Lean Scenes: Variable-Fidelity Models Reduce Machine-Learning Training Requirements

Blake J. Anderton
Torch Technologies, Inc.
Huntsville, Alabama
Blake.Anderton@TorchTechnologies.com

ABSTRACT

This paper describes an approach to avoid waste in machine learning.

Image-processing neural networks characterize objects from features learned in training data. In some applications, the amount of training data is constrained by expensive data collection or limited number of physical images. In such contexts, it may be preferable to first train performance against simulated imagery, then fine-tune by training with physical images. This "transfer learning" procedure helps networks meet performance requirements with reduced demands for amount of physical data consumed.

Simulation-based transfer learning raises two key concerns. First, modeling highly realistic scenery through high-fidelity simulations typically involves significant computational expense. This may constrain the amount of generated imagery available for training, leading to reduced performance. Second, modeling artefacts may result in the network learning to recognize and respond to artificial features having no real-world equivalent.

This paper examines several approaches undertaken by Torch Technologies, Inc. which address these concerns within the application of an image-segmentation neural-network for calibrating camera pose. First, transfer learning is applied to a series of simulation results, beginning with a pool of many moderate-fidelity runs, then fine-tune training on fewer, higher-fidelity cases. Second, the influence of artificial features may be (A) mitigated through blending physical/synthetic imagery through object texturing or (B) monitored through saliency-map diagnostics which inform analysts of image regions most-responsible for network performance. Artefact robustness methods represents an active research area.

These approaches demonstrate that (A) simulation output need not be exclusively highest fidelity to be of utility to early-phase training, and (B) overall computational expense can reduce through training sequences which increase modeling realism while also reducing number of generated samples. In these ways, variable-fidelity simulations dynamically provide the modeled realism appropriate to a machine-learning algorithm's evolving capabilities.

ABOUT THE AUTHOR

Blake J. Anderton has worked 15 years within the DoD industry with remote-sensing modeling/simulation. Dr. Anderton presently works for Torch Technologies researching machine-learning applications in passive sensor technology, swarm UAS optimization, and cybersecurity. In 12 years at Raytheon, he served as principal investigator for several modeling/simulation efforts including operations research for optimized-deployment and multi-sensor fusion. He earned a Ph.D. in Optical Science from University of Arizona.

Lean Scenes: Variable-Fidelity Models Reduce Machine-Learning Training Requirements

Blake J. Anderton
Torch Technologies, Inc.
Huntsville, Alabama
Blake.Anderton@TorchTechnologies.com

INTRODUCTION

Forms of Waste when Training Machine-Learning Models

Machine-learning (ML) training can be "wasteful" in multiple ways. Each data batch consumed without significantly affecting model proficiency could be considered a misappropriation of time & computational resources. Similarly, field-acquiring an exorbitantly large pool of physical data for training can also be considered wasteful when alternative training methods exists. Using sim-generated output as training data offers promise but – when the ML model's proficiency has not yet been assessed – such could introduce dangers such as over-/under-specifying the fidelity/volume of generated output. When such results in impractically-high computational demands (high-fidelity) or performance susceptibility to simulation artefacts (low-fidelity), training becomes impractical or performance-capped, incurring yet another waste form.

This paper presents an initial study performed by Torch Technologies seeking to explore these forms of waste and offer suggestions on how such can be avoided through alternative simulation modeling procedures, a staged (across disparate data pools) training sequence, and procedures which iteratively (1) assesses sim-trained model performance on physical imagery and (2) feeds back the observed performance deficiencies into reconfiguring sim-data generation before repeating the sim-data training.

The Computational and Data Constraints of Training Machine-Learning Models

Machine-learning methods produce predictive capabilities through processing data in order to capture correlative relationships among input variables and a desired outcome. This processing of input data for the purpose of updating the model is referred to as "training" the model, and this training procedure is where the "learning" may be considered to occur. The performance of an ML model is primarily determined by the choice of model architecture, the quality & quantity of available training data, and the choices in how the execution of the training procedure.

The training procedure can be demanding in terms of the computational expense required as well as the amount of data which must be consumed during training. One factor influencing the computational expense is the depth of the selected network architecture (i.e., how many layers compose the model). Deeper architectures will require additional computational expense when forward-propagating data from the input layer, through all intermediate layers, to the final output layer. The volume of data also presents computational demands, especially when training procedures must progress through the entire training data set (or "epoch") in multiple passes. Thus, computational demands may present a constraint to training.

