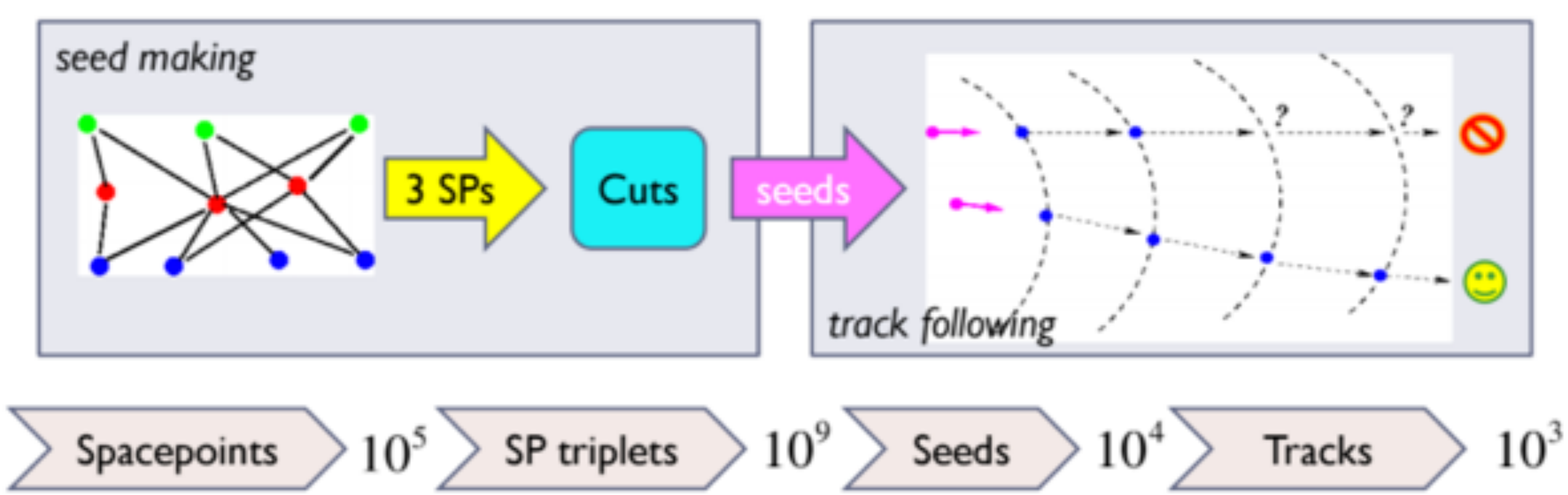
- **Some goals**
  - Explore the broad space of ideas on simplified tracking problems
  - Develop a toolkit of promising ideas
    - ideas that work (physics constraints)
    - ideas that *scale* (computing constraints)
- **The work is in an *exploratory phase***
  - Testing ideas in a breadth-first fashion
  - Very much a work-in-progress

Current algorithmic approach (ATLAS, CMS)

- Divide the problem into sequential steps
    1. Cluster hits into 3D spacepoints
    2. Build triplet “seeds”
    3. Build tracks with combinatorial Kalman Filter
    4. Resolve ambiguities and fit tracks







Credit: Andy Salzburger

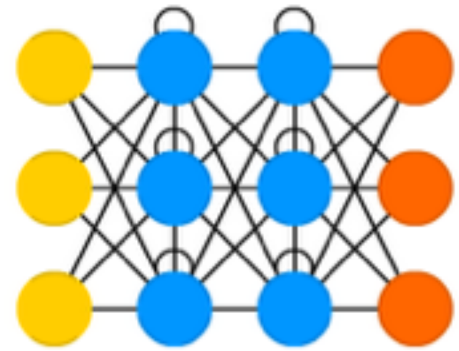
How to incorporate machine learning techniques?

- **What part(s) of the problem to replace?**
  - Seeding, single-track building, fitting?
  - Seeded multi-track finding?
  - All-in-one hits to list of tracks?
- **How to represent the data?**
  - Clustered hits in continuous space or raw pixel data?
    - or *binned* clusters..?
  - List of hits, or list of 4-momenta?
    - uncertainties, too?
- **How to deal with the many challenges?**
  - sparsity and irregularity in the data
  - defining *differentiable* cost functions (wrestling ambiguities)
  - requirements for fine-level control and interpretability of the model
  - and of course: *space and time complexity constraints!*

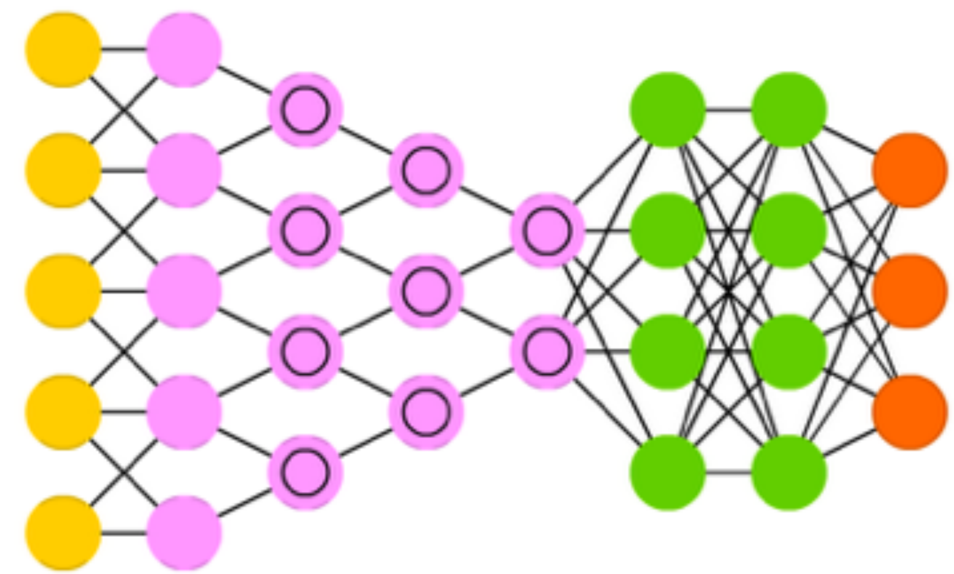
Deep neural network architectures

Deep Feed Forward (DFF)

Recurrent Neural Network (RNN)



Deep Convolutional Network (DCN)

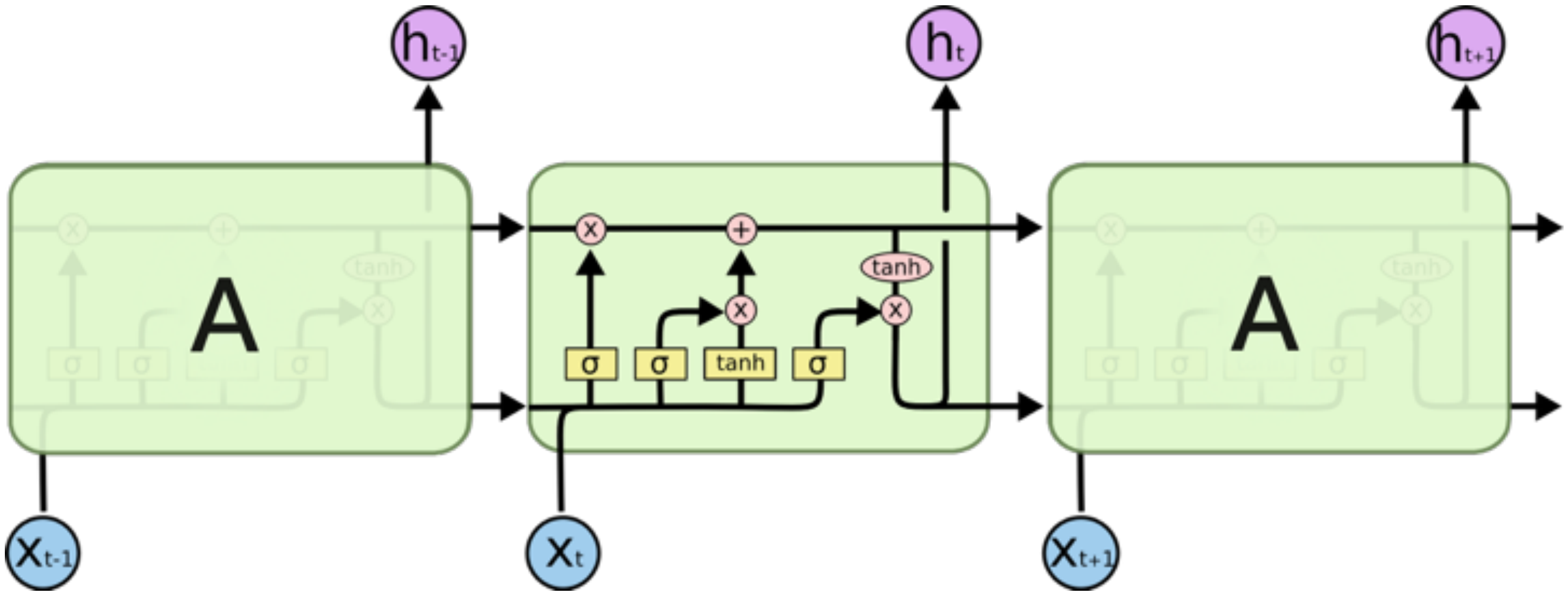


- **Fully-connected (feed-forward) networks**
  - Vanilla MLPs with fixed input, output size
  - Good for classification, regression
  - Common building block in complex models
- **Recurrent networks**
  - Model dependencies in sequence data
  - Variable-length data
- **Convolutional networks**
  - Hierarchical pattern finders (local to global)
  - Exploit translational invariance in data

http://www.asimovinstitute.org/neural-network-zoo

LSTM networks

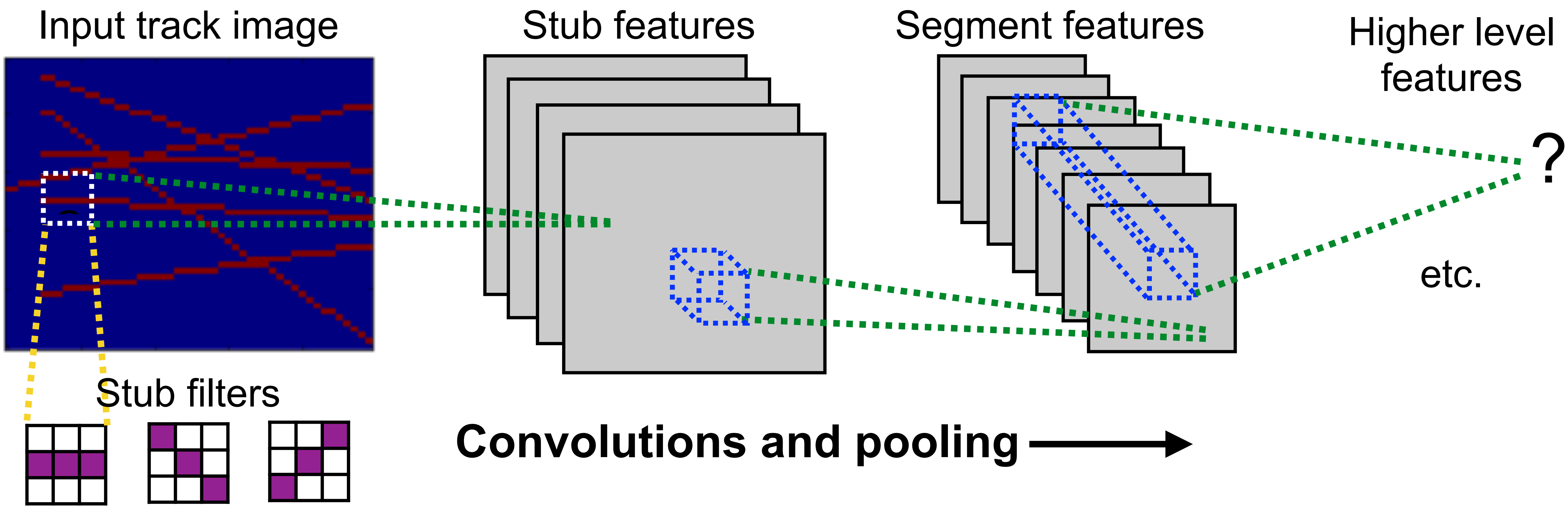
- LSTM (Long Short Term Memory) networks are *recurrent neural networks* that model long term dependencies in sequence data by carrying a *memory*



- Can be used for state estimation and modeling of track dynamics
  - Kinda like a Kalman Filter
  - But it might actually be smarter!
    - Maybe it can model combinatorics for a track in one pass
    - Maybe it can process multiple tracks at once

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Convolutional networks as track finders

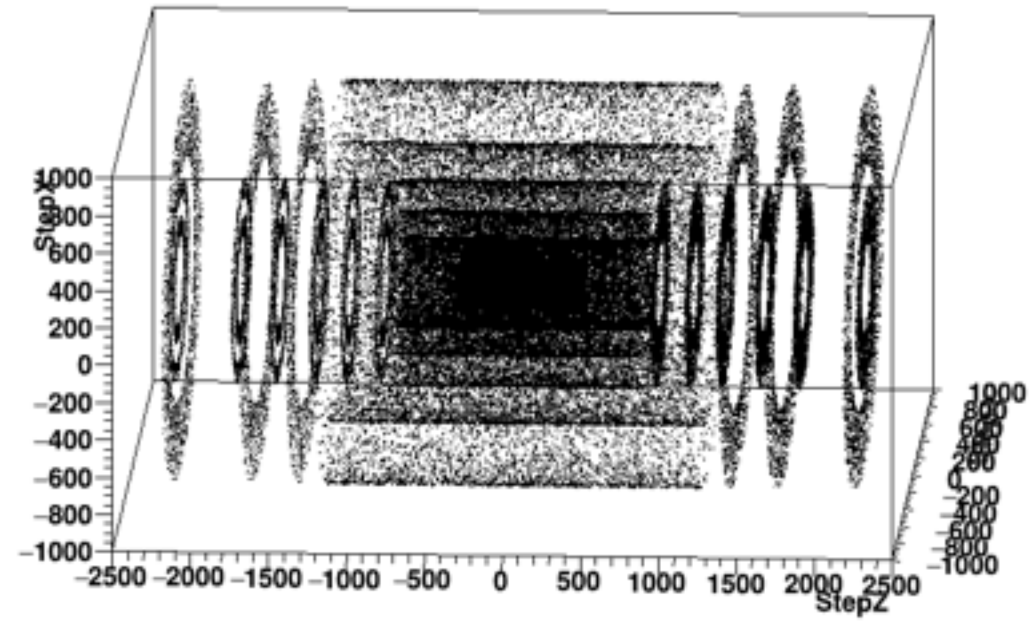


- **Convolutional filters can be thought of as track pattern matchers**
  - Early layers look for track stubs
  - Later layers connect stubs together to build tracks
  - Learned representations are in reality optimized for the data => may be abstract and more compact than brute force pattern bank
- **The learned features can be used in a variety of ways**
  - Extract out track parameters
  - Project back to detector image and classify hits

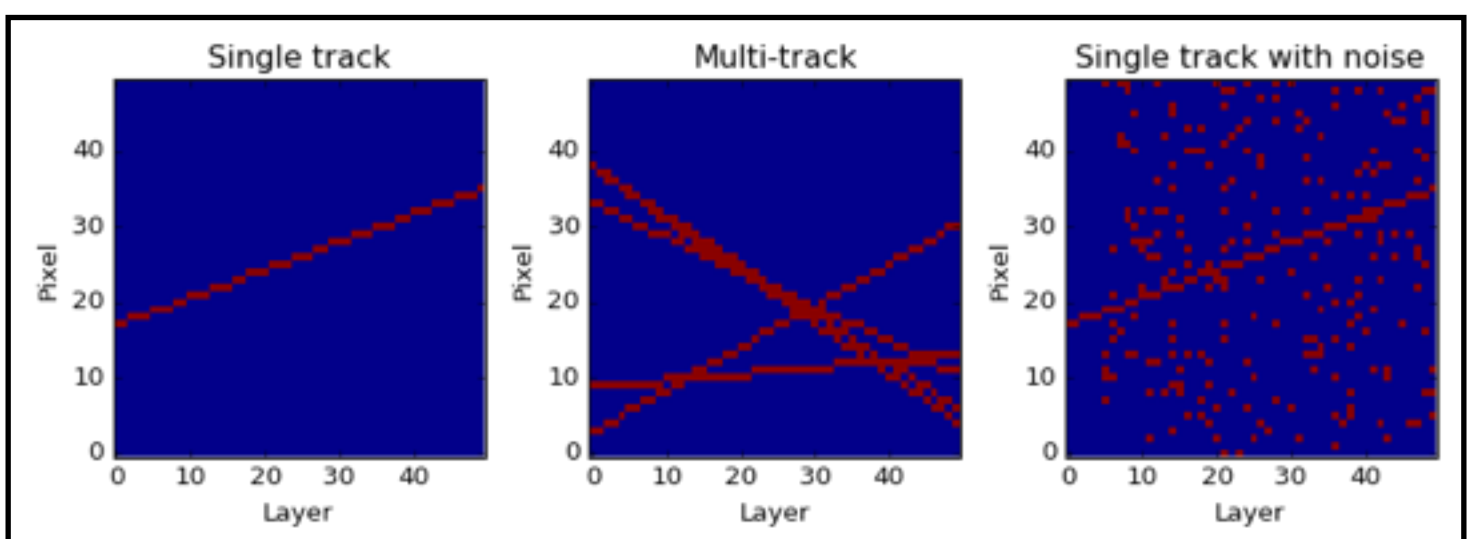
Datasets

- **Currently working with *absurdly simple* toy datasets**
  - Straight line tracks in 2D or 3D on simple detector planes
  - Perfect binary hits; no holes or charge-sharing
  - Random background tracks and/or uniform noise
- **We have also started playing with ACTS data**
  - KF-like models being explored now
  - The models I show today need to be extended to work on “realistic geometry”
    - Even then we expect to ignore endcaps for now ;)

