REAL TIME INFERENCE

Don Brittain, GTC 2019

REAL TIME VISUAL PROCESSING

Possible application areas

- Real time upscaling
  - Render at lower res, display at higher res (compute limited cases)
  - Transmit video at lower res, display at higher res (bandwidth limited cases)
- Visual processing
  - Post processing effects: denoising, antialiasing, color-correction
  - Optical flow analysis, video codec support
  - Temporal interpolation or extrapolation (AR/VR)
  - Artistic enhancements (e.g. style transfer, in-painting)
- Other (non-visual) applications

  Pose estimations, facial animation, text-to-speech, voice control, etc.

Typical Deep Learning Practioner View of the World

TRAINING

Try to solve hard problems using clever network design, data filtering and augmentation, advanced ML techniques

→

INFERENCE
Cross fingers and hope for the best performance

Emphasis is on trying to improve the speed and accuracy of *training*!

Typical End-user View of the World

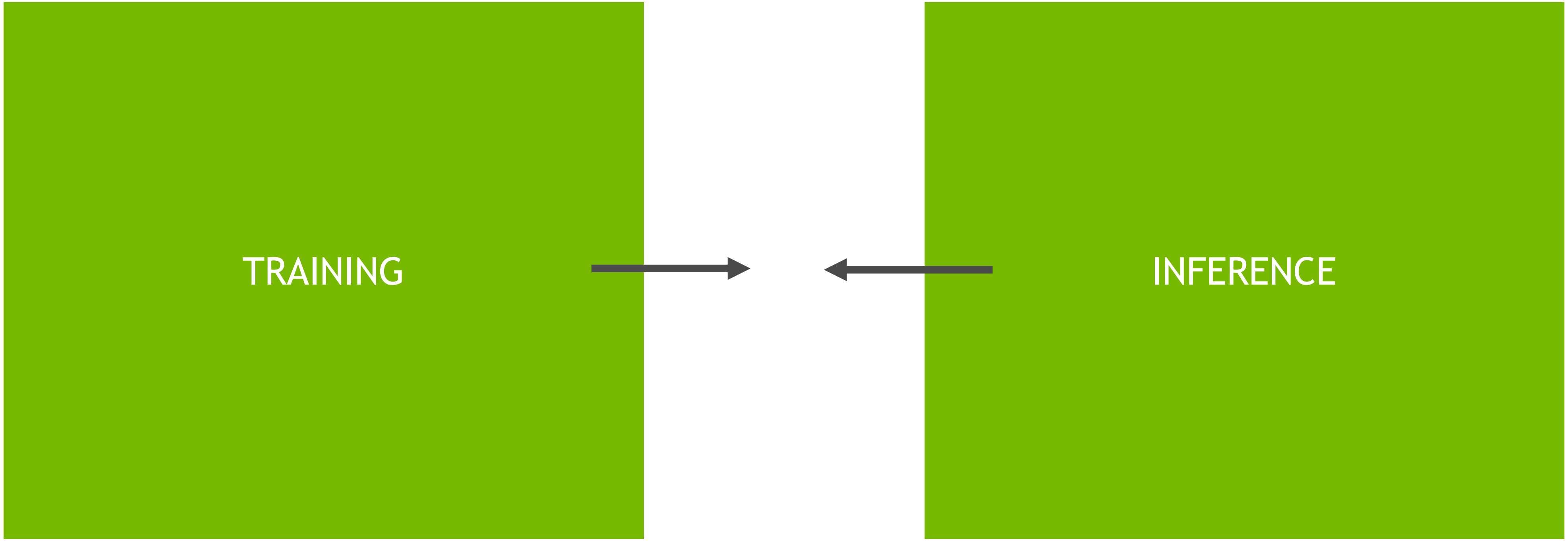
TRAINING → INFERENCE

This is what makes content creation, data analysis, video downloads and games go faster or look nicer!

CULTURE SHOCK!

NVIDIA®

FOR REAL TIME VISUAL APPLICATIONS



Treat training performance (quality) and inference performance (speed) as equal participants in the network design process

Inference speed requirements can be HUGE constraints to network design!