In addition to computational constraints, it is frequently the case that the amount of training data is limited. This limit is often due to the expenses involved in procuring a suitably large training data set. For image-processing applications, high-fidelity training data involves physical imagery obtained under a broad span of conditions (lighting, perspective, camera settings, objects present, etc.) which translate into data acquisition expenses. Some applications, such as image-segmentation, may also require time-intensive post-processing, such as truth-labeling each pixel according to the type of object contained within that pixel. Thus, limited availability of high-fidelity data may also present practical limitations to training.

The choice of how to conduct training is influenced by restrictions on computational resources and amount of training data. Recent research has pushed back against these restrictions in the form of recommended practices to lessen the

amount of data consumed to train to a specific level of proficiency. These include methods which specify training hyperparameters (notably, the learning rate) in a way to expedite the model's proficiency. Such methods include discriminative (or "layer-specific") learning rates (Howard J., and Ruder S.: 2018), learning-rate annealing (Bengio Y., 2012), and alternative learning-rate schedules such as cyclical learning rates (Smith L.N., 2017). The accelerated learning obtained through these methods reduces the computational expense (relative to training at fixed learning rates). In addition to establishing training hyperparameters, the use of training data can be expanded through data augmentation methods, such as flipping/rotating/cropping existing imagery.

In cases where available pool of data is extremely limited, these methods to set the learning-rate and augment the data may still be insufficient. This paper explores an application which produces an effect similar to training data augmentation in the form of utilizing simulation data within a transfer-learning sequence

Transfer Learning with Simulation Data

In many ML applications, it is frequently the case that similarity exists between application contexts. One application may classify images to classes of {dog, cat, horse, …} while another to classes of {plane, tank, launcher, …}. When using neural network (NN) models, the lower-level layers (nearest to the input) capture primitive features such as those associated with 3-5-pixel edge-detection. Such features are often common across multiple applications; only the topmost layers (involving composite features and their synthesis into classification labels) are strongly coupled to a particular application. This commonality of lower layer features across application context presents an opportunity to use other context's training data in a "transfer learning" training process:

(1) First, the model is trained upon the data from that similar context. In such a procedure, the upper layers' parameters may be "frozen" during training, so that they are not updated and only the lower-layer parameters are modified.
(2) With model parameters now at the values obtained from training in the similar context, training continues with the data pool from the desired context, with parameters in all layers updated.

This process can be distributed to multiple stages if additional data pools are available.

Transfer learning is particularly applicable when limited data is available in the relevant context. In such cases, there is frequently a large pool of data available from another, less-relevant context. The initial training on the *large, less-relevant* pool serves to effectively initialize the model's parameters for subsequent training in *small, more-relevant* pools. In the literature, this latter training stage is referred to as "fine-tuning" the model.

When a credible simulation capability exists, the transfer learning technique raises the prospect of first training upon sim output, then fine-tune training upon physical data. In this regard, the context-relevance of the sim output is governed by sim fidelity. Frequently, the computational demands of the sim are proportional to fidelity (e.g., for imagery rendering methods, lower-fidelity ray-*casting* vs. higher-fidelity ray-*tracing*). Transfer learning upon sim output may thus take place across multiple stages, beginning with a large volume of low/moderate-fidelity output, then progressing towards smaller volumes of higher-fidelity output. In this procedure, transfer learning provides an alternative to the computationally-demanding generation of a large pool of high-fidelity sim output.

Simulation Artefacts: Challenges and Mitigations

Due to the varying levels of sim fidelity, it is possible that artefacts may be introduced into sim output. Examples of fidelity-based artefacts in rendered imagery may fall into multiple areas of consideration, such as lighting (e.g., omission of ambient occlusion shadows), solid geometry (e.g., presence of meshing triangles in complex surfaces), materials/surfaces (e.g., inaccurate reflectance models), camera settings (e.g., omitting blurring outside the hyperfocal range), and by-products of the rendering procedure (e.g., grainy texture within shadowed regions, resulting from limited number of traced rays). These examples illustrate the span of areas among which a balanced choice of model fidelity must be made: removing all artefact sources in all areas is done at premium computational expense (and often impractical).