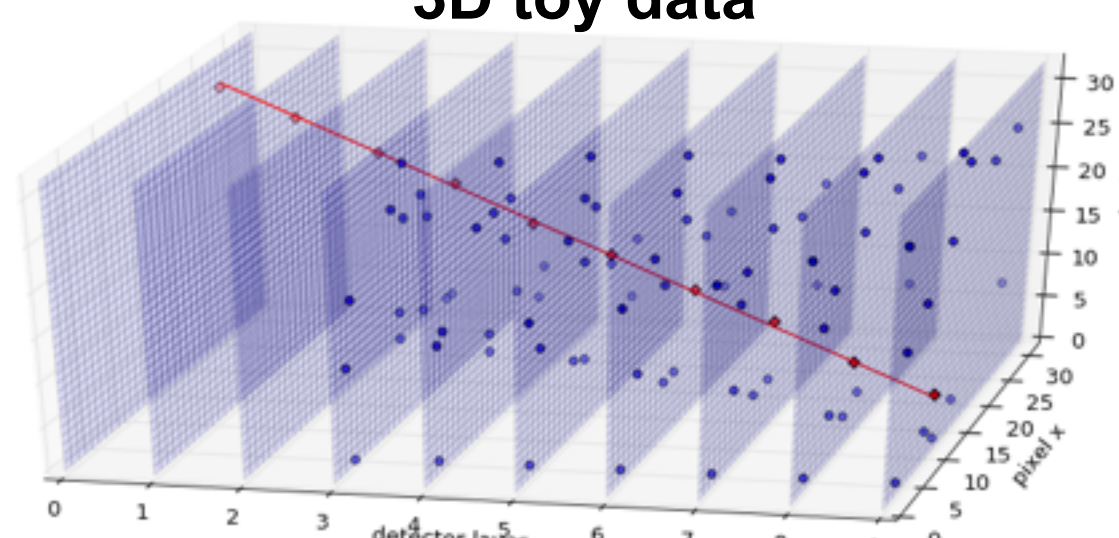
ACTS generic tracker



2D toy data

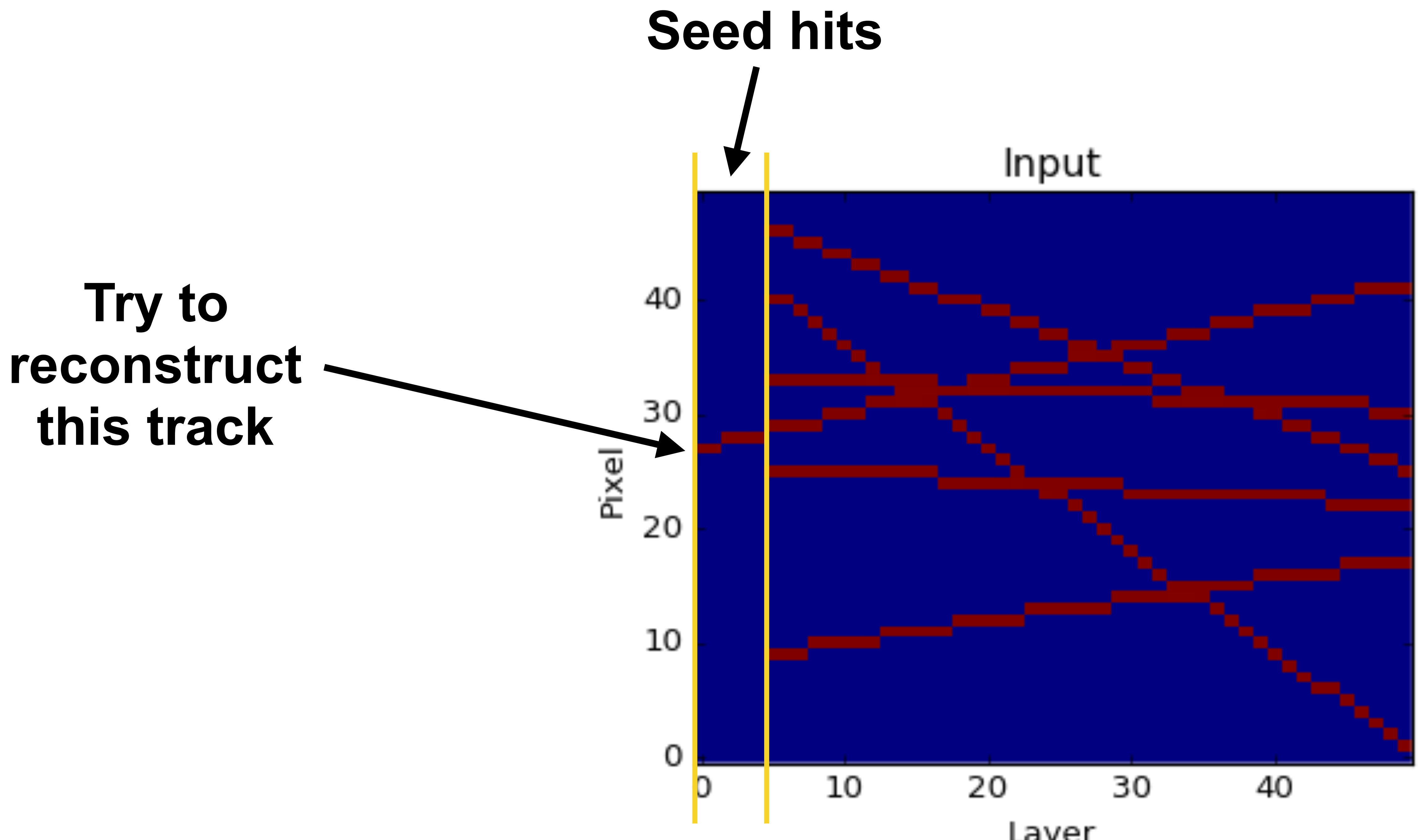


3D toy data



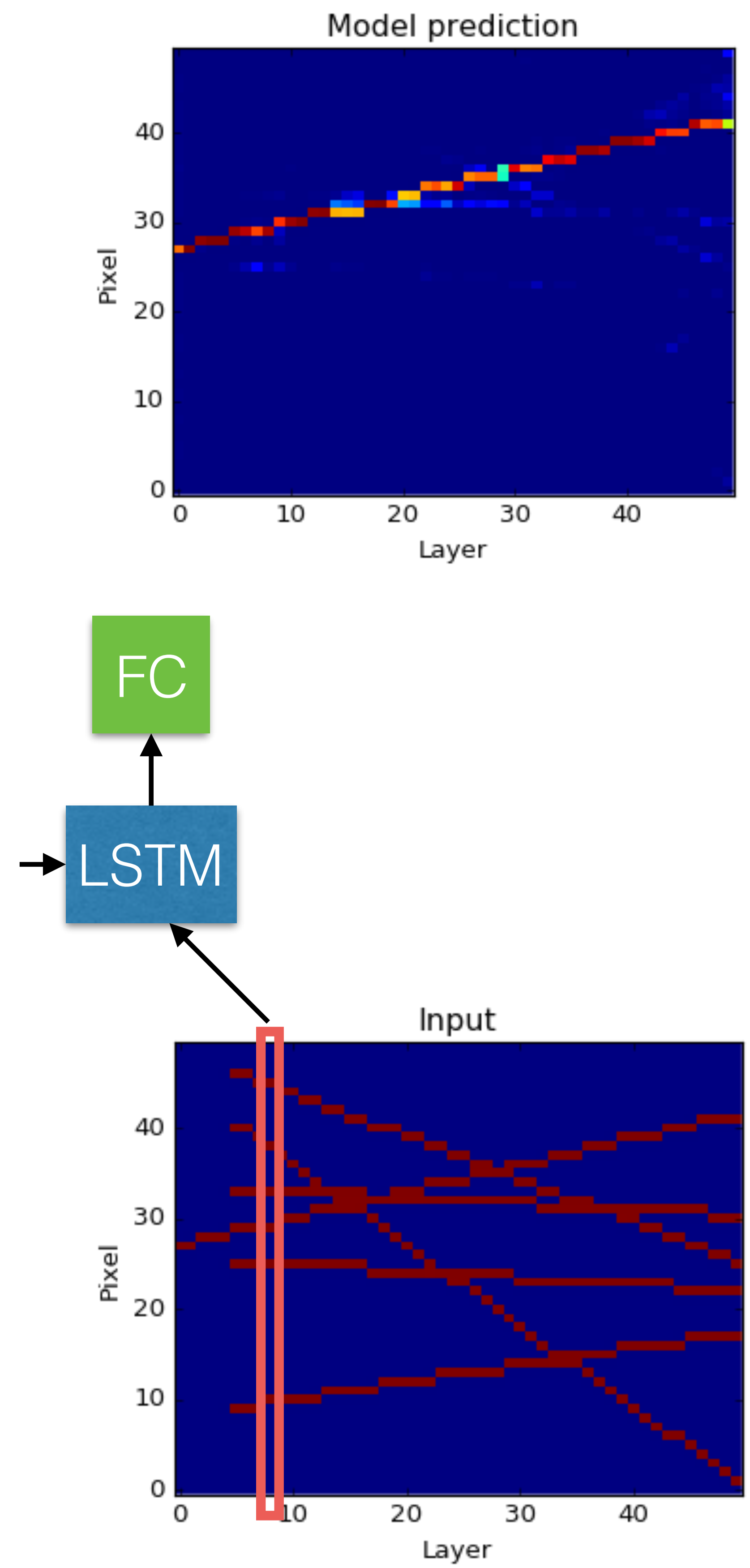
Track finding with LSTMs

- Try to build a single, *seeded* track from a set of hits with backgrounds



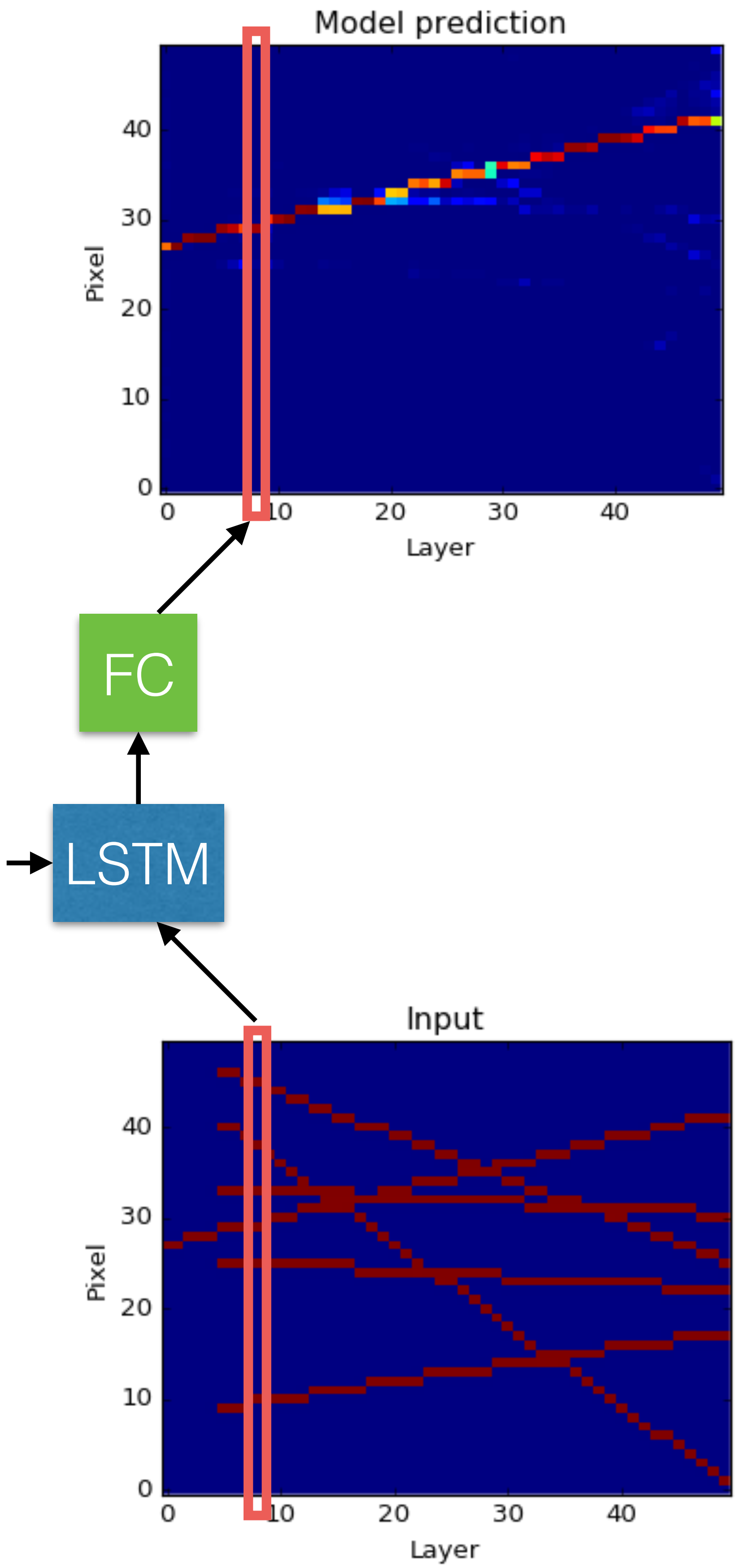
Track finding with LSTMs

- Try to build a single, *seeded* track from a set of hits with backgrounds
- Detector plane pixel arrays fed into the model one at a time



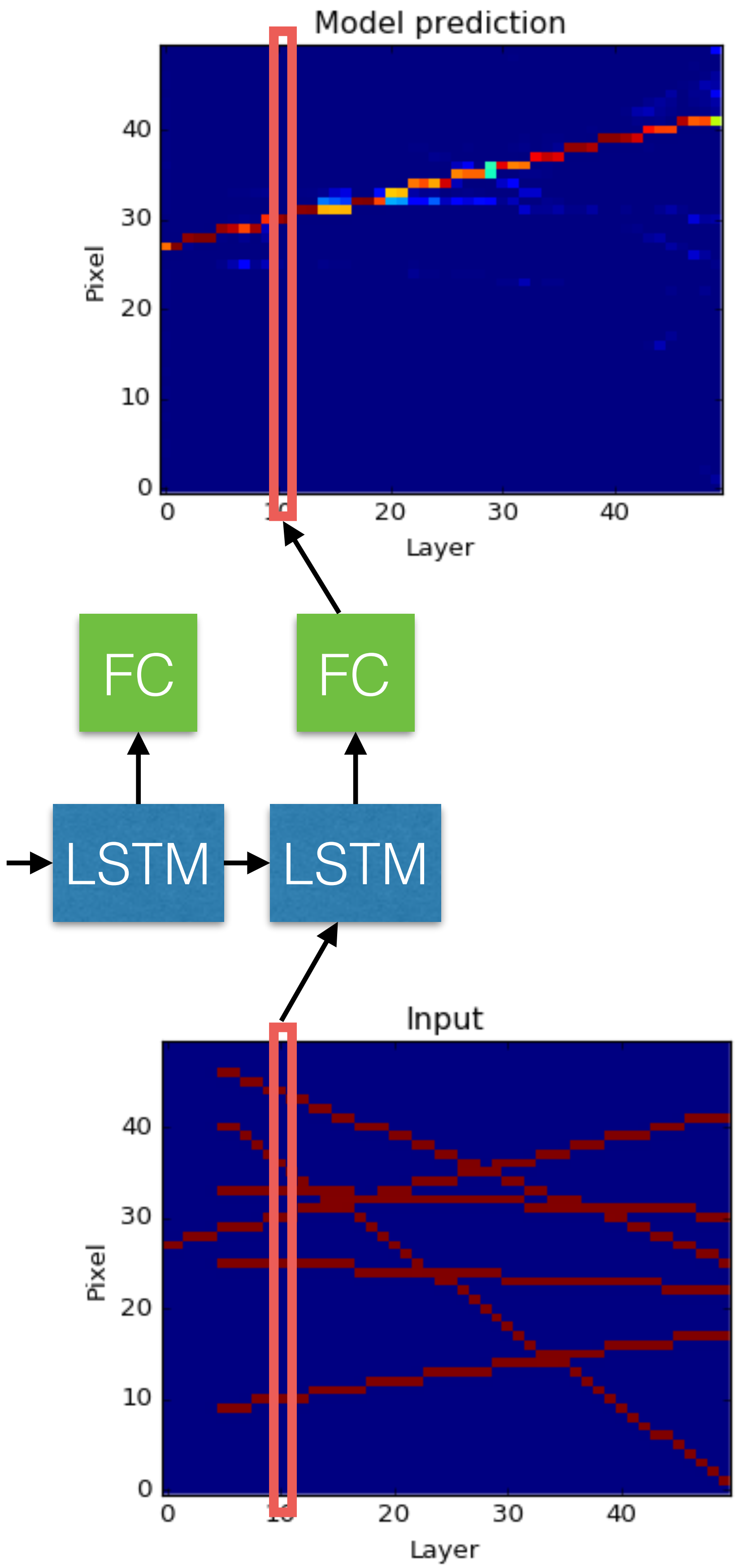
Track finding with LSTMs

- Try to build a single, *seeded* track from a set of hits with backgrounds
- Detector plane pixel arrays fed into the model one at a time
- The model spits out an array of “scores” for that detector plane
    - Pixel predictions (or hit “classification”)



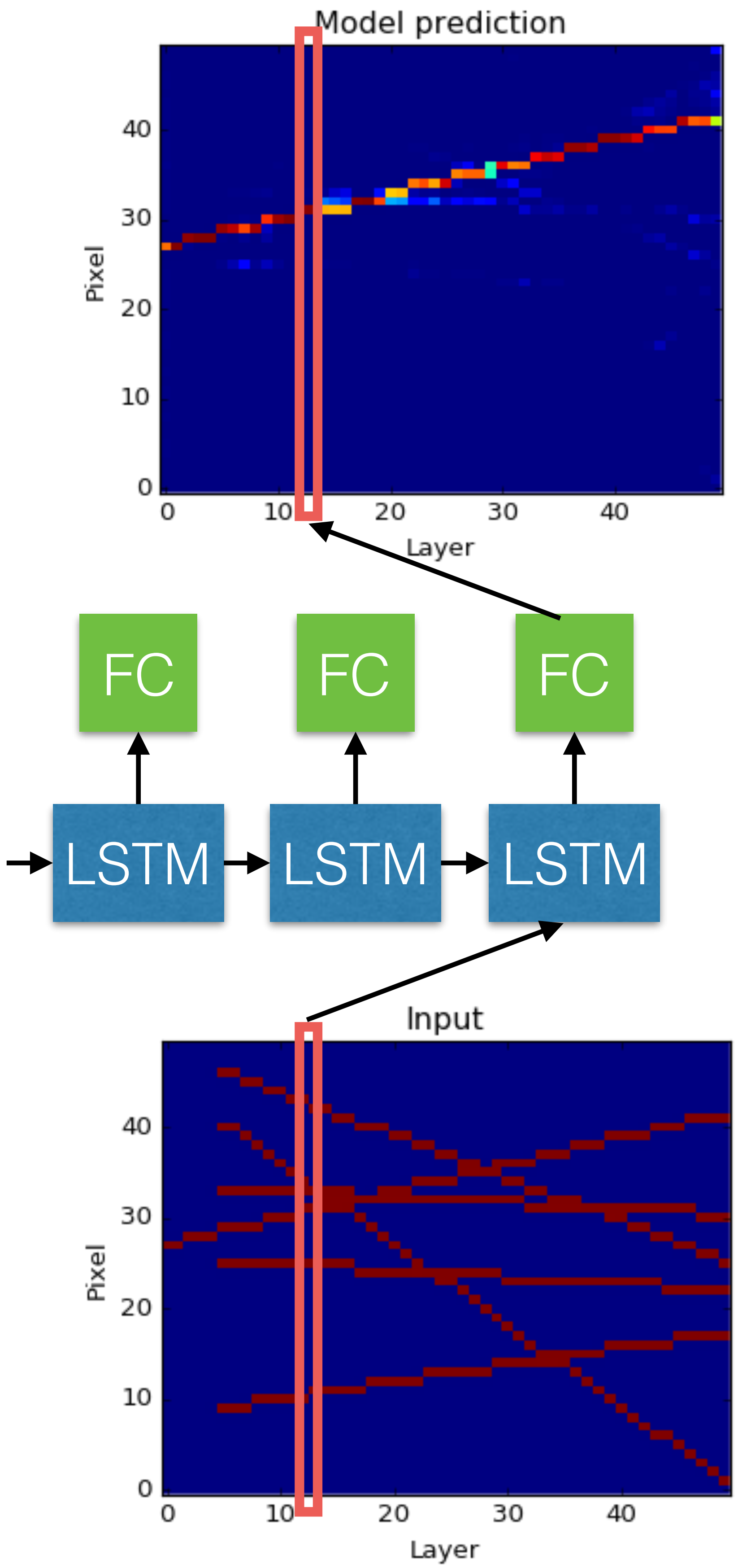
Track finding with LSTMs

- Try to build a single, *seeded* track from a set of hits with backgrounds
- Detector plane pixel arrays fed into the model one at a time
- The model spits out an array of "scores" for that detector plane
    - Pixel predictions (or hit "classification")
- The LSTM memory is used to carry the dynamic state estimate, updated at each iteration



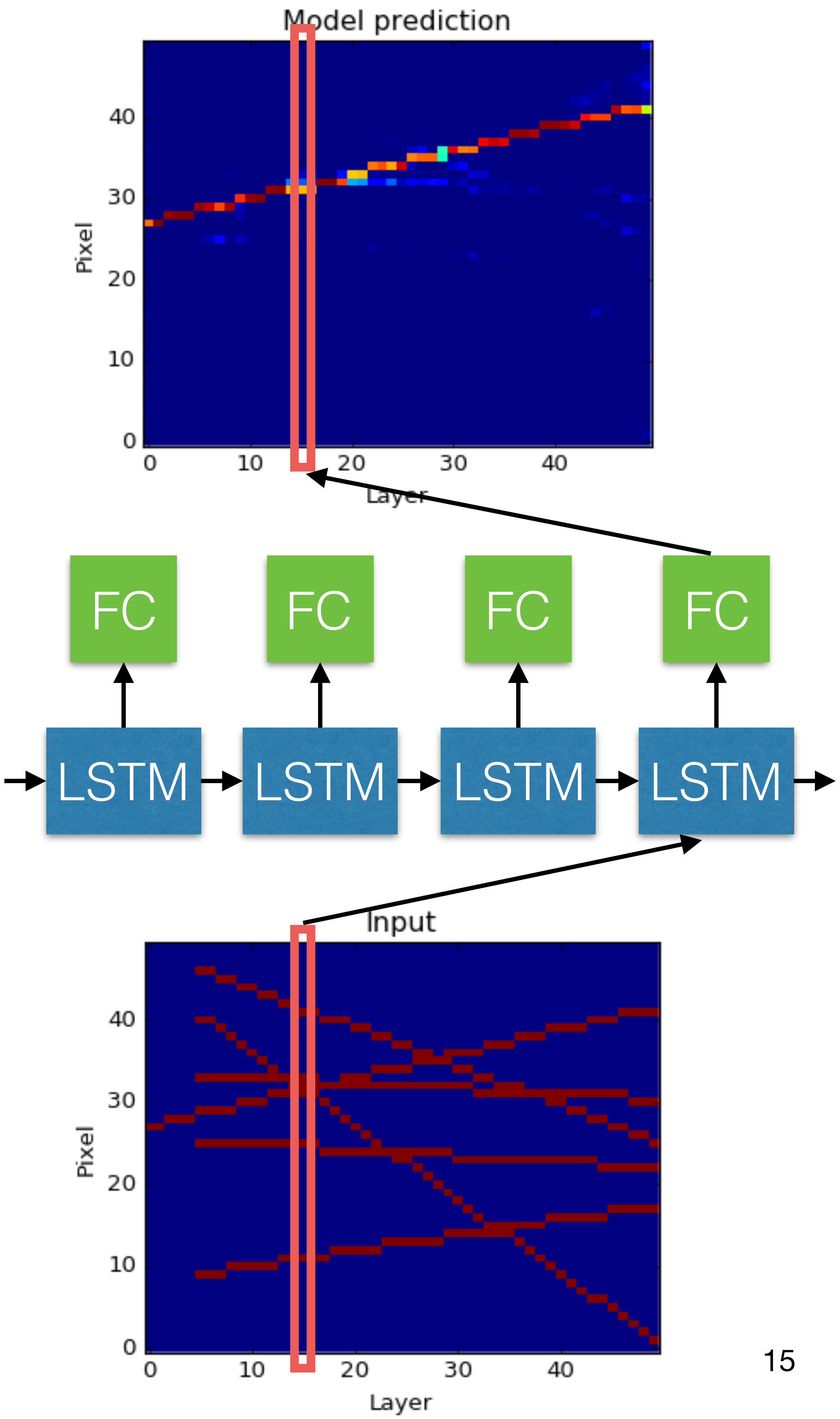
Track finding with LSTMs

- Try to build a single, *seeded* track from a set of hits with backgrounds
- Detector plane pixel arrays fed into the model one at a time
- The model spits out an array of “scores” for that detector plane
  - Pixel predictions (or hit “classification”)
- The LSTM memory is used to carry the dynamic state estimate, updated at each iteration