KEY TAKE-AWAY:

Fast inference is also *training* problem

It must be considered during
network design and training!

Check perf early and often, and run lots of experiments

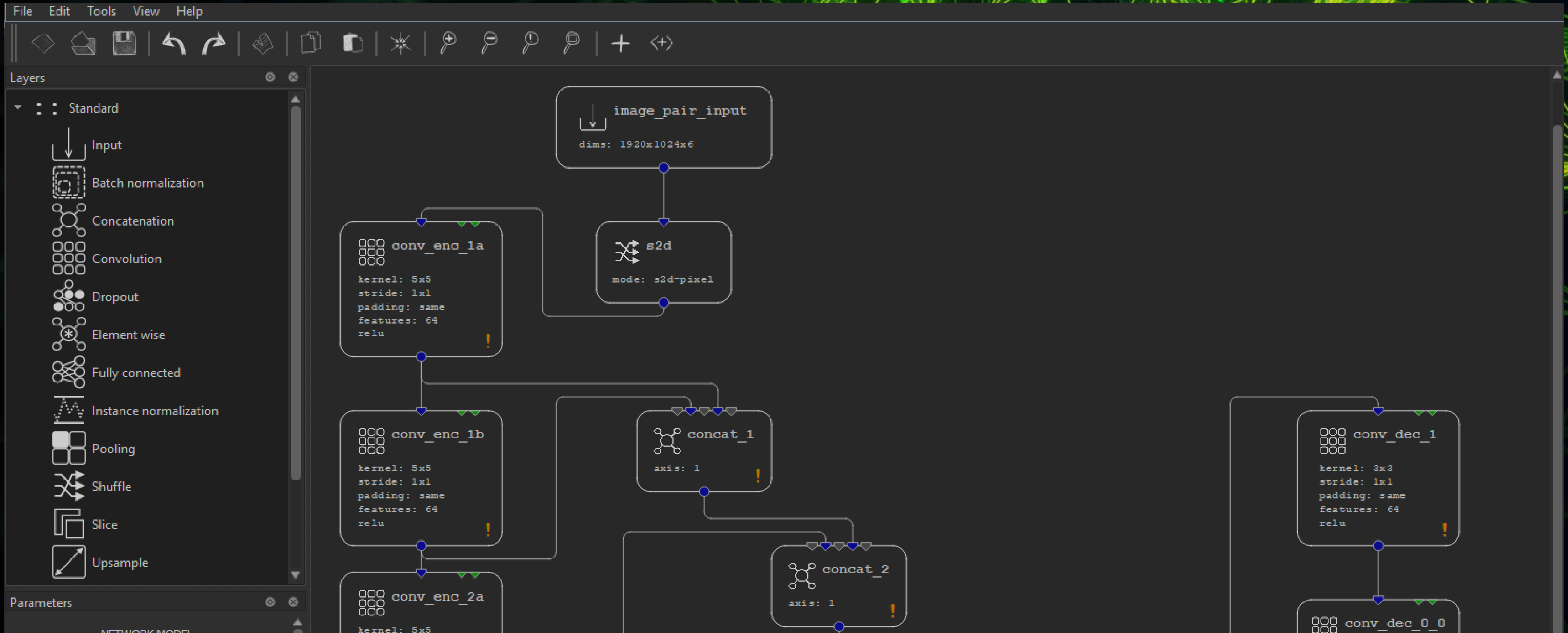
DESIGN, TRAINING, AND IMPLEMENTATION

With Fast Inference as a Goal

- **Choice of model:** For Tensor Cores, stay with multiple-of-8 feature counts in conv layers
  - Start small, add layers or features only when needed to boost quality
  - Concentrate on inference performance rather than training convenience
- **Choice of Loss Function (and training data):** Getting the most out of a small network
  - Common loss like L1, MSE are probably NOT adequate (consider HFENN, content, style, etc)
  - Pay attention to having very clean data, and making sure loss is driving what you want
- **Layer and Computation Graph Optimizations:**
  - Always fuse (or eliminate) operations where possible. Stick with 0-padding, ReLU activation
  - Cache partial results that will be needed again, and resuse memory to keep footprint small

DEMO

MODEL DESIGN: MOTION DETECTION



NCHW AND NHWC

Yes, you do need to know this

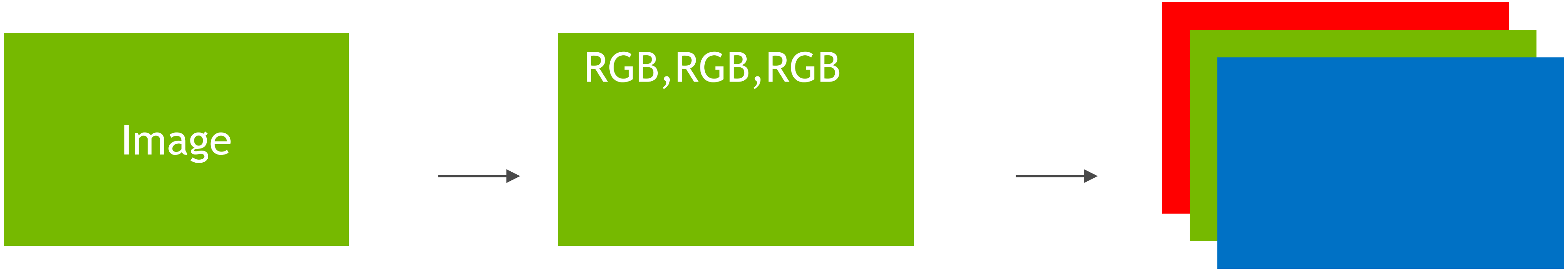
We think “2D” array of pixels

It’s really a “3D” array of RGB values

More like this…

NCHW AND NHWC

Yes, you do need to know this



We think "2D" array of pixels

It's really a "3D" array of RGB values

More like this…

In Memory:

NHWC is the "normal" image storage format *RGBRGBRGBRGBRGBRGB, first row across, followed by each row down*

NCHW is the "normal" tensor storage format *All R's are stored, first row across then down, then all G's, then all B's*

NCHW AND NHWC

Yes, you do need to know this

We think "2D" array of pixels

It's really a "3D" array of RGB values

More like this…

In Memory:

NHWC is the "normal" image storage format

Easy to access neighboring data

NCHW is the "normal" tensor storage format

Easy to process entire "channels"

Tensor Cores "require" NHWC memory layout (using fp16)

fp64 is a 64-bit "double precision" floating point number
fp32 is a 32-bit "single precision" float
fp16 is a 16-bit "half precision" float

THINGS TO CONSIDER

Use untrained inference performance as a guide

- Just as you can train without knowing ultimate inference consequences, you can do "inference" without knowing ultimate trainability