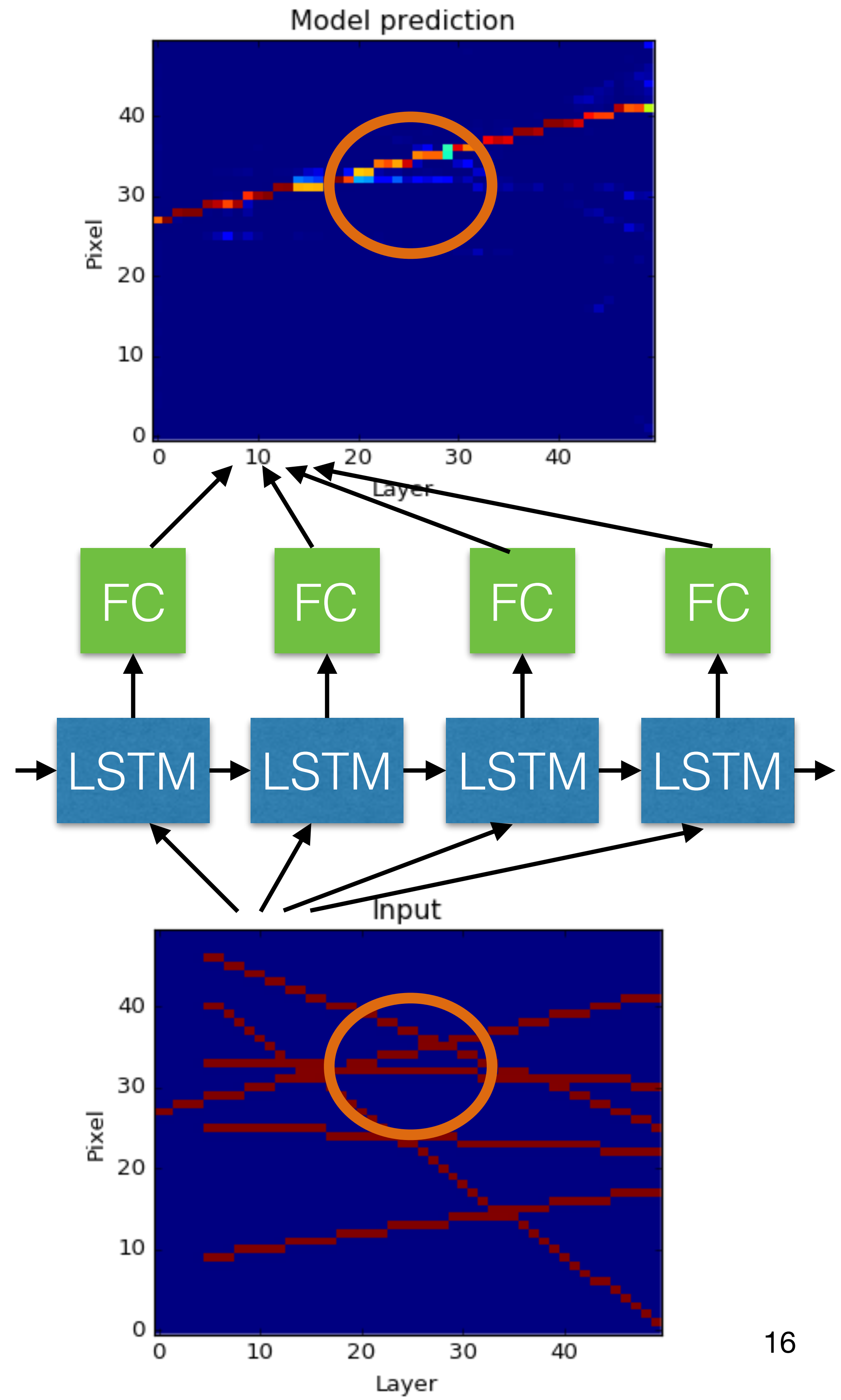
Track finding with LSTMs

- Try to build a single, *seeded* track from a set of hits with backgrounds
- Detector plane pixel arrays fed into the model one at a time
- The model spits out an array of “scores” for that detector plane
  - Pixel predictions (or hit “classification”)
- The LSTM memory is used to carry the dynamic state estimate, updated at each iteration



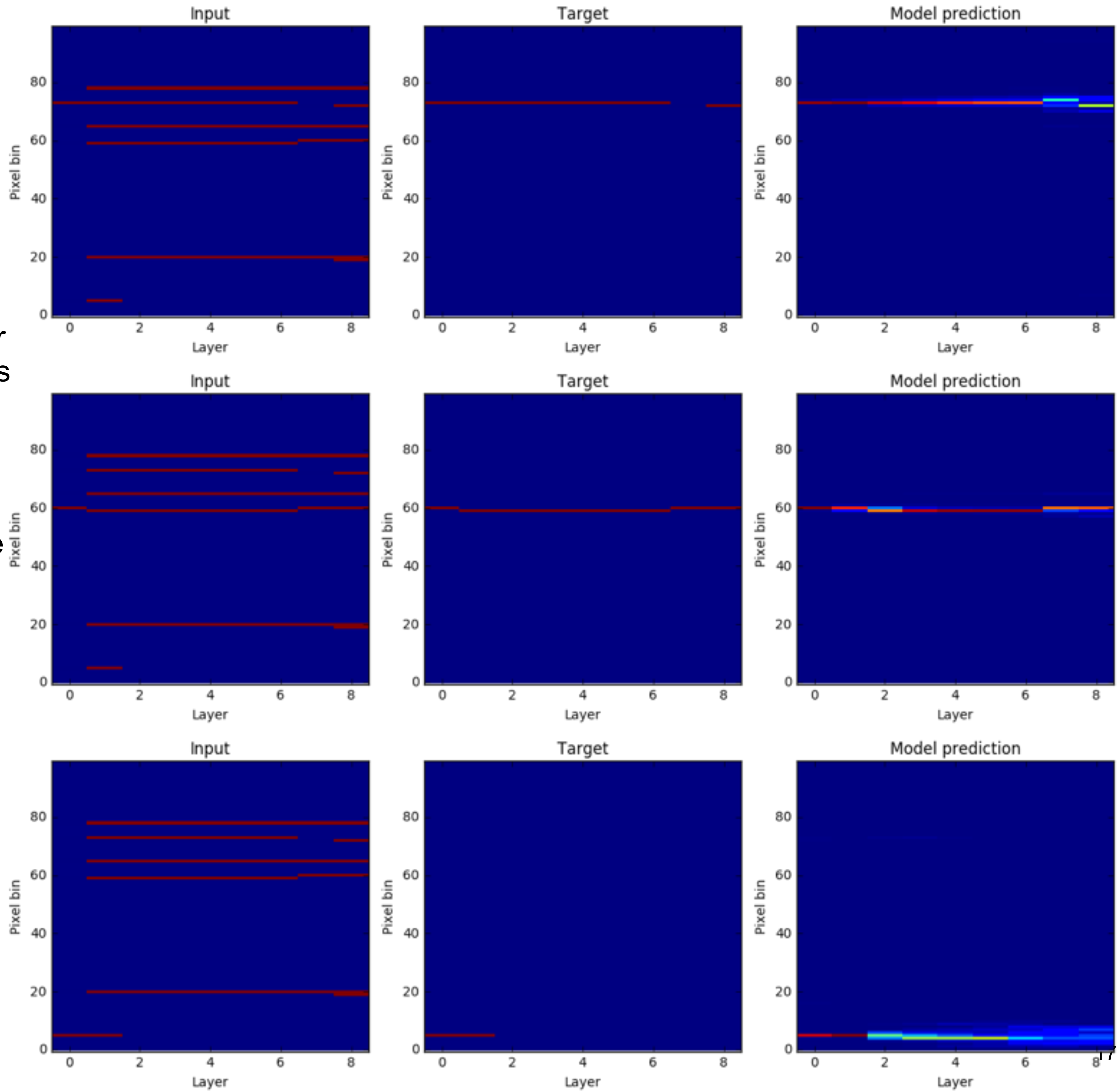
Track finding with LSTMs

- Try to build a single, *seeded* track from a set of hits with backgrounds
- Detector plane pixel arrays fed into the model one at a time
- The model spits out an array of “scores” for that detector plane
  - Pixel predictions (or hit “classification”)
- The LSTM memory is used to carry the dynamic state estimate, updated at each iteration
- The model may consider multiple candidate paths, but hopefully converges on correct one



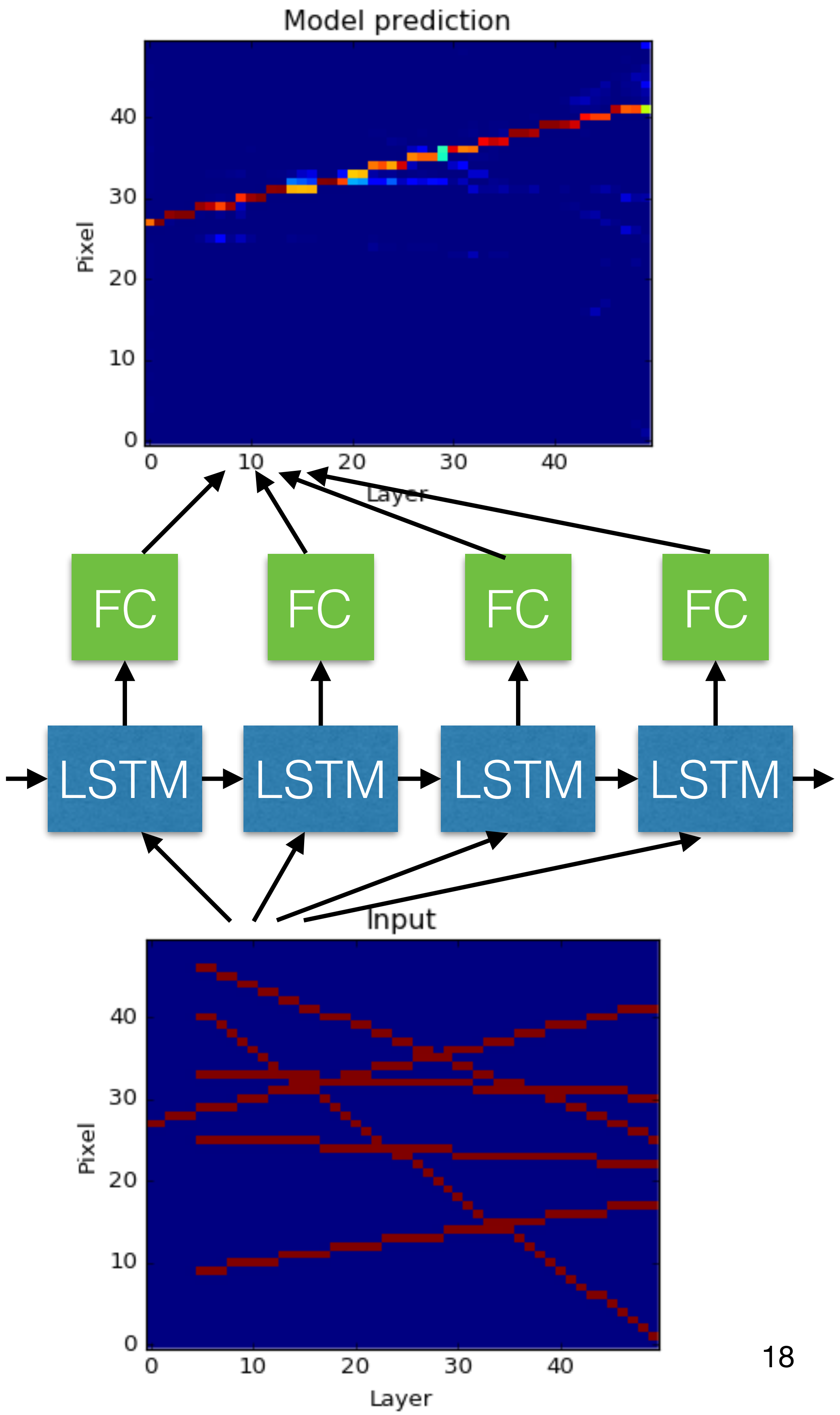
Ramp challenge

- Rebin phi to 200 bins in each layer
- Use first layer hits as seeds
- Loop over seeds, use LSTM to score hits
- For each hit, take best track assignment as label



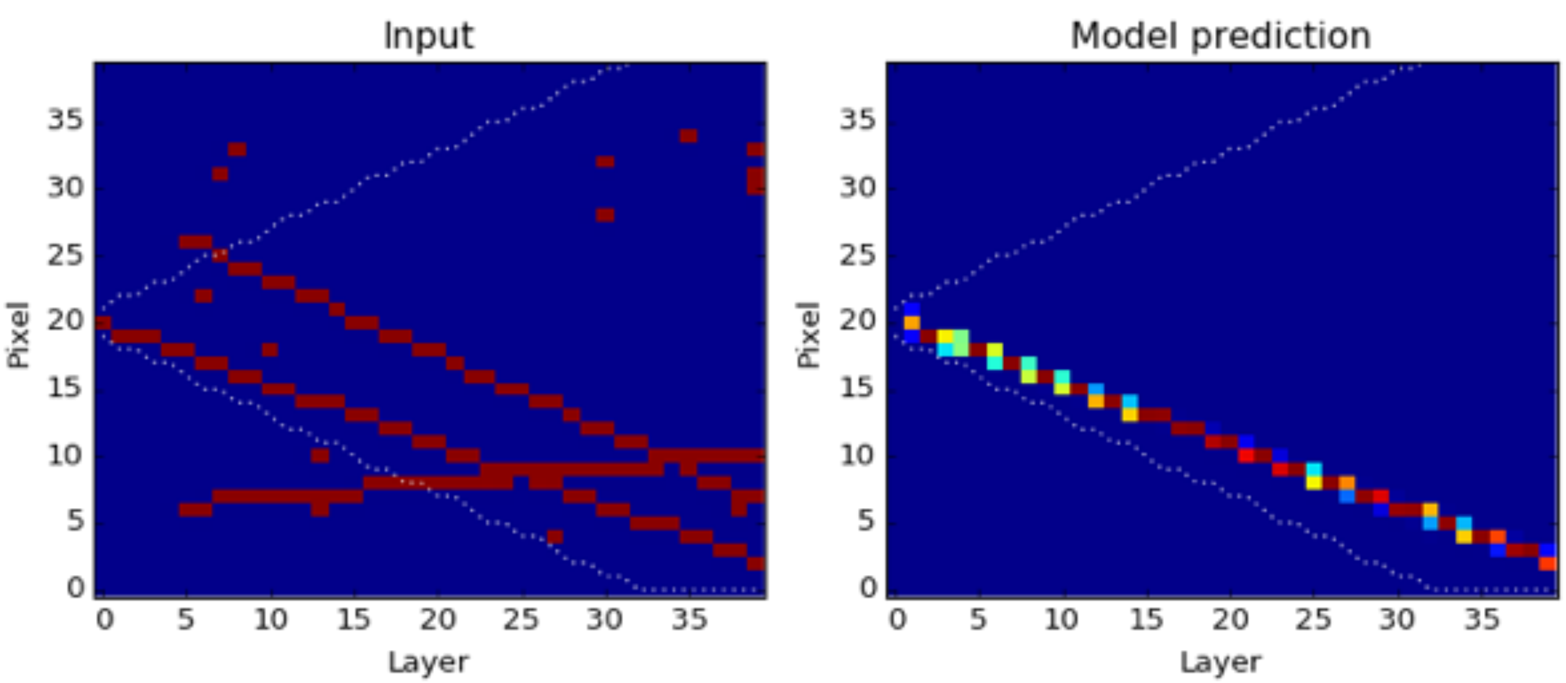
Track finding with LSTMs

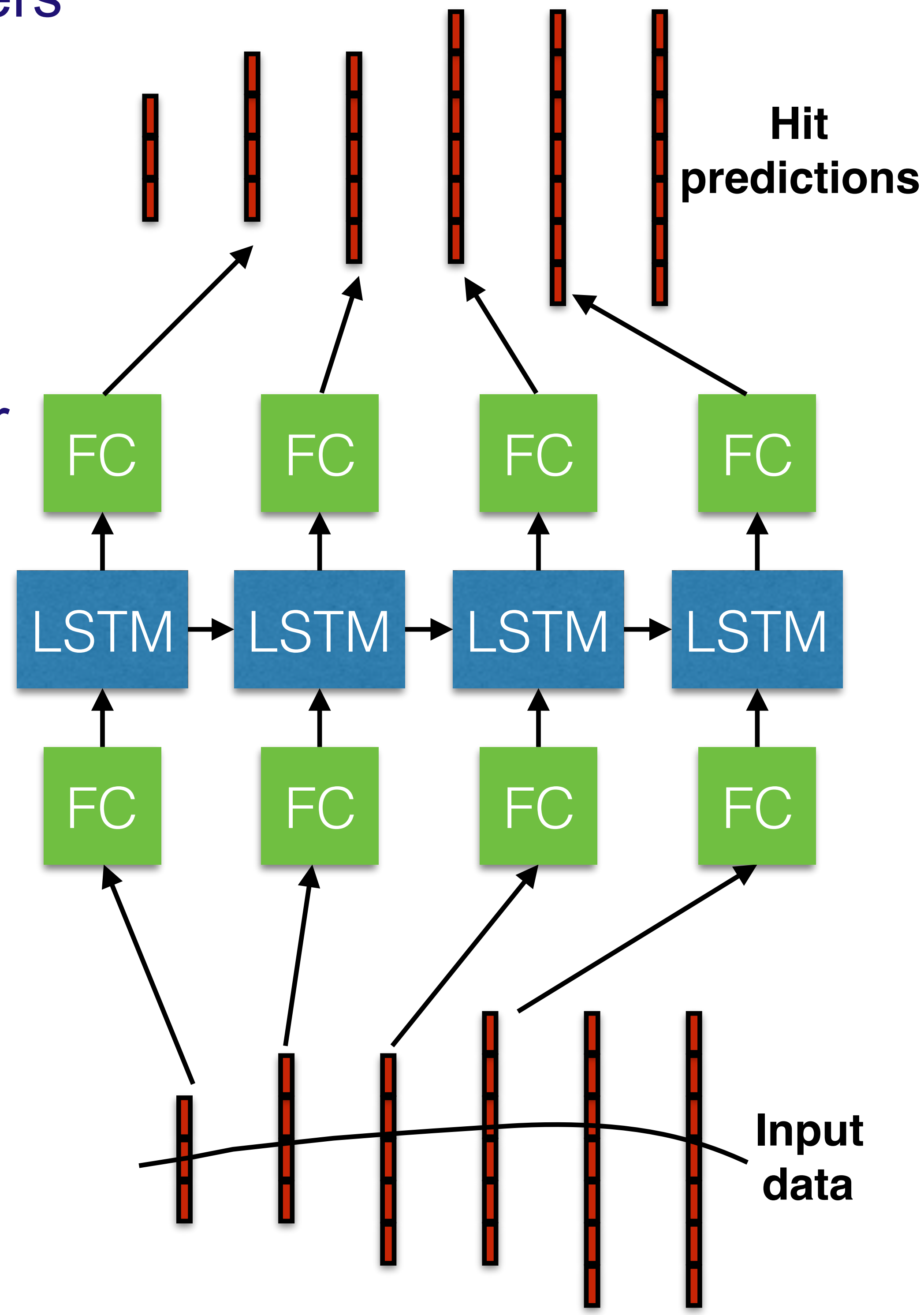
- Try to build a single, *seeded* track from a set of hits with backgrounds
- Detector plane pixel arrays fed into the model one at a time
- The model spits out an array of “scores” for that detector plane
    - Pixel predictions (or hit “classification”)
- The LSTM memory is used to carry the dynamic state estimate, updated at each iteration
- The model may consider multiple candidate paths, but hopefully converges on correct one
- Can be made more effective in several ways
    - Attach regression layer to get track params
    - Iterate multiple times to smooth prediction
    - Multiple tracks at once



Extending to variable-size detector layers

- LHC detector data doesn't come in fixed size layers
  - We have cylindrical layers increasing in size
- We can extend the model by first mapping each layer onto a fixed size latent (embedding) space
- Output transformations correspondingly map a fixed-size prediction onto the target detector layer
- Generate data for this by selecting subset of the square detector data:

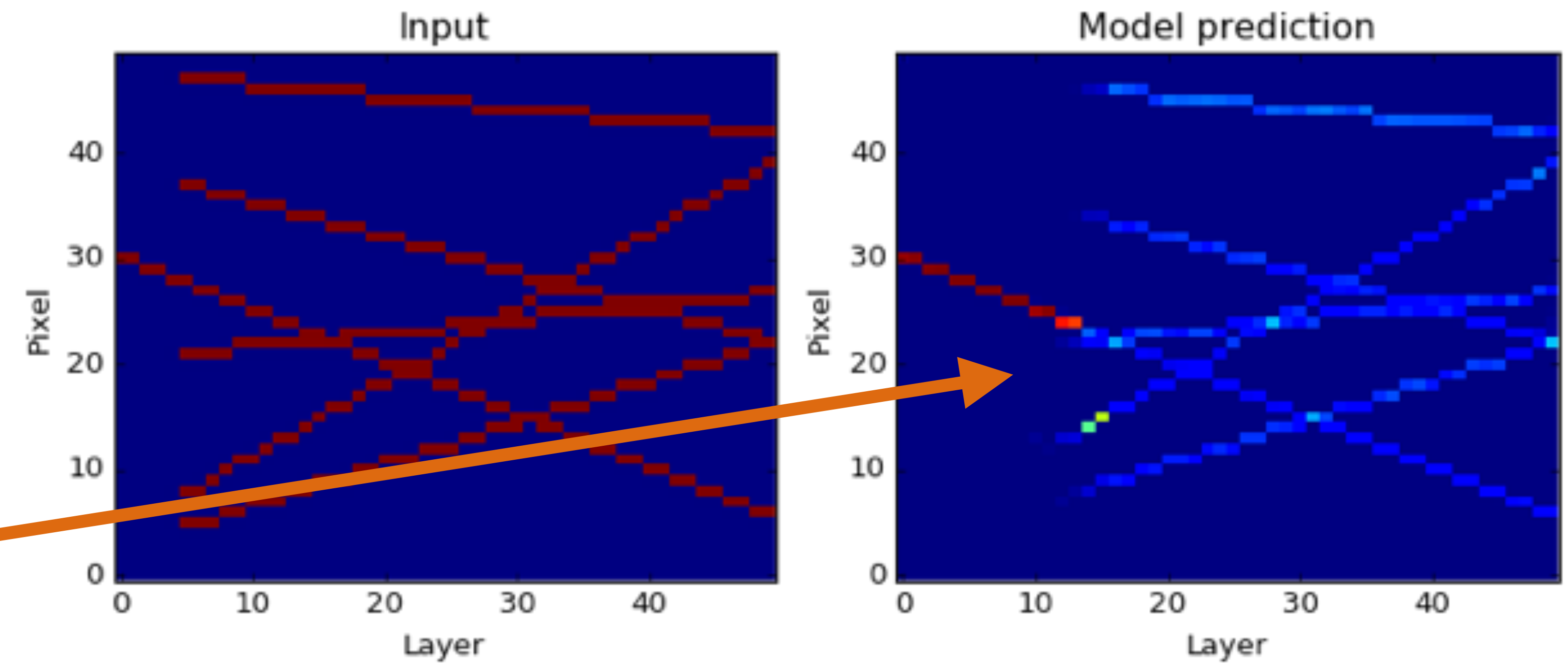




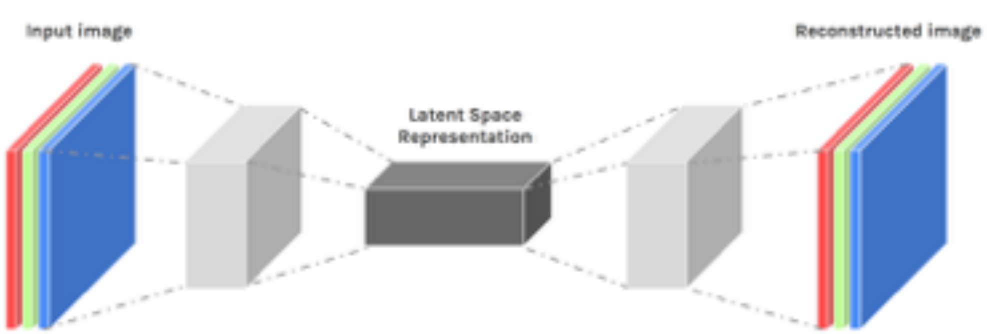
How about convolutional networks?

- Convolutions can also extrapolate and find tracks
- Need to ensure information propagates across entire detector
  - Extrapolation reach can be limited by network architecture

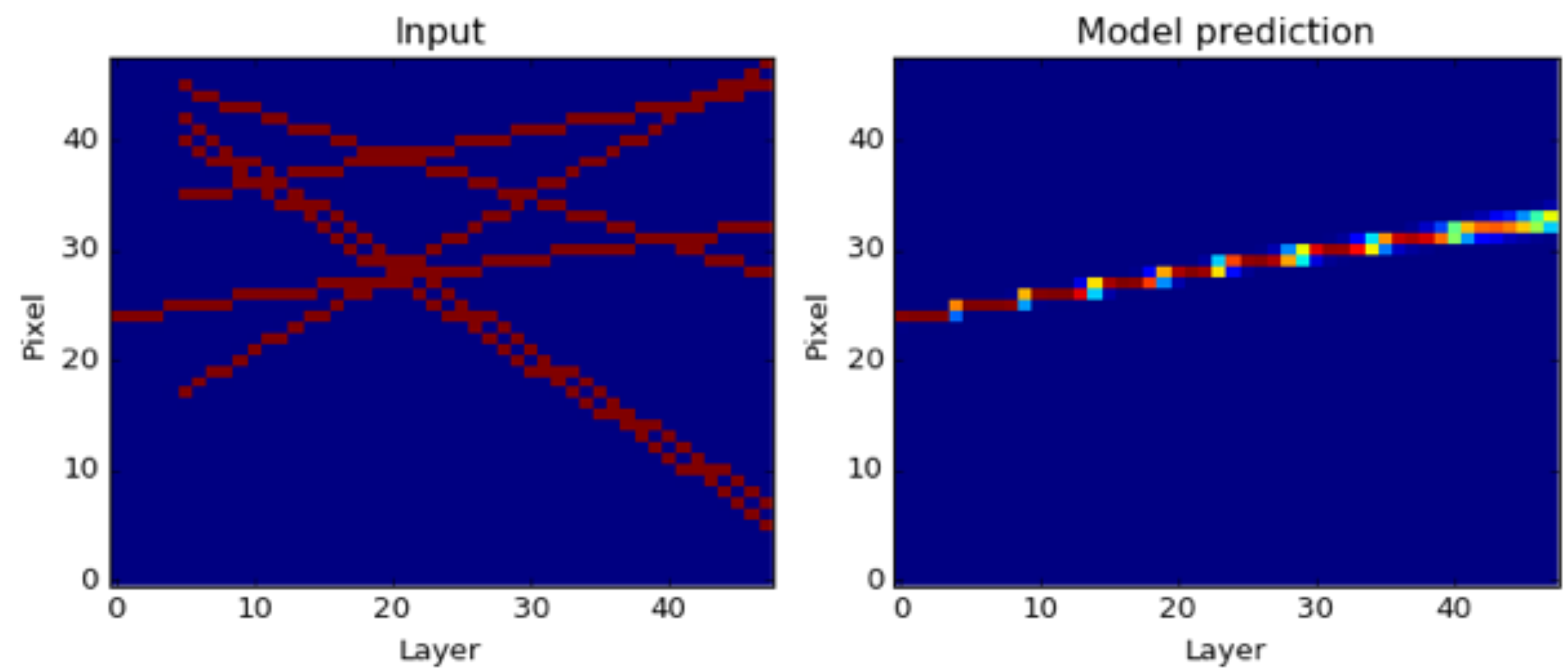
Trained with 10 conv layers, no down-sampling



- Convolutional autoencoder seems to be a good fit

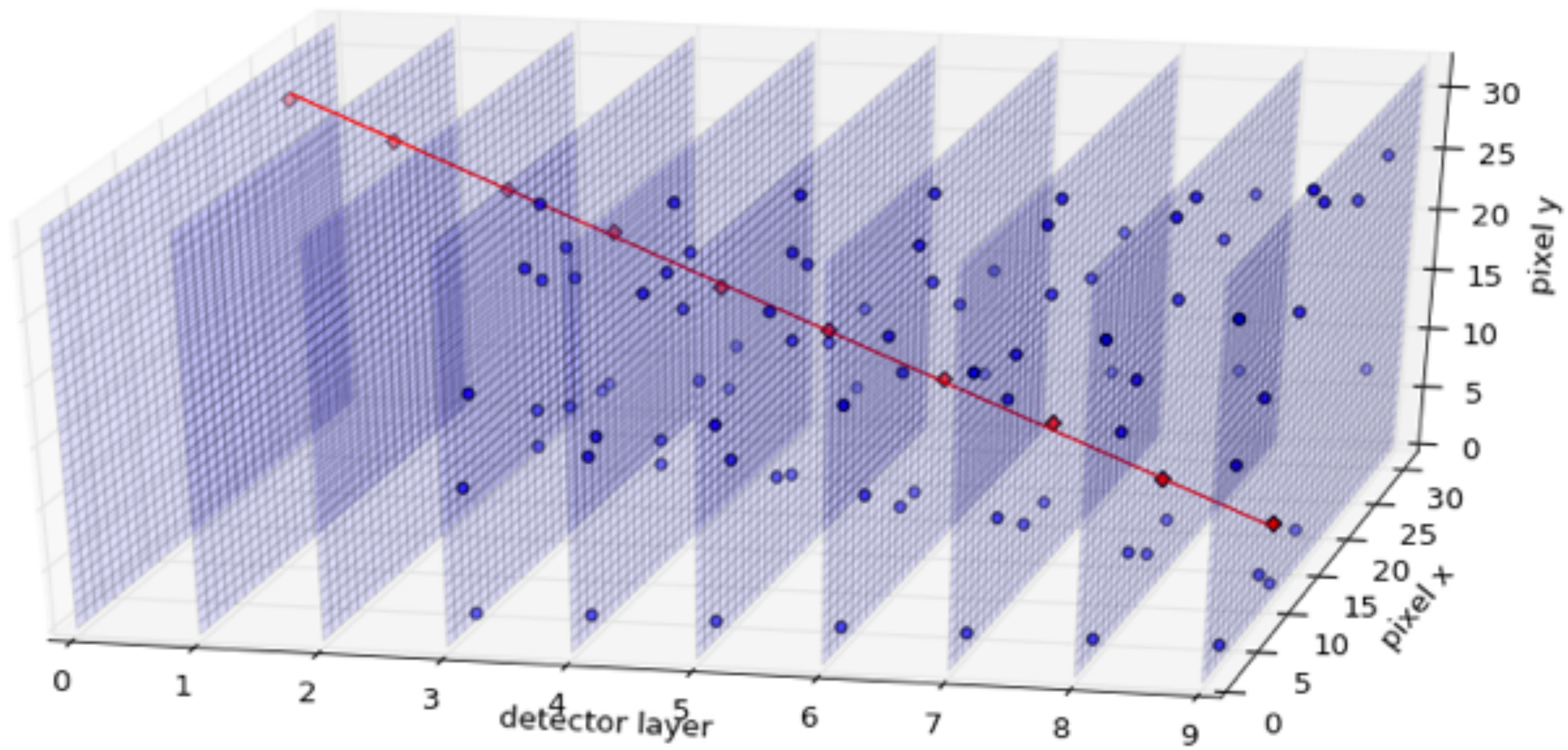


9-layer convolutional autoencoder



https://hackernoon.com/autoencoders-deep-learning-bits-1-11731e200694

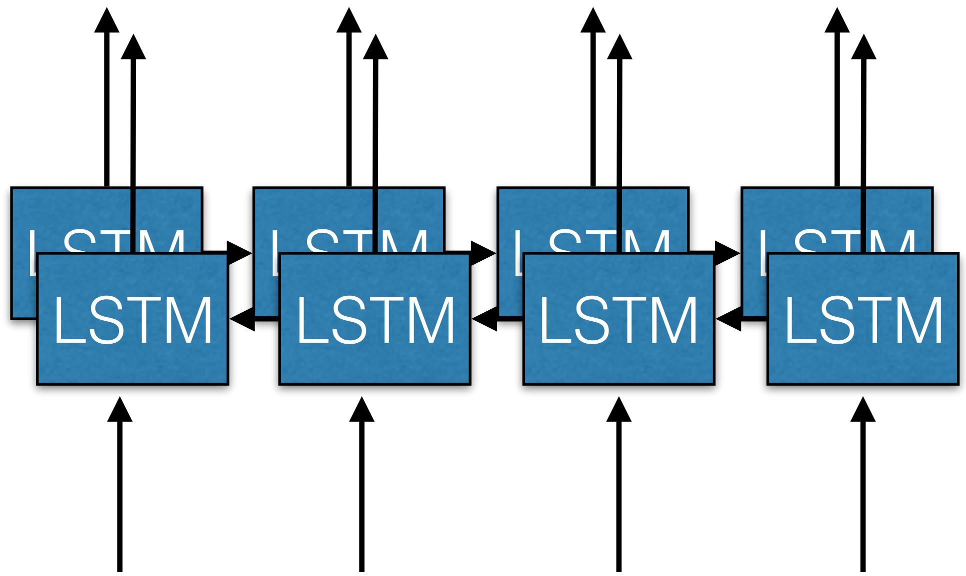
3D toy detector data



- Starting to get a little more "realistic"
  - 10 detector planes, 32x32 pixels each
  - Number of background tracks sampled from Poisson
  - With/without random noise hits
- Adapting my existing models to this data is mostly straightforward
  - Flatten each plane for the LSTM models
  - Use 3D convolution

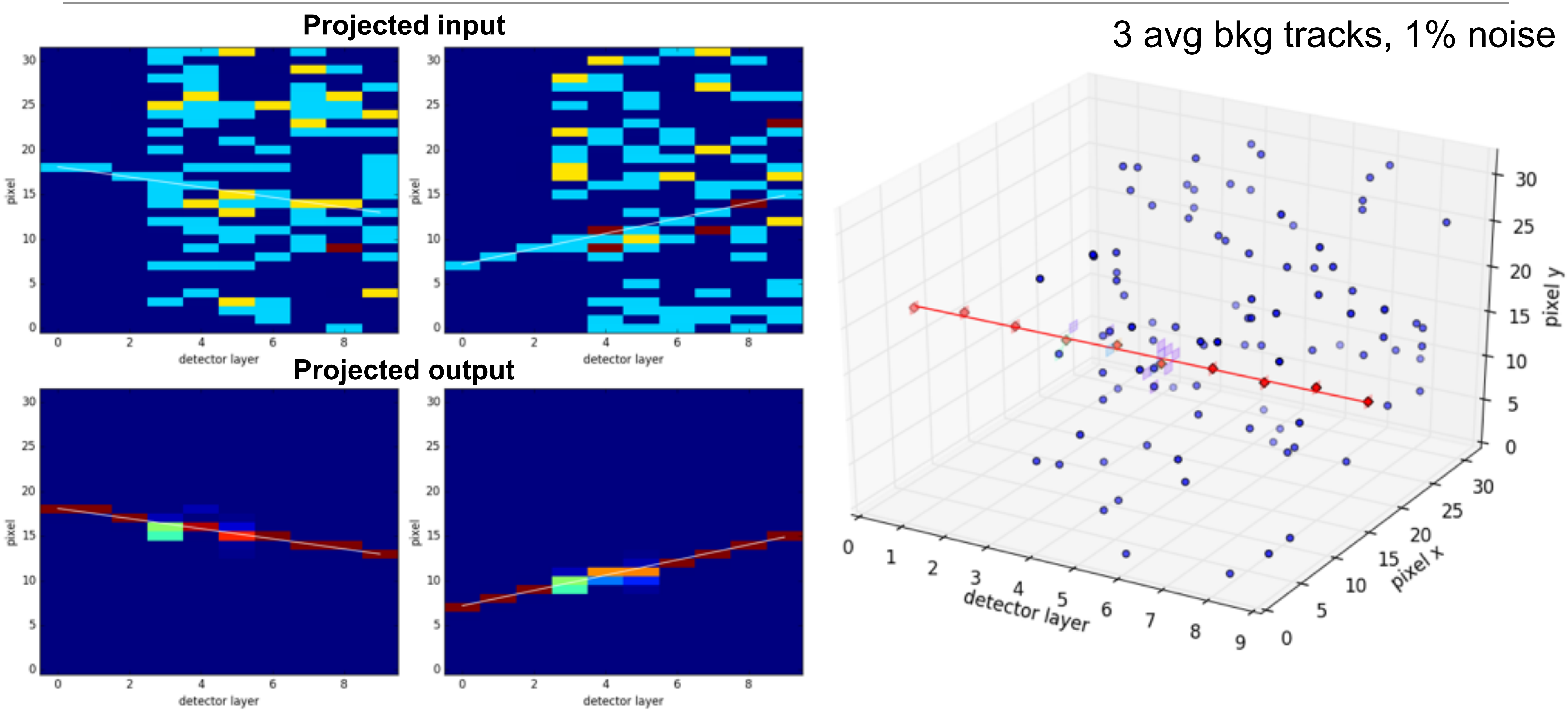
Trying more models

- Deeper LSTM model
  - Adds fully-connected layers before/after the LSTM
- Bi-directional LSTM
  - Adds a second LSTM running over sequence *in reverse*
  - Concatenate the two outputs



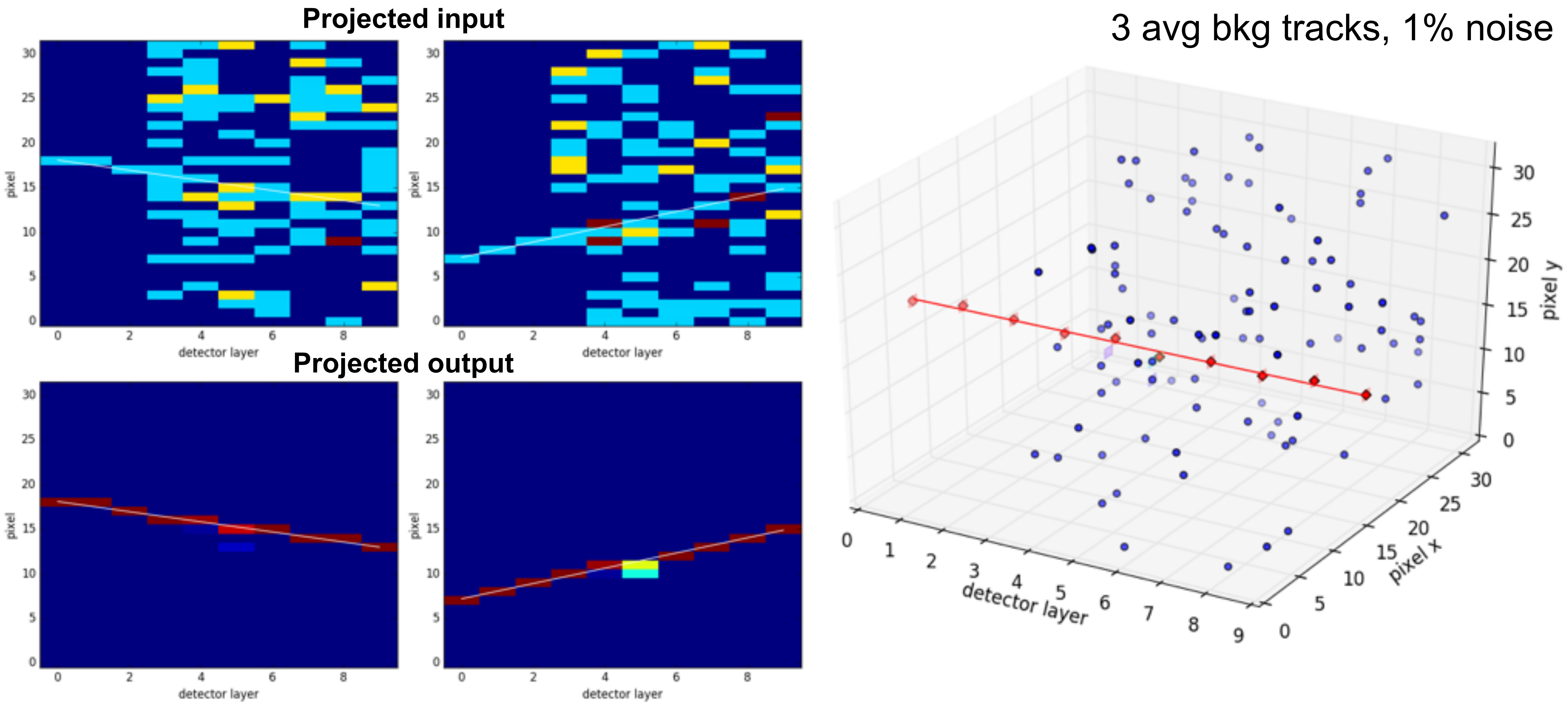
- *Next-layer* LSTM
  - Predict where the hit will be on the *next* detector plane, rather than the current detector plane
  - Basically just an extrapolator, but might be interesting to compare
- 3D convolutional model
  - 10 layers, no downsampling
- 3D conv autoencoder model
  - Uses max-pooling to downsample
  - Decodes with single fully connected layer

LSTM prediction



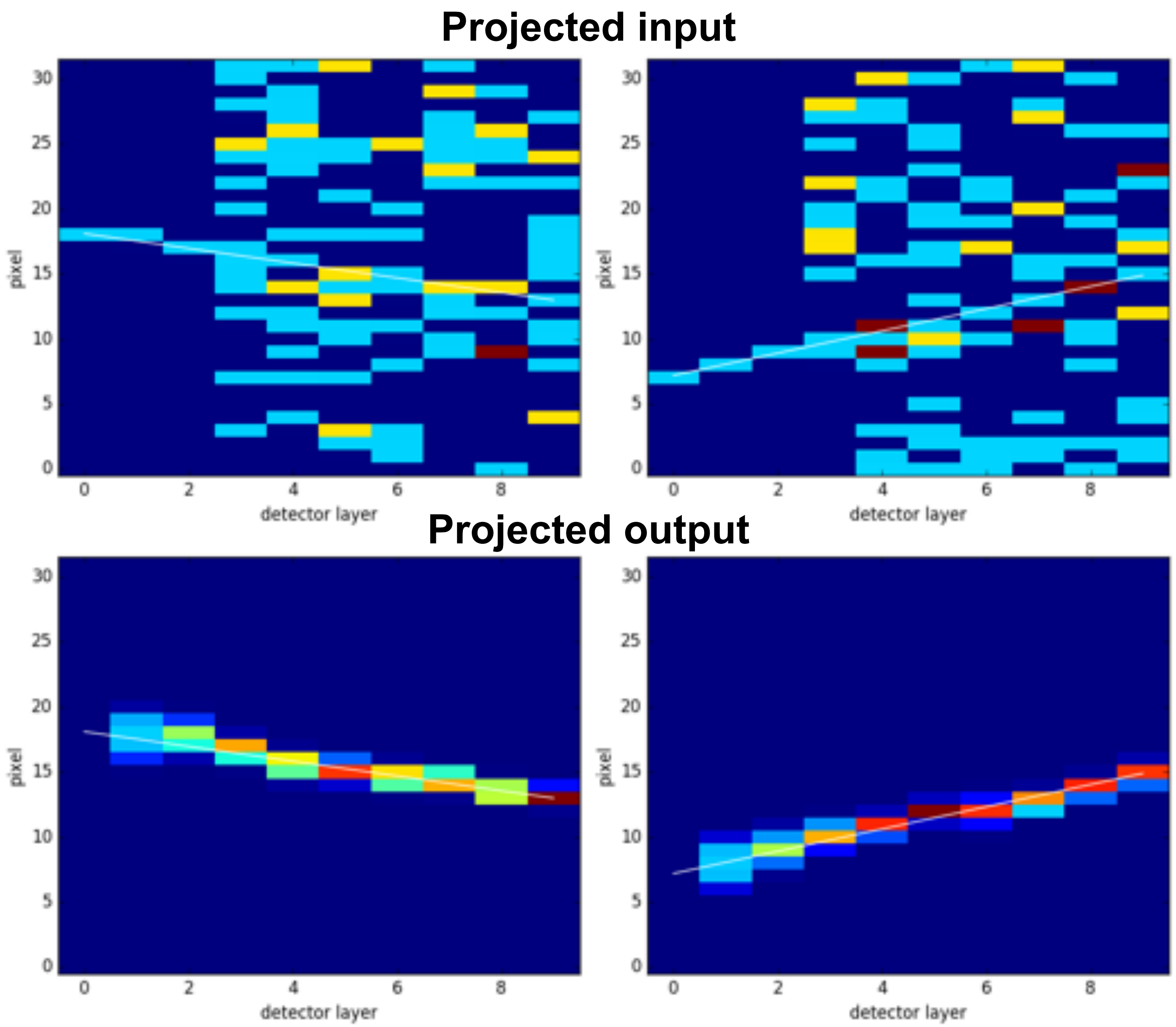
- Sometimes gives predictions that are not smooth
- Occasionally fooled by adjacent hits, though it tends to correct itself