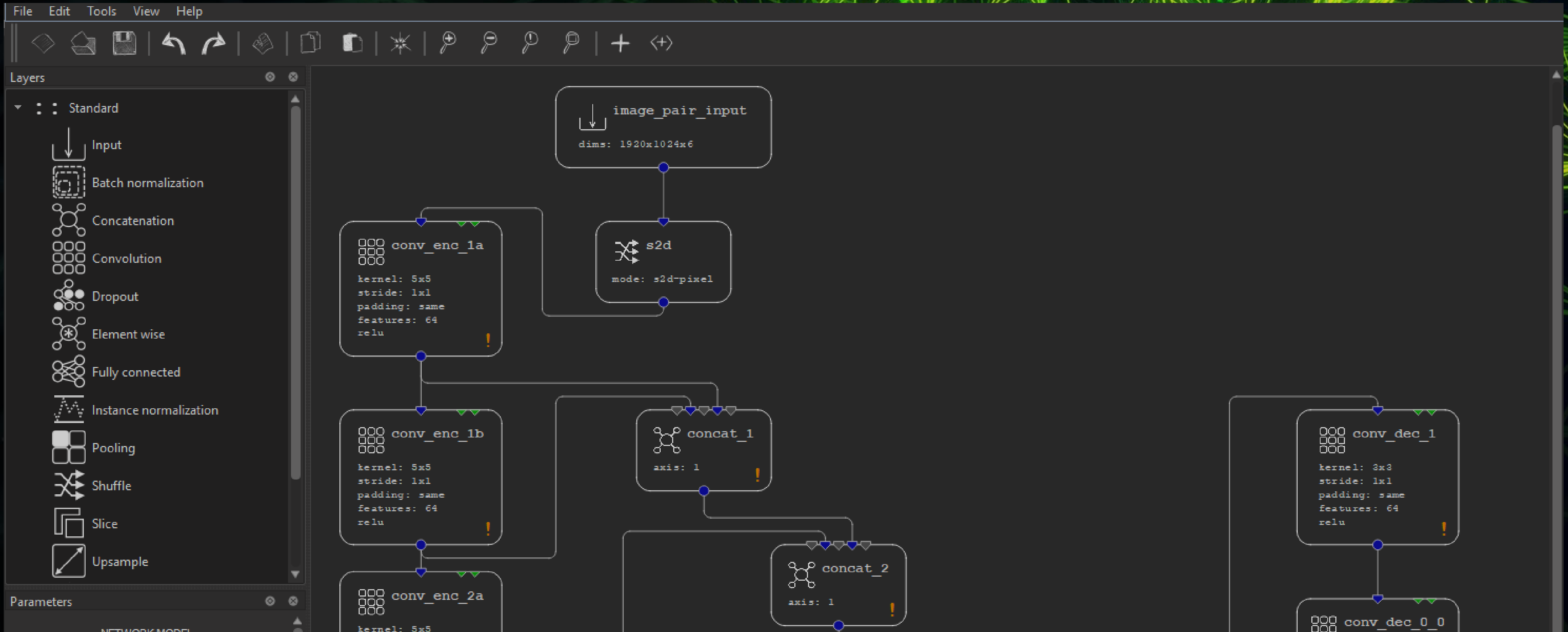
- It's worth looking for fast inference paths (relatively cheap) before investing too much in time-and-compute expensive training. Fusing always helps!

- The fastest performance might not come from the obvious path

- Choice of loss function can dramatically affect how efficiently network capacity is used. Experiment with loss functions to get the best quality per inference batch.

DEMO

I LIED!



File
Edit
Tools
View
Help
Layers
Standard
Input
Batch normalization
Concatenation
Convolution
Dropout
Element wise
Fully connected
Instance normalization
Pooling
Shuffle
Slice
Upsample
Parameters
image_pair_input
dims: 1920x1024x6
s2d
mode: s2d-pixel
conv_enc_1a
kernel: 5x5
stride: 1x1
padding: same
features: 64
relu
conv_enc_1b
kernel: 5x5
stride: 1x1
padding: same
features: 64
relu
concat_1
axis: 1
concat_2
axis: 1
conv_enc_2a
kernel: 5x5
conv_dec_1
kernel: 3x3
stride: 1x1
padding: same
features: 64
relu
conv_dec_0_0

THINGS TO CONSIDER

KISS (Keep It Simple Stupid)

- Eliminate “training training wheels” if possible
  - Normalization layers (Instance norm, batch norm and similar) are probably not needed for small, real time networks
  - Leaky ReLU or ELU can probably be replaced with just use ReLU
- Work from simplified network “up”, rather than complex network “down”
- Work in “self-normalized” space, centered about 0 (i.e. whiten data explicitly)
  - E.g. transform image 0-255 values to -0.5 to 0.5 space
  - Only use zero padding on conv layers if possible