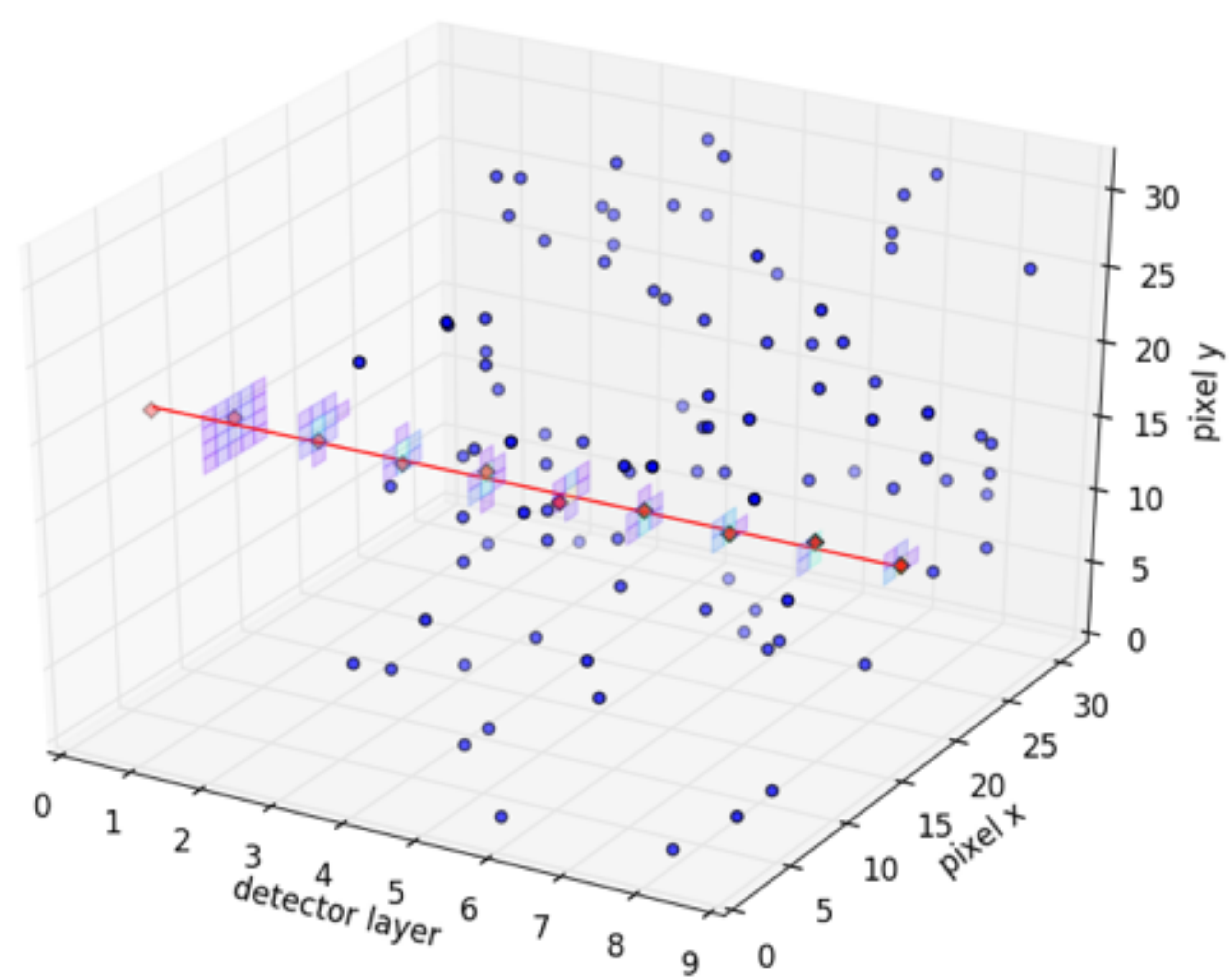
Bidirectional LSTM prediction



- Very precise predictions
  - can see into the future, which presumably helps
- still has few rare artifacts

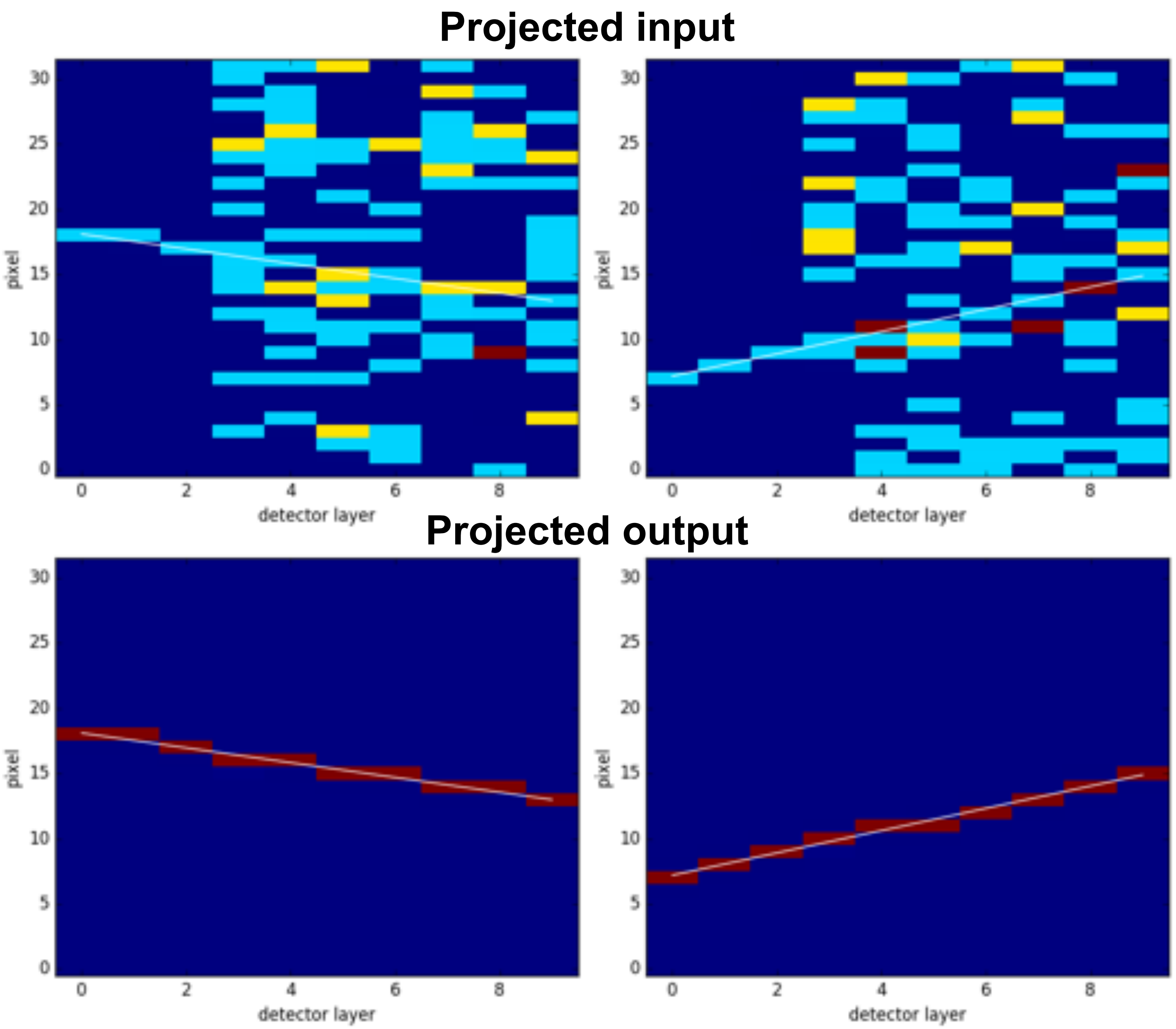
Next-layer LSTM prediction



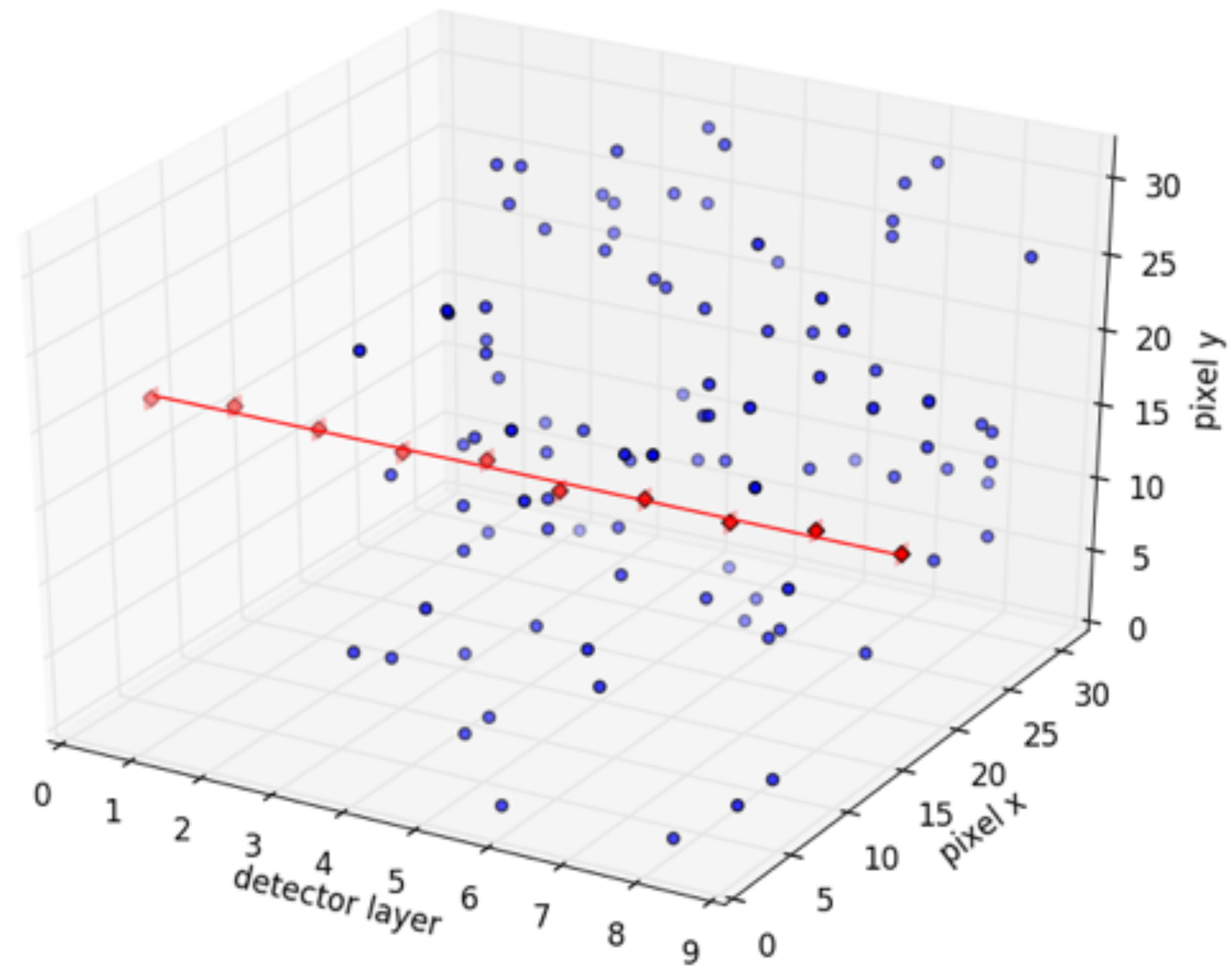


- Next-layer model gives predictions that are less precise but smoother and more accurate
  - Mostly unaffected by nearby stray hits
- With this detector occupancy, they are the best at classifying hits
  - but this may change with higher occupancy

ConvNN prediction

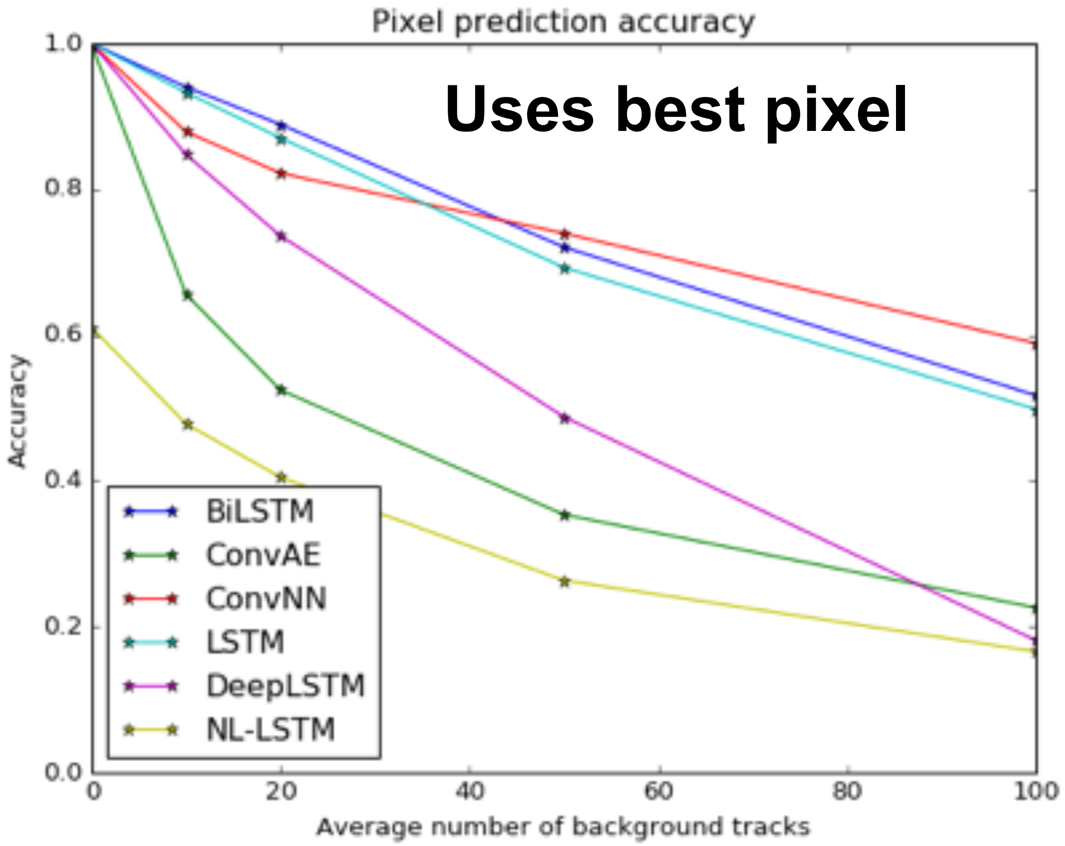


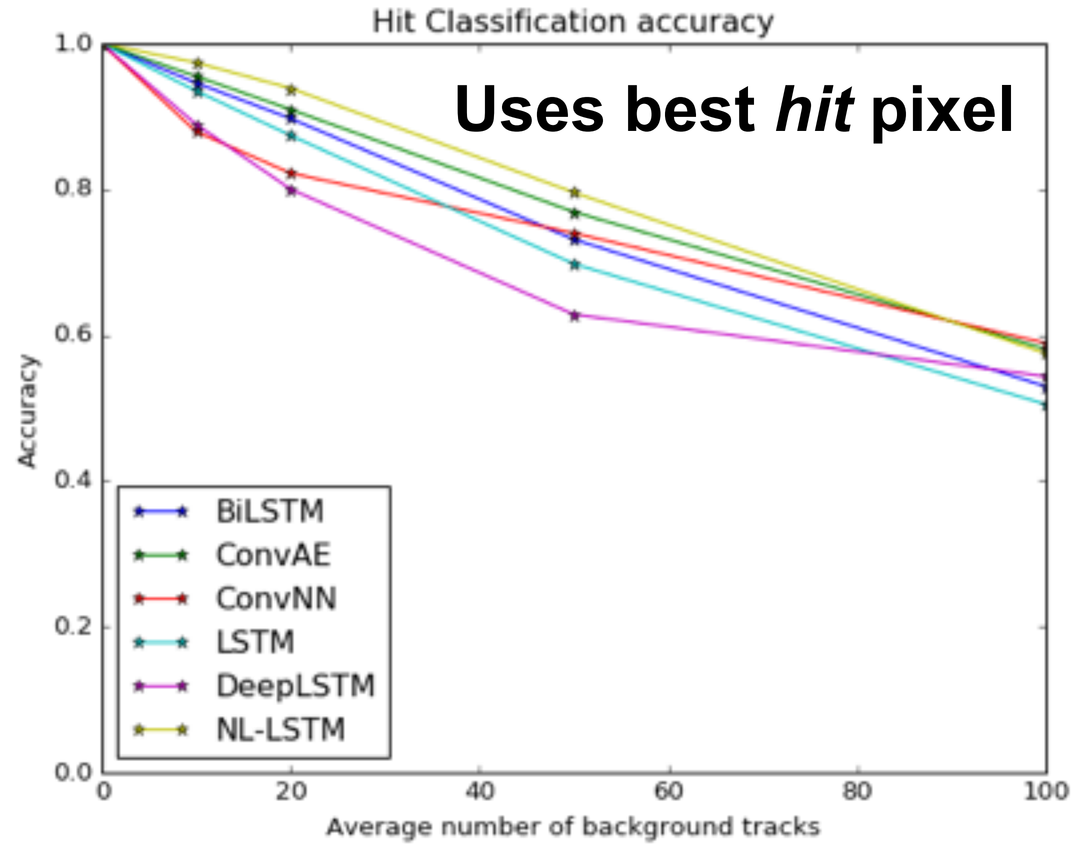
3 avg bkg tracks, 1% noise



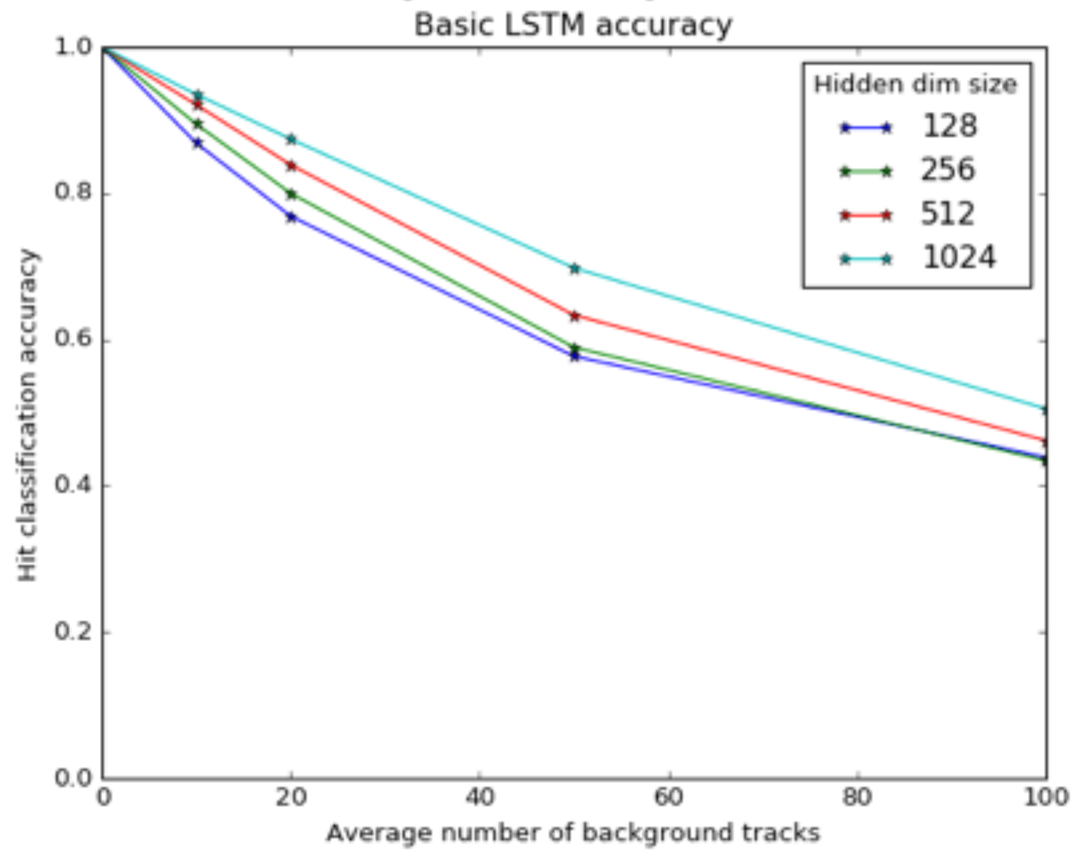
- Simple conv net is clean and precise in this case

Architecture comparisons





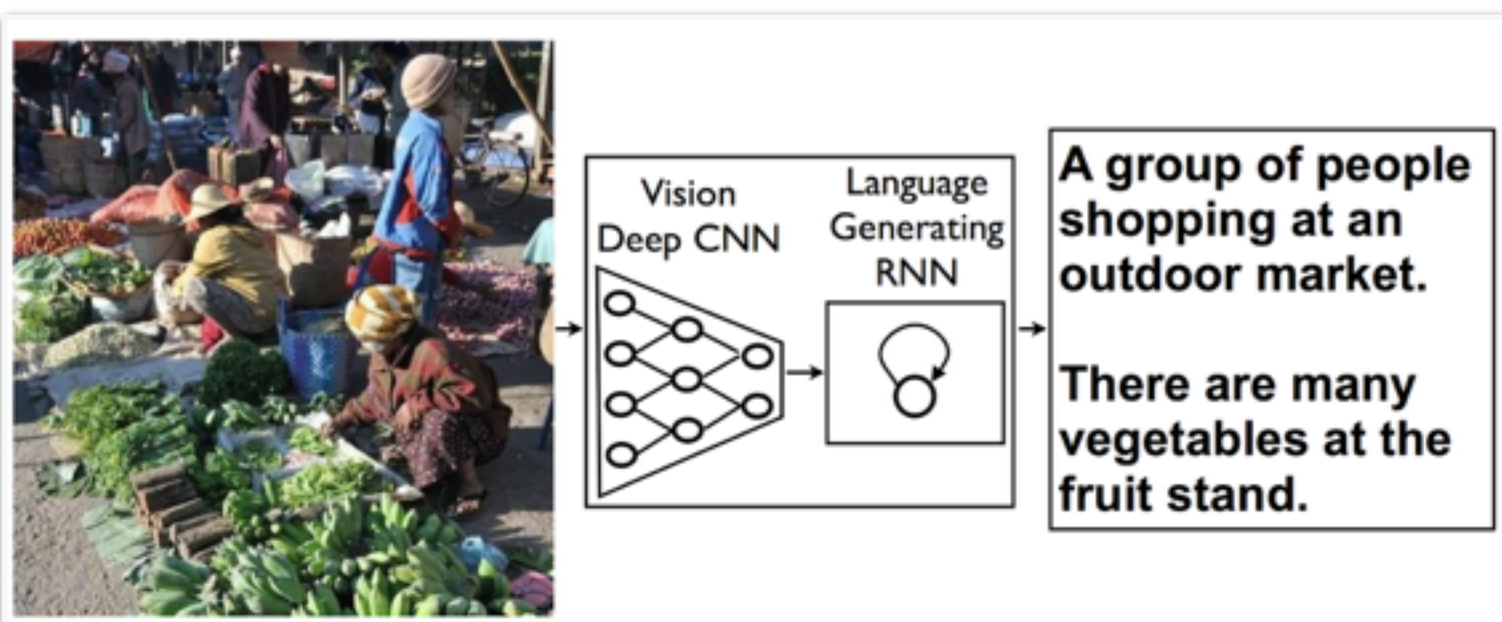
- Models' performance tanks with increasing track multiplicity
  - ConvNN scales the best
- Interesting tradeoffs between the architectures

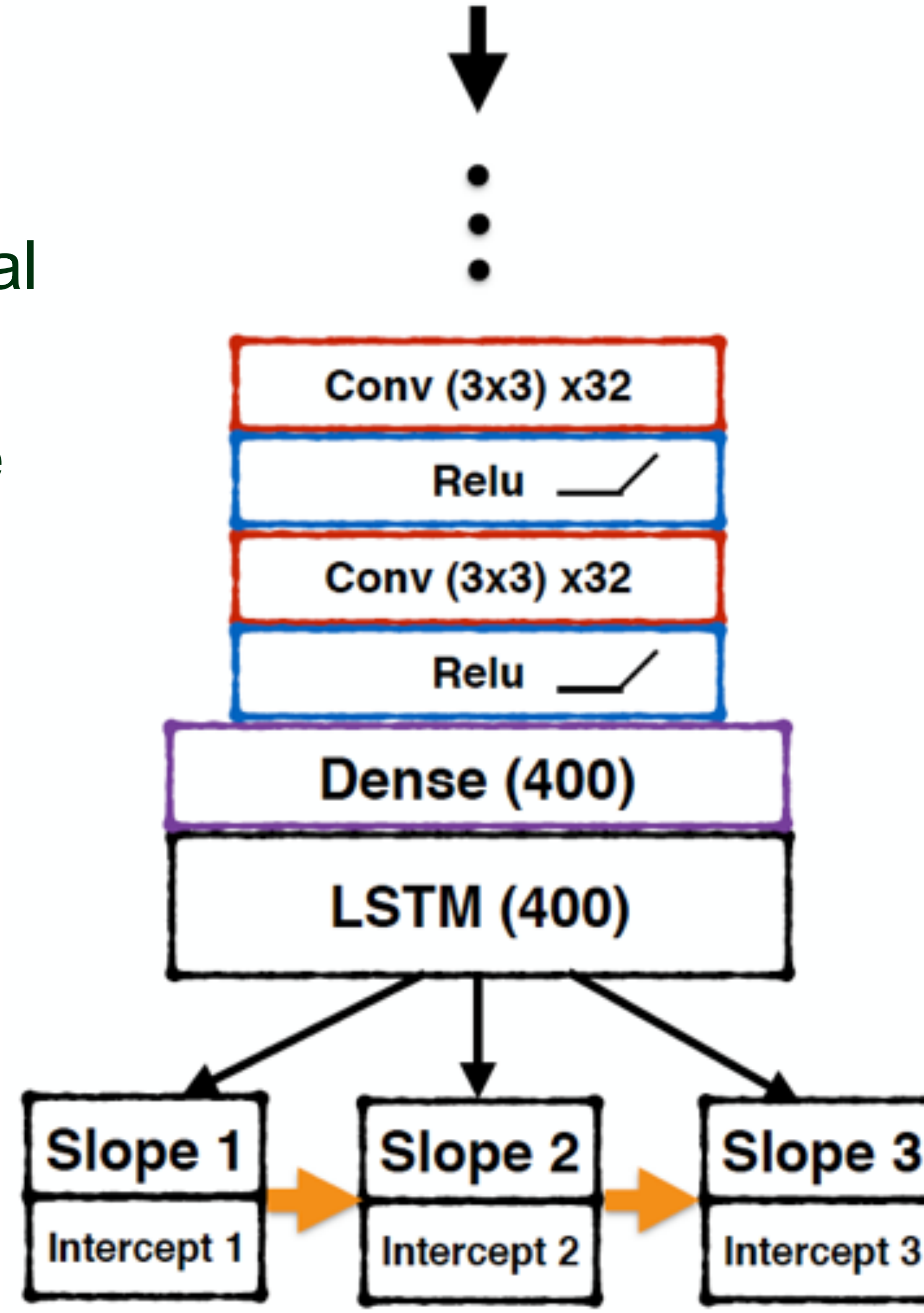


End-to-end track finding

Work of Dustin Anderson

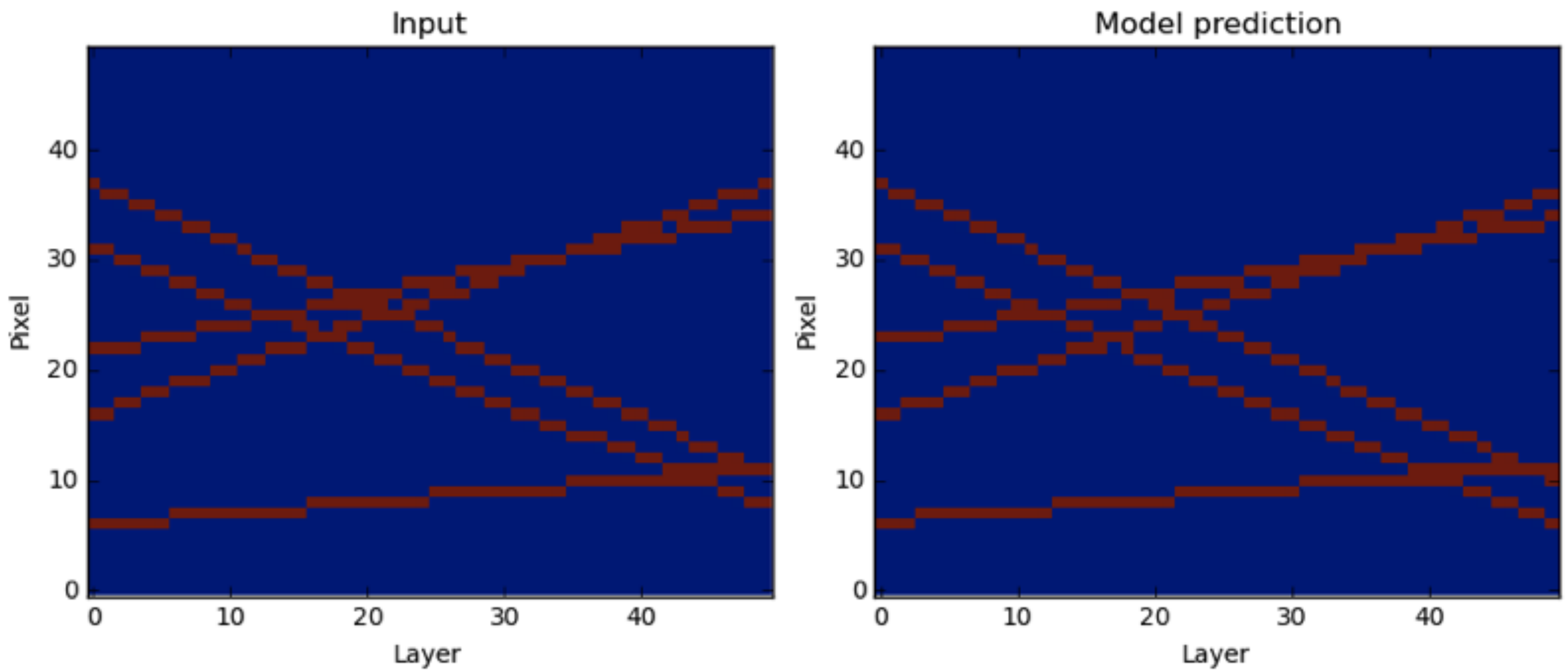
- Can we simply convert raw detector signals into physics quantities?
  - Process the detector “image” with convolutional layers into a *latent representation*
  - Use an LSTM to spit out the parameters of the tracks, one by one!
- Close analogy to the *image captioning problem*





Pixels to track parameters in 2D toy data

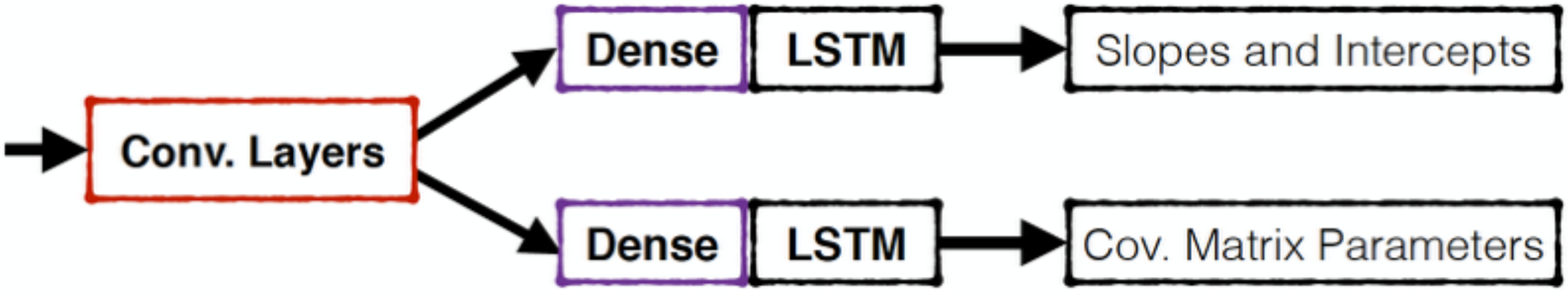
- Sampling number of tracks from Poisson, with a maximum imposed
- Model spits out slope and intercept for each track
- With poisson(3), max=6, give mean validation loss = 1.6



- Work ongoing to implement this with an attention mechanism and also fold in hit assignment

Estimating uncertainties on parameters

- In addition to the track parameters, we would need the covariance
- How do we extend the model to spit out reasonable uncertainties?
  - Add additional output to model for the covariance matrix:



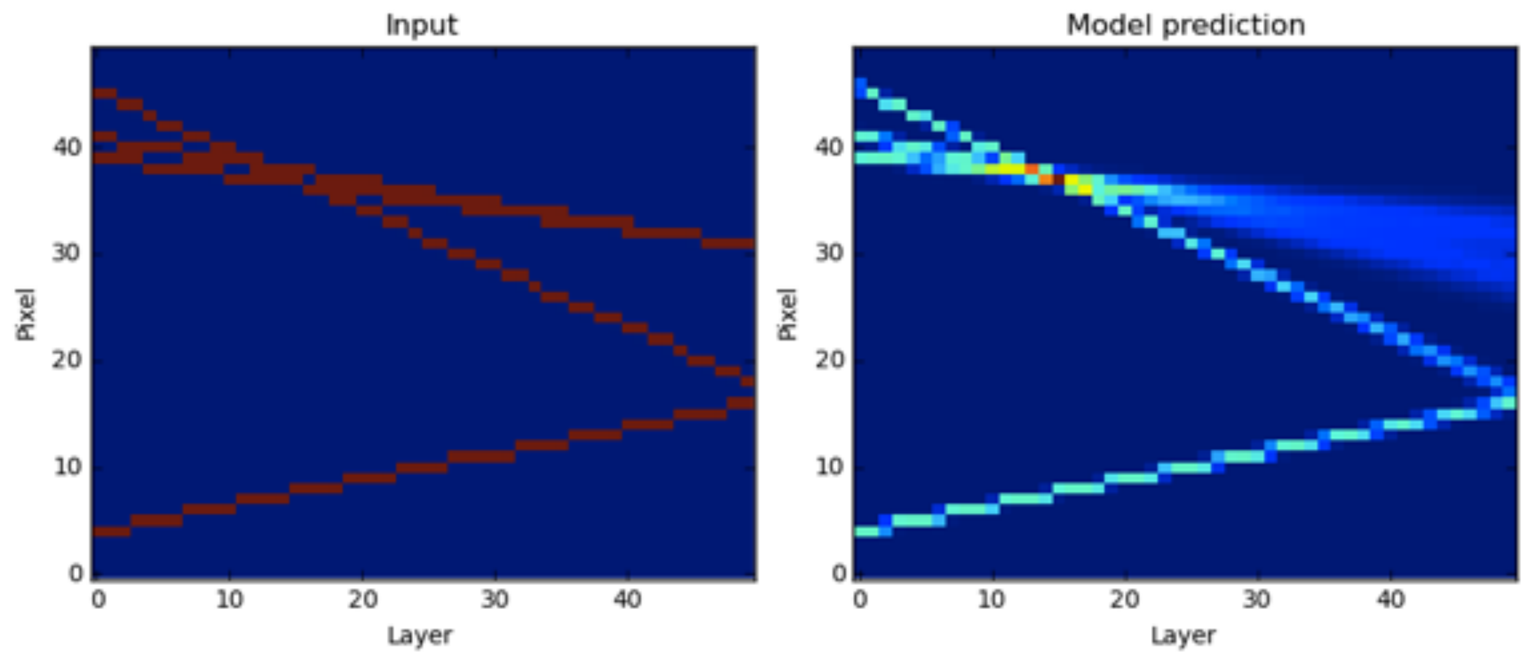
  - Replace mean-squared-error loss function with a log gaussian likelihood:

$$L(\boldsymbol{x}, \boldsymbol{y}) = \log|\boldsymbol{\Sigma}| + (\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x}))^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x}))$$

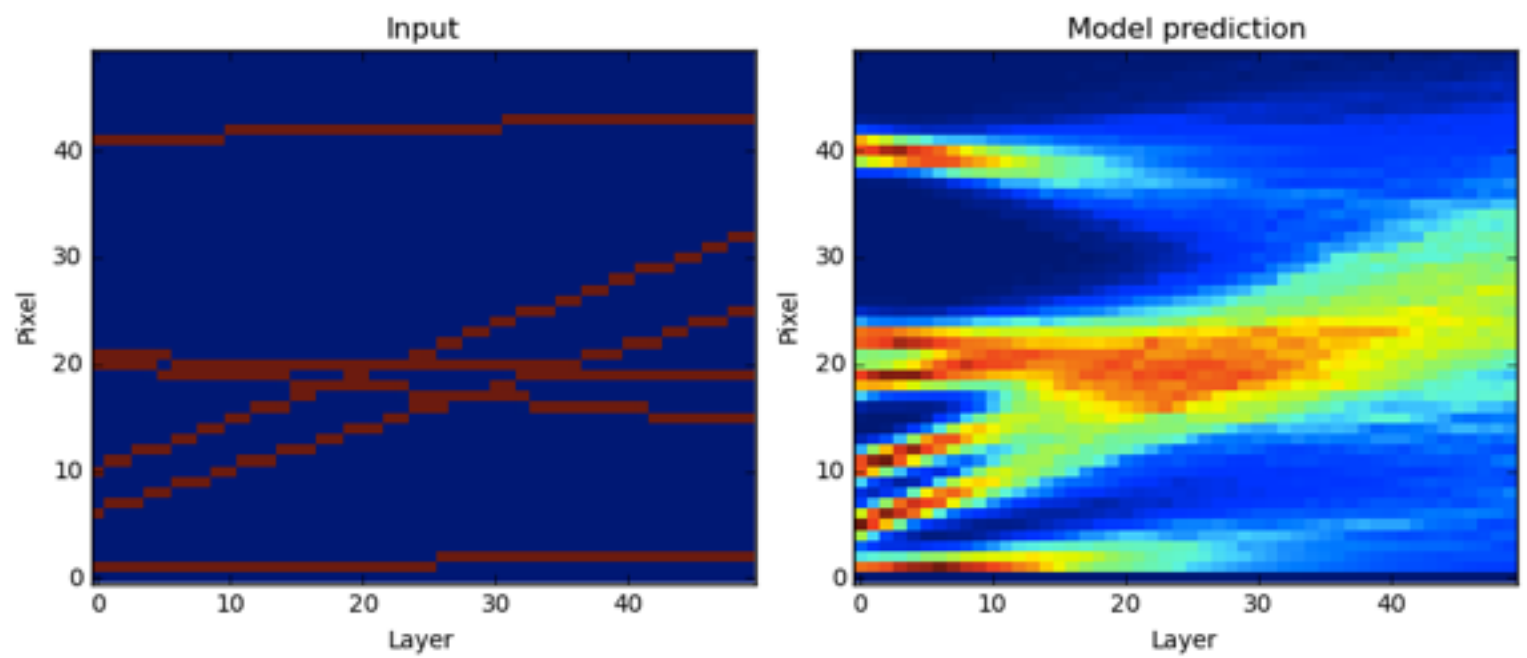
Minimize this during training

Estimating uncertainties on parameters

- We can visualize the uncertainties on the predictions



- However, it does get unstable with large numbers of tracks

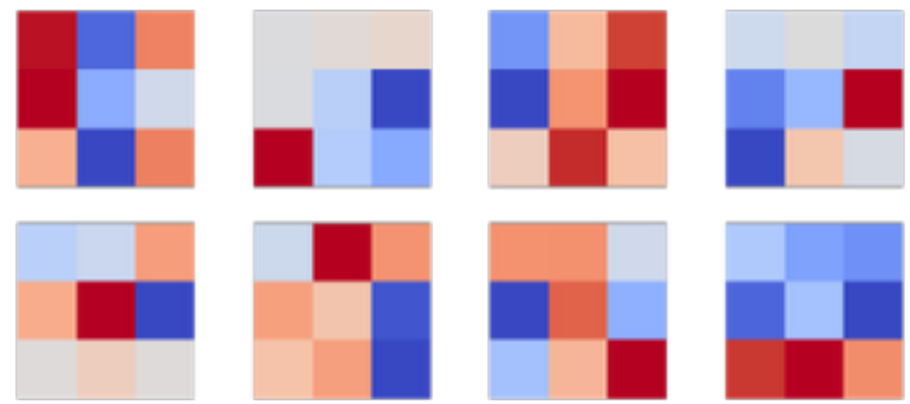


Improvements in development

Visualizing convolutional networks

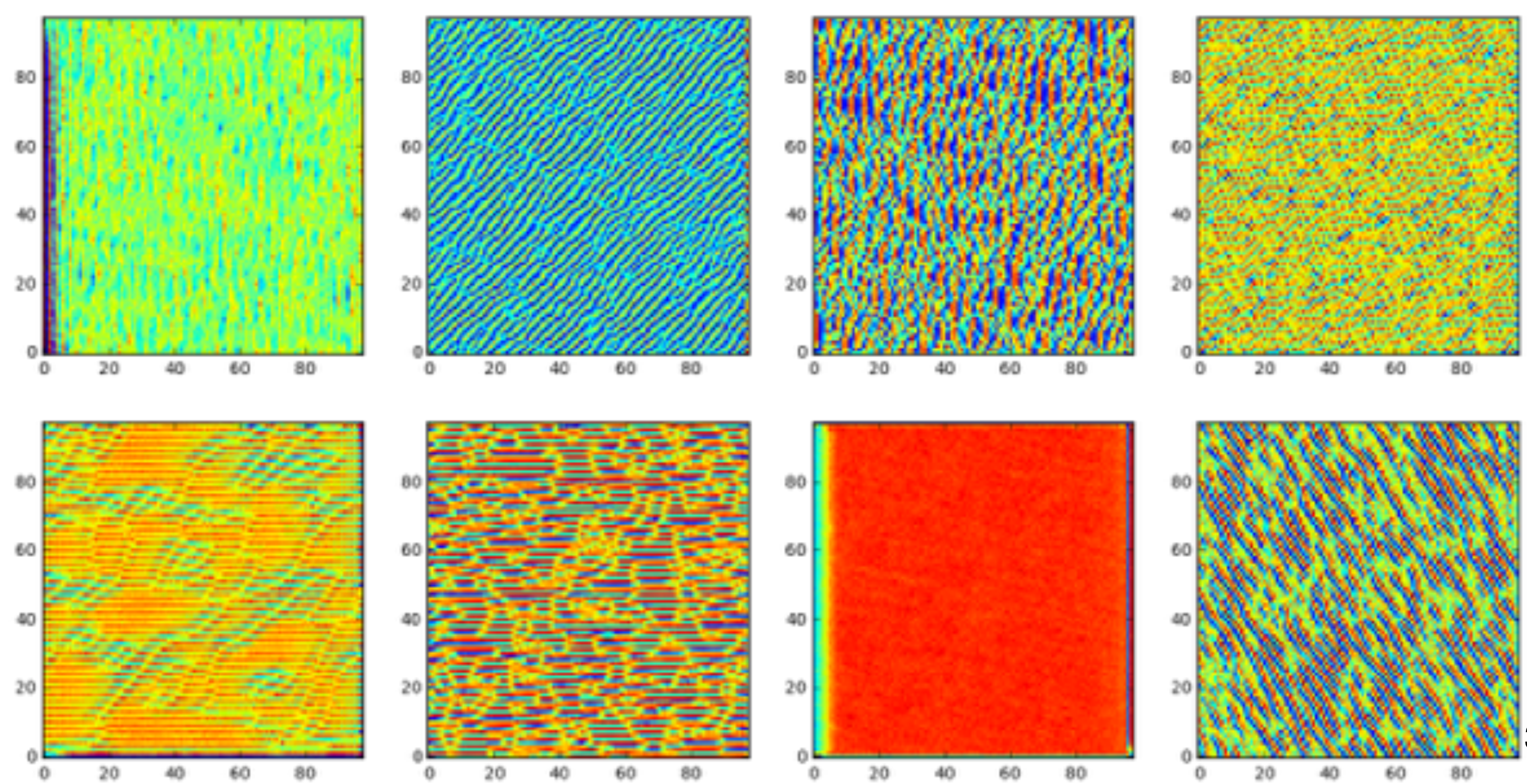
- First layer filters don’t really look like track stubs, as intuition might suggest
  - The model instead learns something abstract, probably more compact