THINGS TO CONSIDER

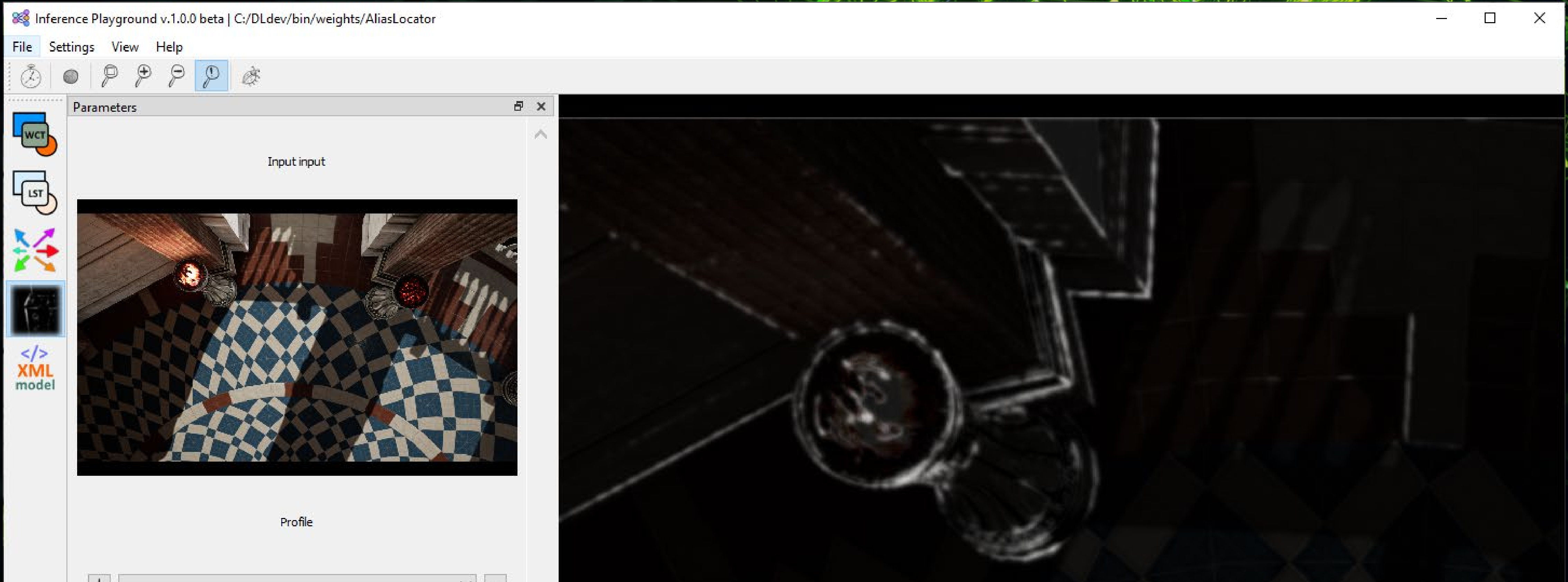
Don’t try to learn what you already know

- This slows both training and inference
- Can lead to temporal instability
- Just because “you can” doesn’t mean “you should”
- Example: use “residual learning” to avoid problems from too much MUSH

- Note: *MUSH* means “Making Up Shtuff” – yeah, we’ll go with that – and rarely does a network create temporally stable data during image (re)construction without being highly encouraged in that direction

DEMO

ALIASING LOCATOR



THINGS TO CONSIDER

Consider both compute and memory bandwidth costs

- Real time image processing touches a TON of data, and there are many cases where just accessing the data (multiple times) constrains wall-clock throughput
- Examples:
  - For an autoencoder, consider replacing convolution/pool layer pairs with strided (2x2) convolutions, even if you need to add features
  - Consider places where space-to-depth operations can help.
  - Test feature counts for “sweet spots” in the hardware pipeline (akin to finding freeways rather than staying on surface streets). Tensor cores virtually always require feature counts that are multiples of 8.
  - Explicitly “fuse” multiple layers of processing together whenever possible (or restrict you model to layers where pre-fused implementations are available (e.g. 0-pad, ReLU with conv layers)

DEMO

REAL TIME STYLE TRANSFER

THINGS TO CONSIDER

Advanced Possibilities

- Cache “precomputable” or intermediate results if they will be used more than once
- Choice of qualitative network model can make dramatic differences in perf (and quality)
- Use data reduction if quality is still OK (e.g. 16-bit YUV instead of 24-bit RGB)
- Use lower-precision data types if possible
    - Fp16 instead of fp32 (depending on hardware support), Int8 instead of floats if quality allows
- Specifically design around “run time” inference hardware (e.g. consider memory bandwidth / computation performance ratios, and whether tensor cores are available)
- Choose a hybrid classic-DL blend if this works for your application

Thank You!