From the 2D conv autoencoder hit classifier



- We can iteratively optimize input images for specific filters, letting us visualize what kinds of features the network is looking for:

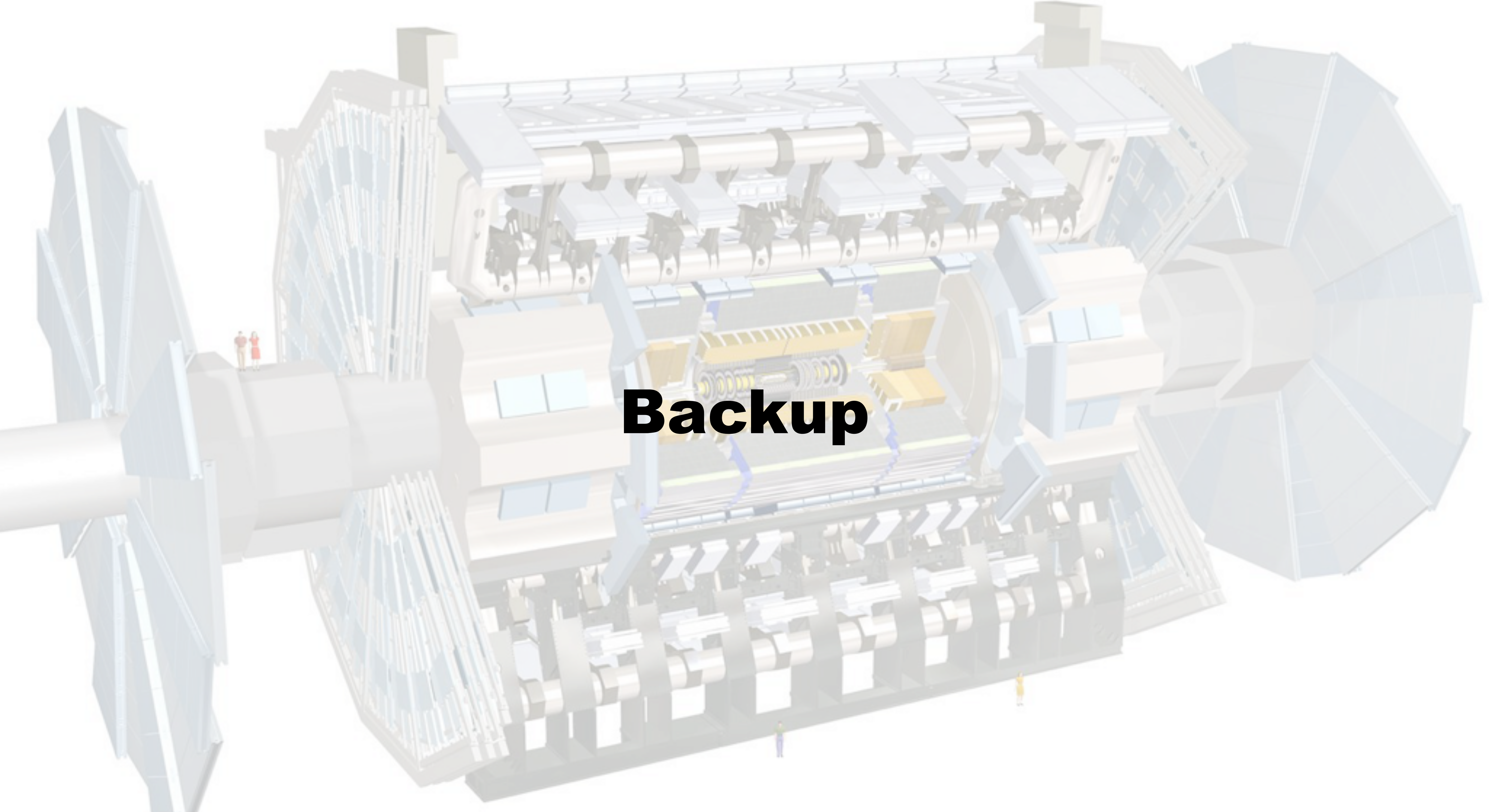
From the 2D track parameter estimator model



Conclusion

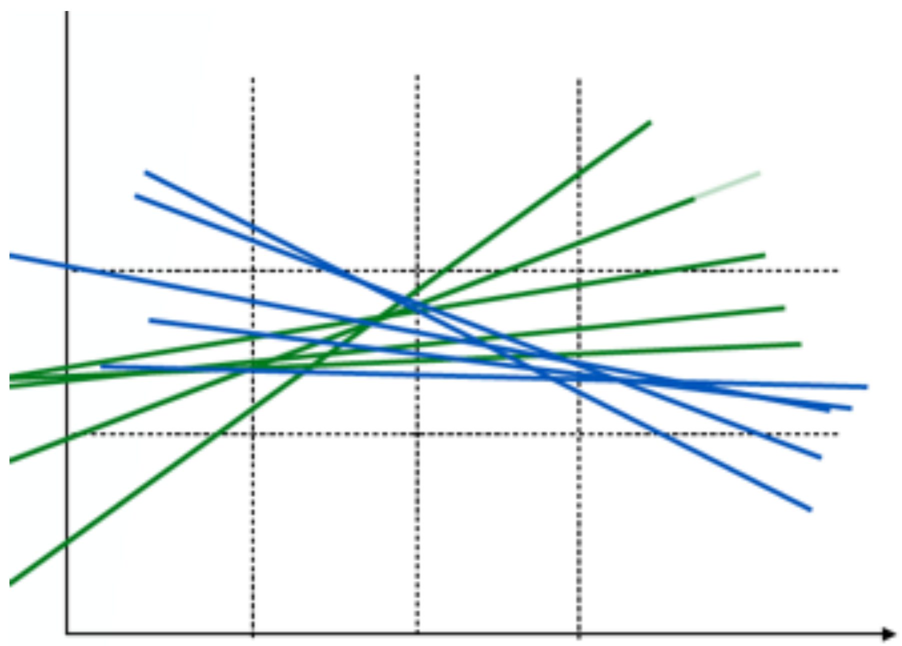
- The HEP.TrkX project was formed to investigate ideas for applying machine learning algorithms to the problem of HEP tracking
  - We’re still in an exploratory phase, testing things out, having fun
- A number of ideas have been demonstrated already on *very simple* toy data
  - LSTM and convolutional networks for track finding
  - End-to-end track finding with Conv + LSTM
  - Other things I haven’t covered today
- Our game plan for the next few months:
  - Increase complexity and realism of the problem (e.g., ACTS data)
  - Converge on a small number of ideas to explore *in depth*
  - Compare to reasonable baselines (e.g. Kalman filter) in performance and complexity
- Pay attention for our future results!

Backup

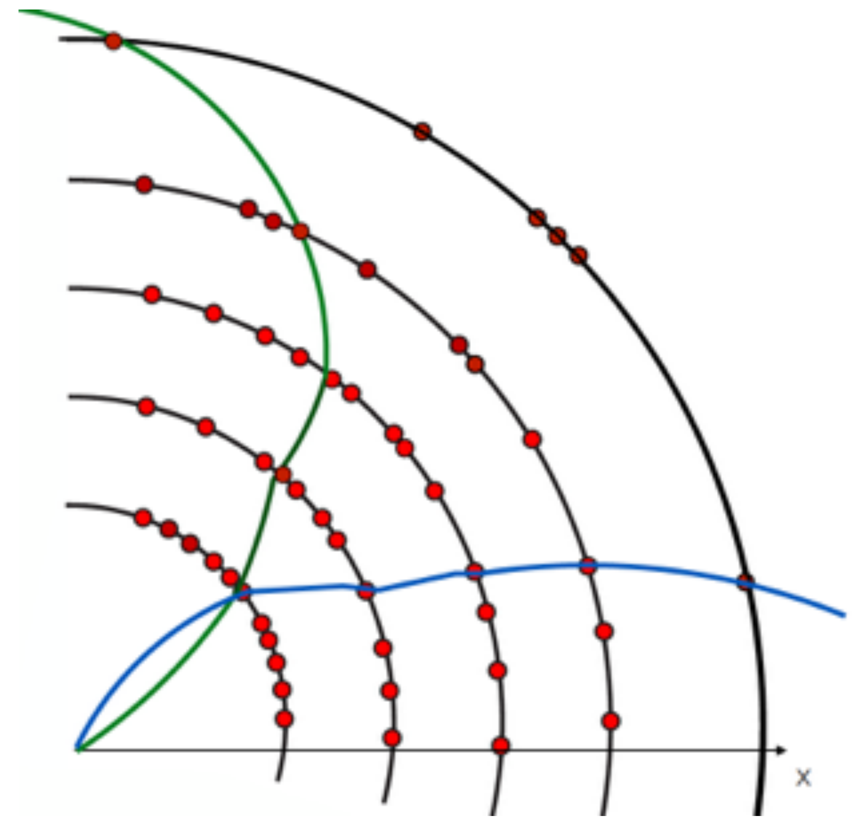


Other ideas - data transforms

- Hough Transform breaks down in LHC-like data due to process noise and high occupancy



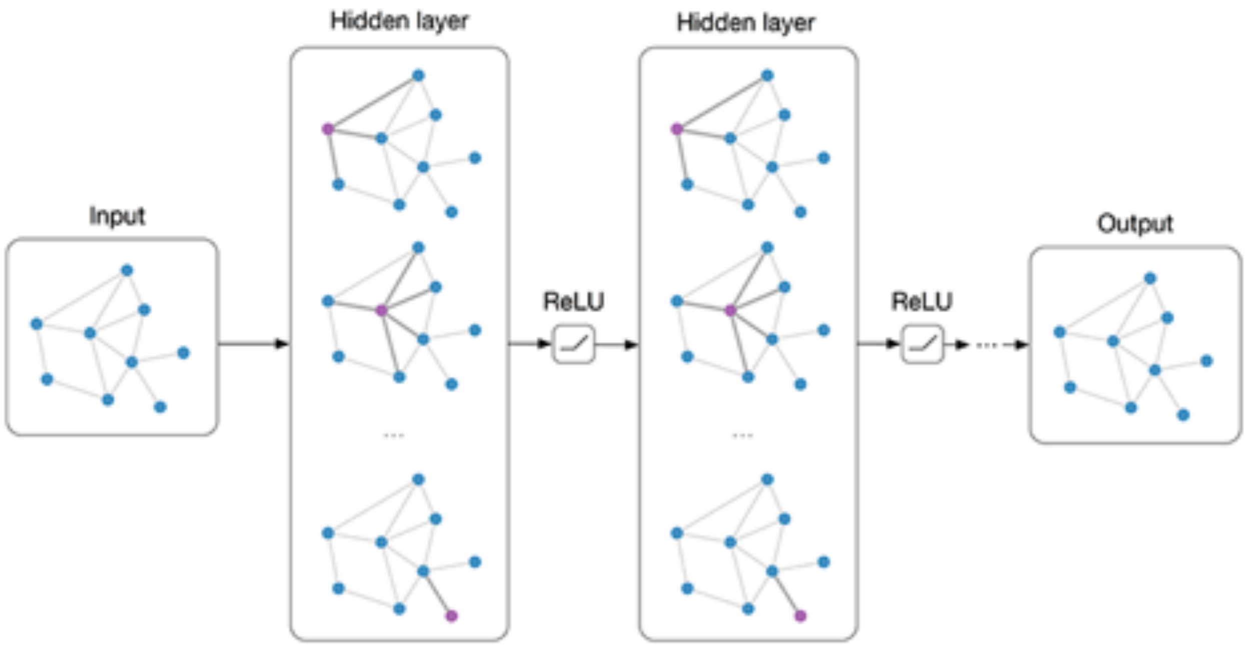
parameter space



- But what if a deep network could *learn* a mapping to group together hits that belong to the same track?
  - You don’t need to impose a specific representation
  - The model could take event context into account

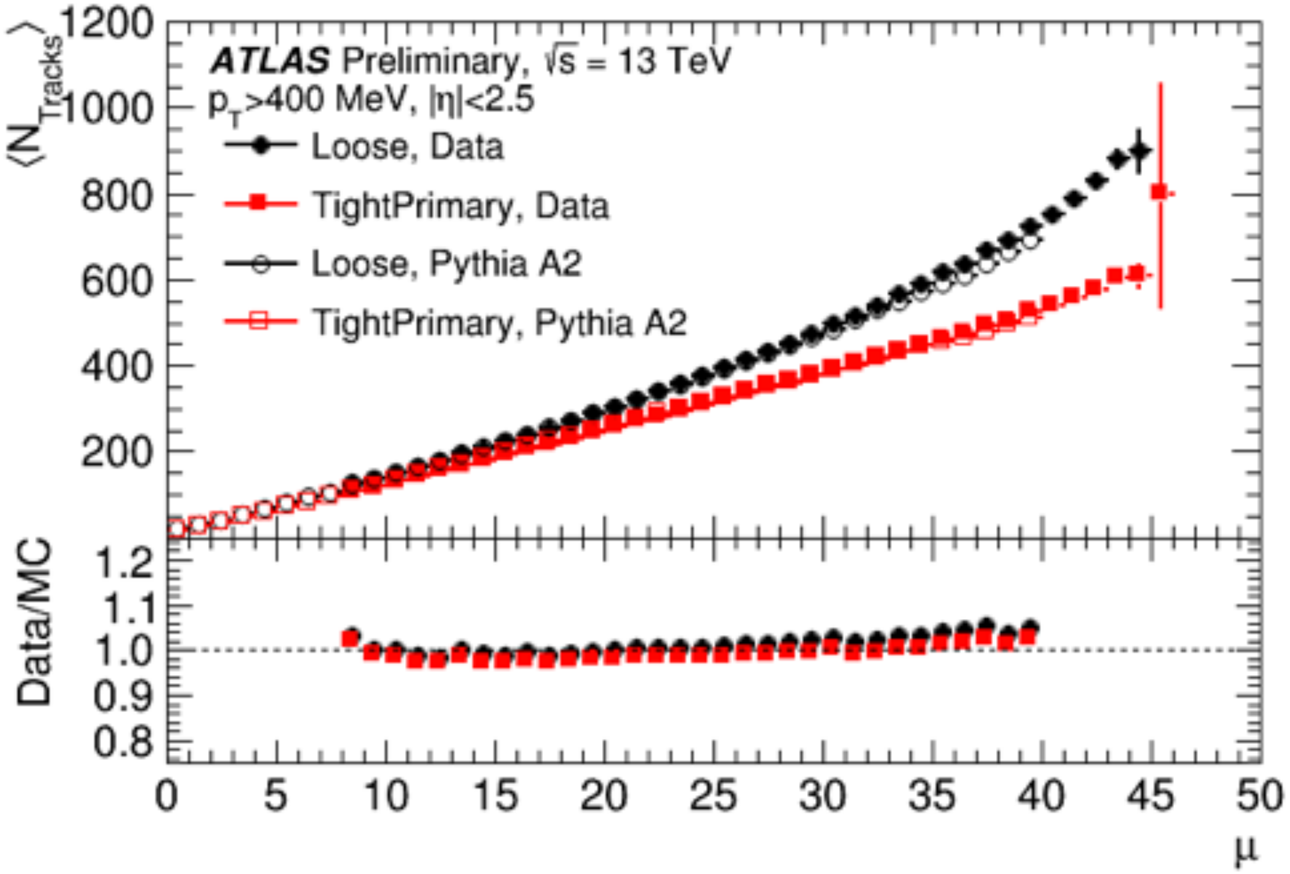
Other ideas - graph convolutions

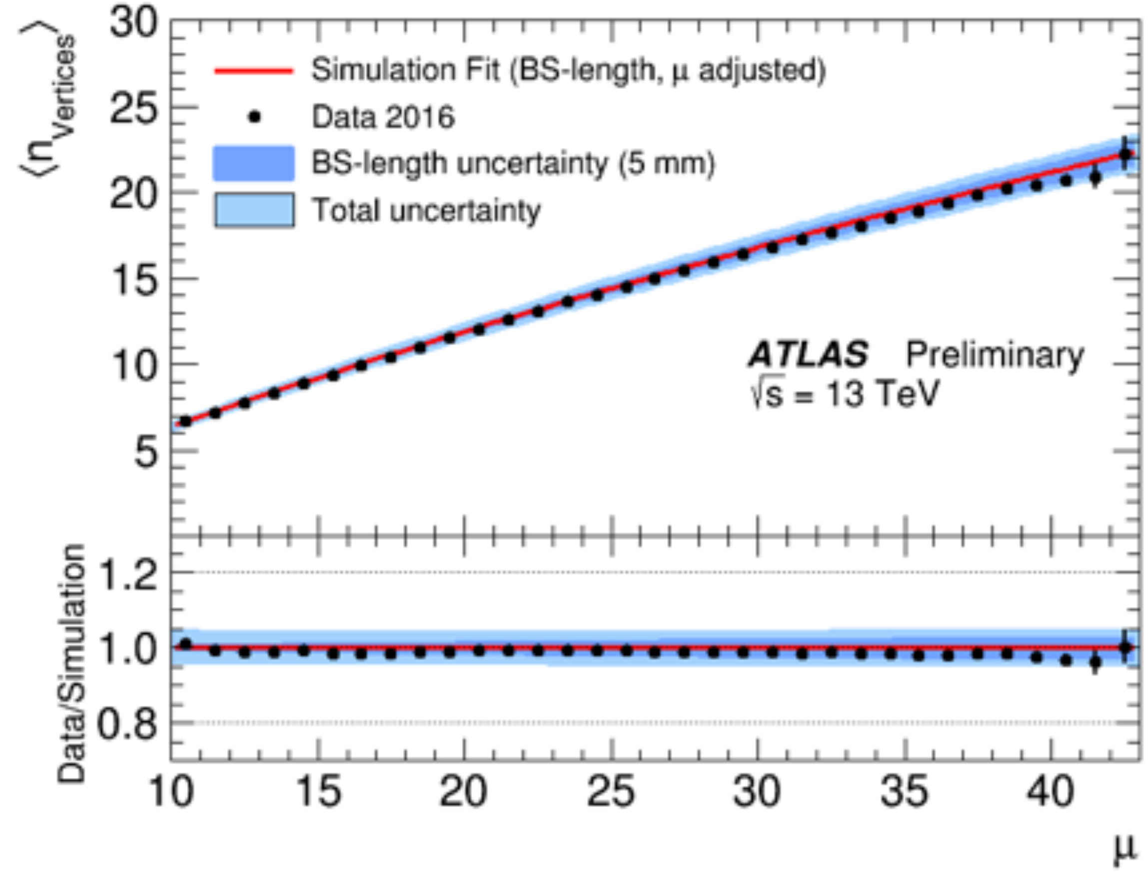
- Graph convolutions operate on graph-structured data, taking into account distance metrics
  - https://tkipf.github.io/graph-convolutional-networks/



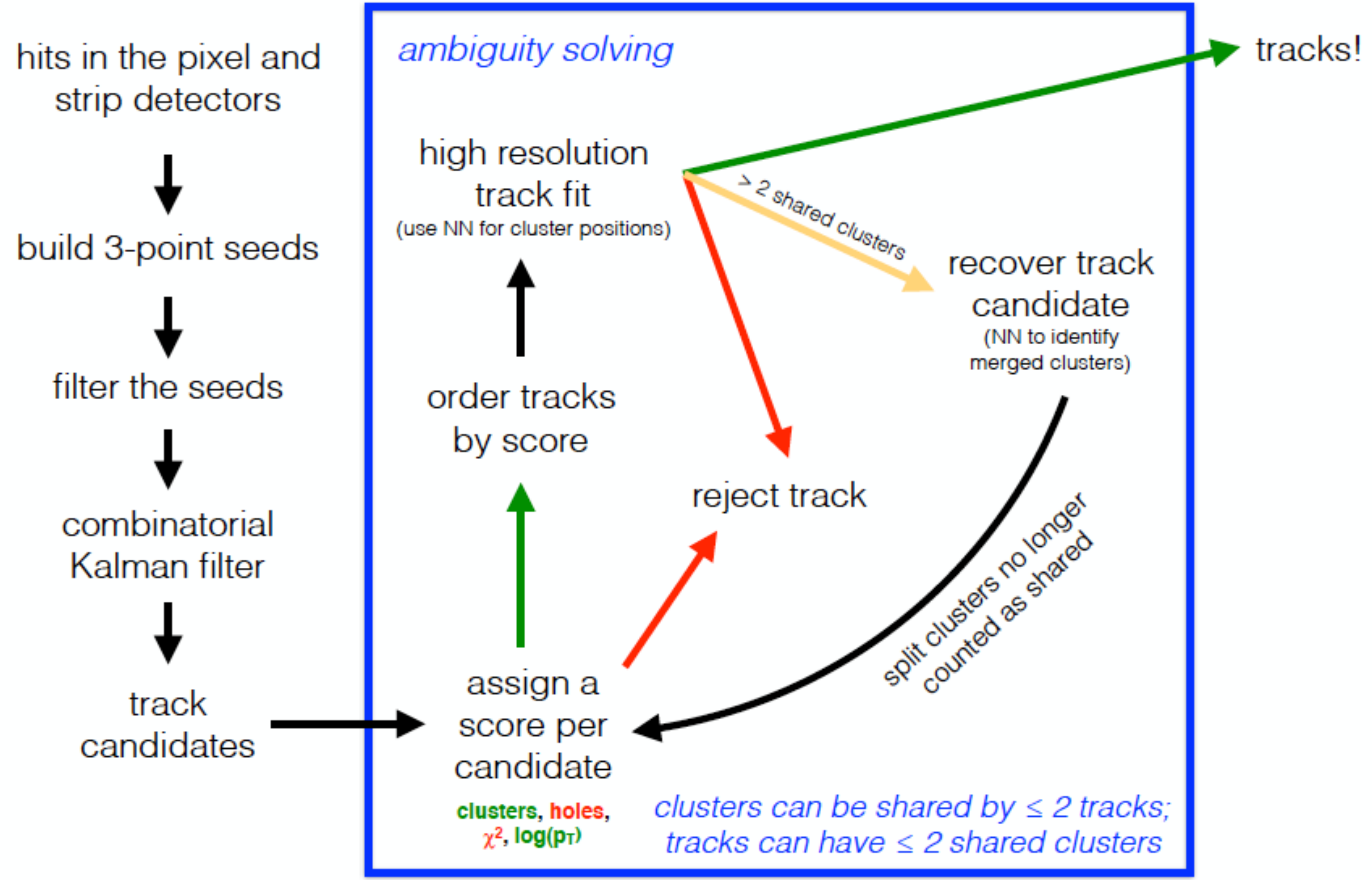
- Connections between ~plausible hits on detector layers can form the graph
  - Handles sparsity naturally
  - Scales naturally with occupancy
- I haven't dedicated much thought to this yet, but it may be versatile enough to do the kinds of things I've already demonstrated

LHC tracking





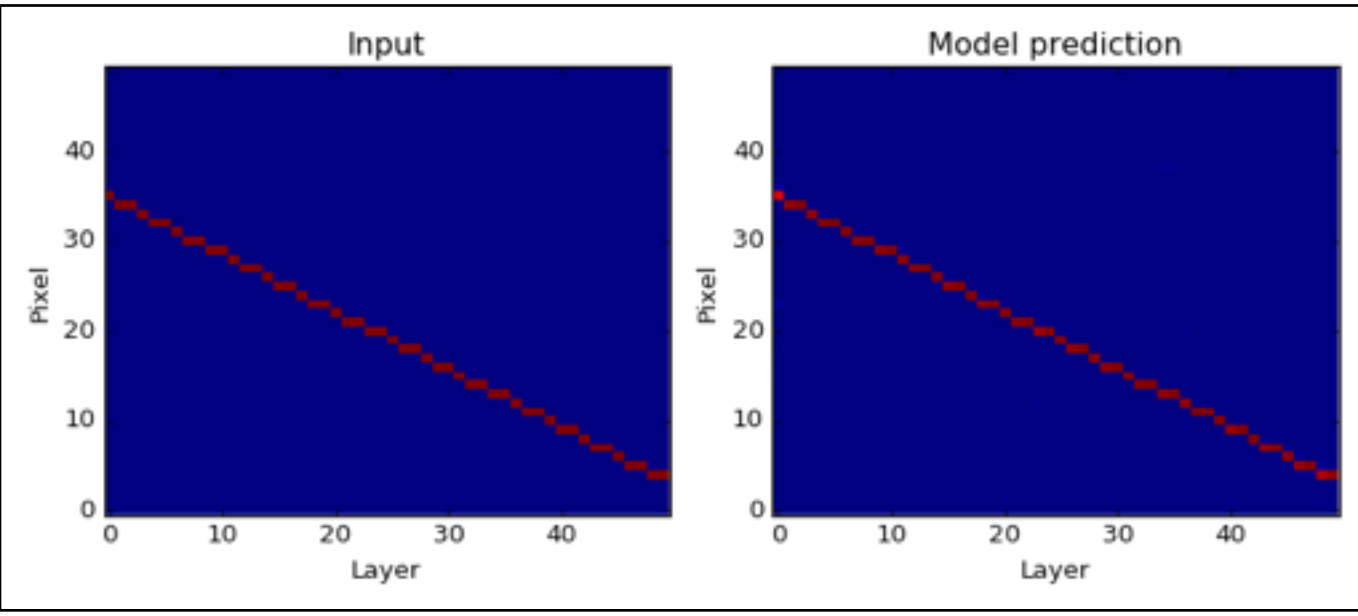
ATLAS tracking in dense environments



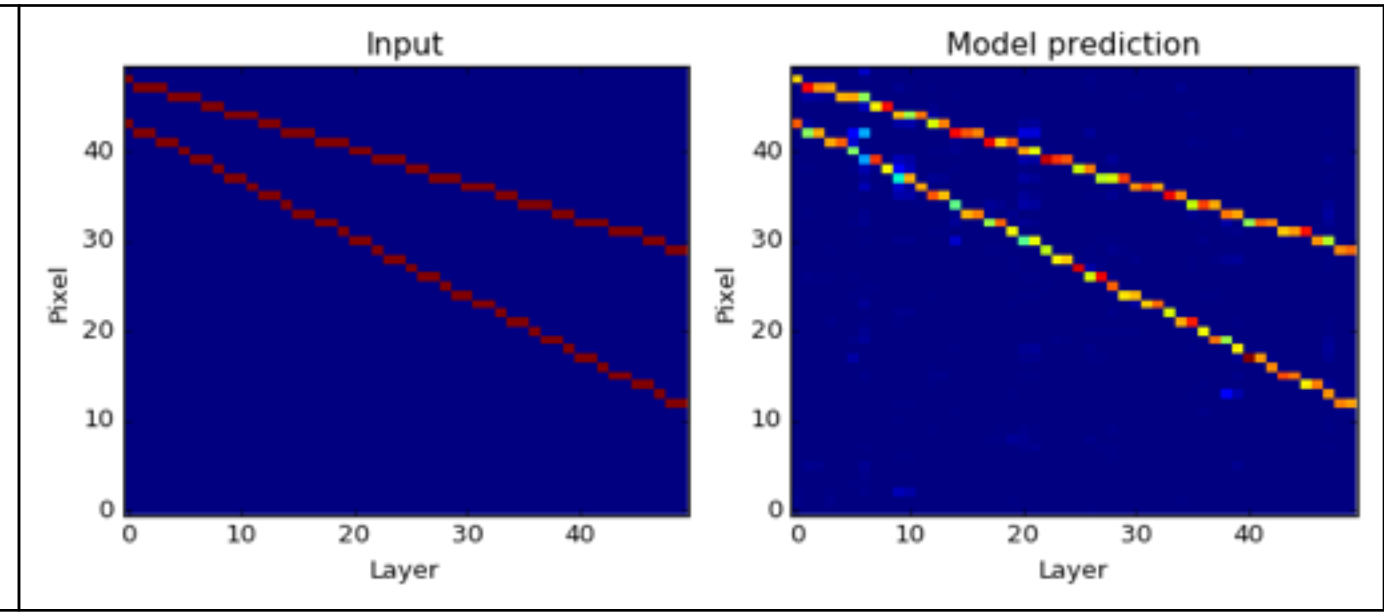
Stolen from Ben Nachman's TPM presentation:
https://indico.physics.lbl.gov/indico/event/433/

LSTMs for track finding (2D toy data)

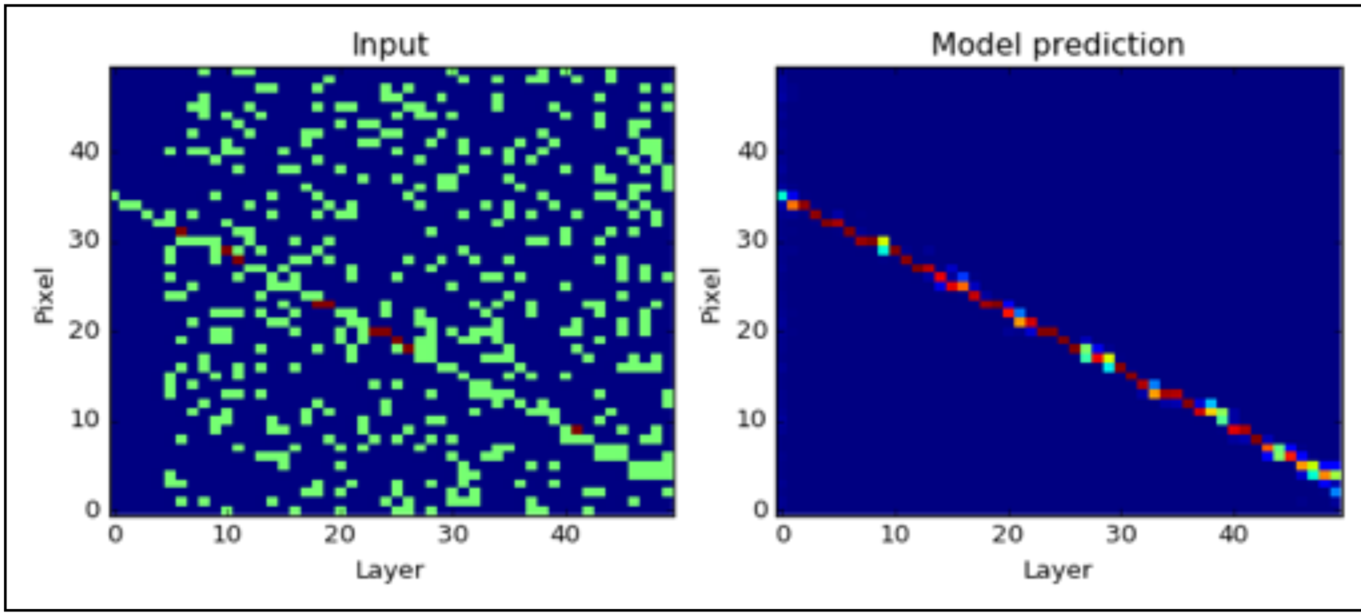
Single track



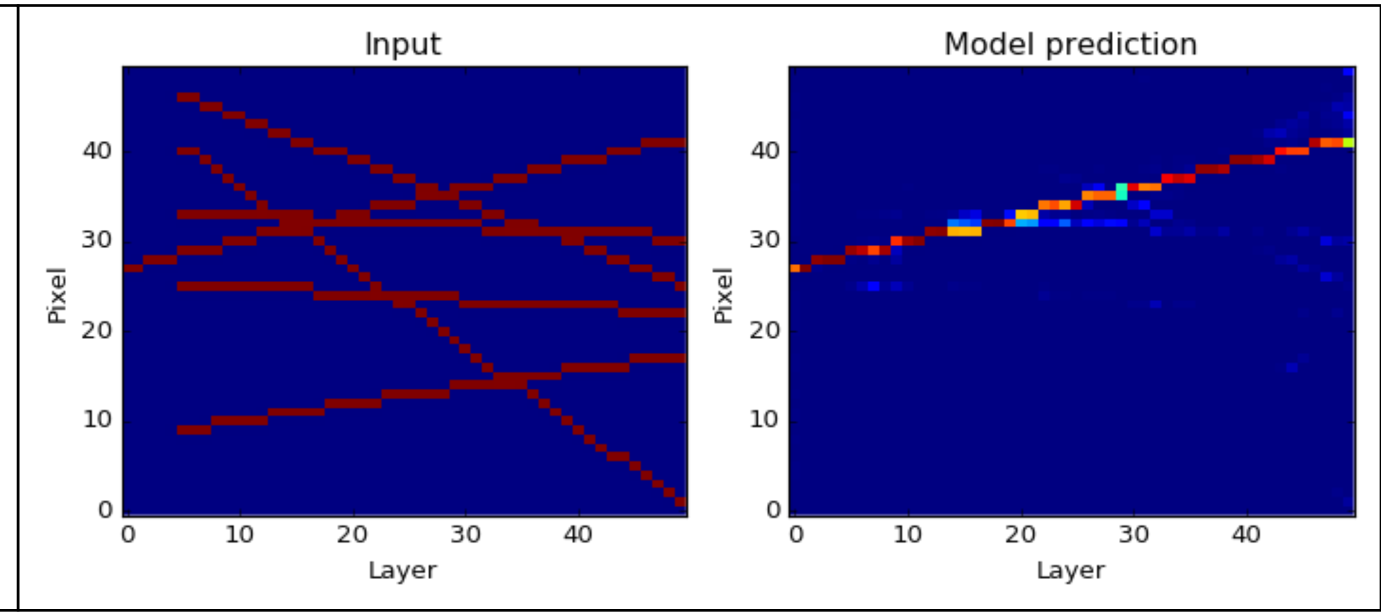
Two tracks



Single track with noise



Single track with background tracks



Model architectures - LSTM

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_1 (InputLayer)             (None, 9, 1024)       0
____________________________________________________________________________________________________
lstm_1 (LSTM)                    (None, 9, 1024)       8392704     input_1[0][0]
____________________________________________________________________________________________________
timedistributed_1 (TimeDistribute(None, 9, 1024)       1049600     lstm_1[0][0]
====================================================================================================
Total params: 9442304
____________________________________________________________________________________________________
```

Model architectures - Deep LSTM

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_1 (InputLayer)             (None, 10, 1024)      0
____________________________________________________________________________________________________
timedistributed_1 (TimeDistribute(None, 10, 1024)      1049600     input_1[0][0]
____________________________________________________________________________________________________
lstm_1 (LSTM)                    (None, 10, 1024)      8392704     timedistributed_1[0][0]
____________________________________________________________________________________________________
timedistributed_2 (TimeDistribute(None, 10, 1024)      1049600     lstm_1[0][0]
____________________________________________________________________________________________________
timedistributed_3 (TimeDistribute(None, 10, 1024)      1049600     timedistributed_2[0][0]
====================================================================================================
Total params: 11541504
____________________________________________________________________________________________________
```

Model architectures - Bidirectional LSTM

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_1 (InputLayer)             (None, 10, 1024)      0
____________________________________________________________________________________________________
bidirectional_1 (Bidirectional)  (None, 10, 2048)      16785408    input_1[0][0]
____________________________________________________________________________________________________
timedistributed_1 (TimeDistribute(None, 10, 1024)      2098176     bidirectional_1[0][0]
____________________________________________________________________________________________________
timedistributed_2 (TimeDistribute(None, 10, 1024)      1049600     timedistributed_1[0][0]
====================================================================================================
Total params: 19933184
____________________________________________________________________________________________________
```

Model architectures - Next-layer LSTM

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_1 (InputLayer)             (None, 9, 1024)       0
____________________________________________________________________________________________________
lstm_1 (LSTM)                    (None, 9, 1024)       8392704     input_1[0][0]
____________________________________________________________________________________________________
timedistributed_1 (TimeDistribute(None, 9, 1024)       1049600     lstm_1[0][0]
====================================================================================================
Total params: 9442304
____________________________________________________________________________________________________
```

Model architectures - ConvNN

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_1 (InputLayer)             (None, 10, 32, 32)    0
____________________________________________________________________________________________________
reshape_1 (Reshape)              (None, 1, 10, 32, 32) 0           input_1[0][0]
____________________________________________________________________________________________________
convolution3d_1 (Convolution3D)  (None, 8, 10, 32, 32) 224         reshape_1[0][0]
____________________________________________________________________________________________________
convolution3d_2 (Convolution3D)  (None, 8, 10, 32, 32) 1736        convolution3d_1[0][0]
____________________________________________________________________________________________________
convolution3d_3 (Convolution3D)  (None, 8, 10, 32, 32) 1736        convolution3d_2[0][0]
____________________________________________________________________________________________________
convolution3d_4 (Convolution3D)  (None, 8, 10, 32, 32) 1736        convolution3d_3[0][0]
____________________________________________________________________________________________________
convolution3d_5 (Convolution3D)  (None, 8, 10, 32, 32) 1736        convolution3d_4[0][0]
____________________________________________________________________________________________________
convolution3d_6 (Convolution3D)  (None, 8, 10, 32, 32) 1736        convolution3d_5[0][0]
____________________________________________________________________________________________________
convolution3d_7 (Convolution3D)  (None, 8, 10, 32, 32) 1736        convolution3d_6[0][0]
____________________________________________________________________________________________________
convolution3d_8 (Convolution3D)  (None, 8, 10, 32, 32) 1736        convolution3d_7[0][0]
____________________________________________________________________________________________________
convolution3d_9 (Convolution3D)  (None, 8, 10, 32, 32) 1736        convolution3d_8[0][0]
____________________________________________________________________________________________________
convolution3d_10 (Convolution3D) (None, 8, 10, 32, 32) 1736        convolution3d_9[0][0]
____________________________________________________________________________________________________
convolution3d_11 (Convolution3D) (None, 1, 10, 32, 32) 217         convolution3d_10[0][0]
____________________________________________________________________________________________________
reshape_2 (Reshape)              (None, 10, 1024)      0           convolution3d_11[0][0]
____________________________________________________________________________________________________
timedistributed_1 (TimeDistribute(None, 10, 1024)      0           reshape_2[0][0]
====================================================================================================
Total params: 16065
____________________________________________________________________________________________________
```

Model architectures - Conv autoencoder

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 10, 32, 32)	0	
reshape_1 (Reshape)	(None, 1, 10, 32, 32)	0	input_1[0][0]
convolution3d_1 (Convolution3D)	(None, 8, 10, 32, 32)	224	reshape_1[0][0]
convolution3d_2 (Convolution3D)	(None, 8, 10, 32, 32)	1736	convolution3d_1[0][0]
maxpooling3d_1 (MaxPooling3D)	(None, 8, 10, 16, 16)	0	convolution3d_2[0][0]
dropout_1 (Dropout)	(None, 8, 10, 16, 16)	0	maxpooling3d_1[0][0]
convolution3d_3 (Convolution3D)	(None, 16, 10, 16, 16)	3472	dropout_1[0][0]
convolution3d_4 (Convolution3D)	(None, 16, 10, 16, 16)	6928	convolution3d_3[0][0]
maxpooling3d_2 (MaxPooling3D)	(None, 16, 10, 8, 8)	0	convolution3d_4[0][0]
dropout_2 (Dropout)	(None, 16, 10, 8, 8)	0	maxpooling3d_2[0][0]
convolution3d_5 (Convolution3D)	(None, 32, 10, 8, 8)	13856	dropout_2[0][0]
maxpooling3d_3 (MaxPooling3D)	(None, 32, 10, 4, 4)	0	convolution3d_5[0][0]
dropout_3 (Dropout)	(None, 32, 10, 4, 4)	0	maxpooling3d_3[0][0]
convolution3d_6 (Convolution3D)	(None, 64, 10, 4, 4)	55360	dropout_3[0][0]
maxpooling3d_4 (MaxPooling3D)	(None, 64, 10, 2, 2)	0	convolution3d_6[0][0]
dropout_4 (Dropout)	(None, 64, 10, 2, 2)	0	maxpooling3d_4[0][0]
convolution3d_7 (Convolution3D)	(None, 96, 10, 2, 2)	73824	dropout_4[0][0]
maxpooling3d_5 (MaxPooling3D)	(None, 96, 10, 1, 1)	0	convolution3d_7[0][0]
dropout_5 (Dropout)	(None, 96, 10, 1, 1)	0	maxpooling3d_5[0][0]
convolution3d_8 (Convolution3D)	(None, 128, 10, 1, 1)	36992	dropout_5[0][0]
permute_1 (Permute)	(None, 10, 128, 1, 1)	0	convolution3d_8[0][0]
reshape_2 (Reshape)	(None, 10, 128)	0	permute_1[0][0]
timedistributed_1 (TimeDistribute	(None, 10, 1024)	132096	reshape_2[0][0]

Total params: 324488