Given the benefits of sim output in transfer learning, it is important to also recognize the risks. Sim artefacts represent a nonphysical source of imagery structure which an ML model could train to associate with certain classes. Without subsequent fine-tune training within improved realism contexts, a significant portion of such models' performance

will rely upon on the presence of artefacts, limiting model applicability in the real-world domain. In some cases, alternate modeling schemes may reduce artefact significance without requiring large computation (e.g., the *inexpensive* ray trace rendering of a tree by imposing a physical image of a tree as a surface texture, versus *expensive* rendering using a CAD tree with high polygon count due to every tree/branch involved).

Role of Human Intelligence and Model Interpretability

An additional means of mitigating a sim-trained ML model's susceptibility to artefacts is to involve human analysis of model performance. Human-in-the-loop involvement already plays a major role in guiding the training process (e.g., monitor training/validation loss for overfit, specify the number of training epochs, etc.). Expanding this role to include performance quality assessment (through human-detection of performance ties to artefacts) results in performance improvements achieved by human intelligence and machine learning.

In order for human input to detect a coupling between performance and sim-artefacts, it is necessary for the model's performance to be *interpretable*. This can be challenging, especially for deep NN architectures. Recent research in NN model "interpretability" (also "explainability") has produced methods providing insight into how an output (inferred image class) is linked to specific elements in the input content (i.e., which pixels influenced the class inferred). For imagery, this takes the form of localizing the input image regions which have the most-significant influence upon the classification outcome. The results of such methods are typically presented in the form of a heatmap overlaying the original image, highlighting the region's most responsible for the inferred classification. The methods of this approach include class activation maps (Zhou et al, 2016) and various implementations of saliency maps (e.g., Smilkov et al, 2017). The net result of interpretability is a human-understandable result which enables human observation of the input features used by a model to infer its quoted outcome. As an illustration of the value of this approach, this human-in-the-loop with interpretability approach would catch whether object classification was incorrectly linked to the environment (e.g., an object class present only in bright-sky backgrounds) rather than the structure present within the object region alone.

Scope of Present Investigation

This paper explores methods to circumvent typical ML computational/data-availability restraints by implementing an illustrative image segmentation context using sim output in transfer-learning. The selected context exemplifies both limited samples of field imagery (about 10) and a simulation procedure with evident artefacts. The sim used will be seen to offer alternatives to otherwise manual efforts to produce truth data (such as class-labeling individual pixels) and to the choice in rendering complex structures (such as a forest/field scene's many leaves/branches/grass-blades). This application illustrates how simulation capabilities may be appropriately utilized for an expedient ML training procedure producing proficient models resilient to sim artefacts. Additional description of this intended application immediately follows.

APPLICATION: ALIGNING MULTI-PERSPECTIVE CAMERAS VIEWING COMMON OBJECTS

Overview: Geometric Calibration of Multiple Sensor Systems

Multiple camera systems are used in many areas, spanning industries as diverse as film-production to defense. One example defense application involves the simultaneous, high framerate observation of a munition detonation from multiple perspectives. The diversity of perspective allows three-dimensional characterization of potentially lethal phenomena arising from the blast event, such as tracking the flight trajectories of blast-fragments and tomographically reconstructing the blast-wave's volumetric density and pressure.

Such systems require quantitatively-accurate determination of each camera's geometry (location and line-of-sight) in order to construct reliable lethality assessments. One means of determining line-of-sight is through introducing scene objects which fall within the field-of-view of multiple cameras. The placement and assessment of these commonly-viewed "fiducial" objects comprises a key part of field-calibration of system geometry. Figure 1 illustrates a representative test configuration – highlighting overlapping fields-of-view and fiducial placement – for the multi-sensor Optical Warhead Lethality Sensor Suite (OWLSS) produced by Torch Technologies.

Existing methods for establishing accurate geometric calibration involve long-duration procedures which involve multiple iterations of manual steps by field technicians. The duration associated with these procedures can detract from using test-range availability windows efficiently and may (in cases sensitive to local weather) limit the amount of data collected in available schedule windows. Replacing the existing iterative/manual geometric calibration procedure with an autonomous means of localizing (at pixel-level accuracy) fiducials in imagery would improve the resilience, cost-effectiveness, and data-collecting capacity of such systems.

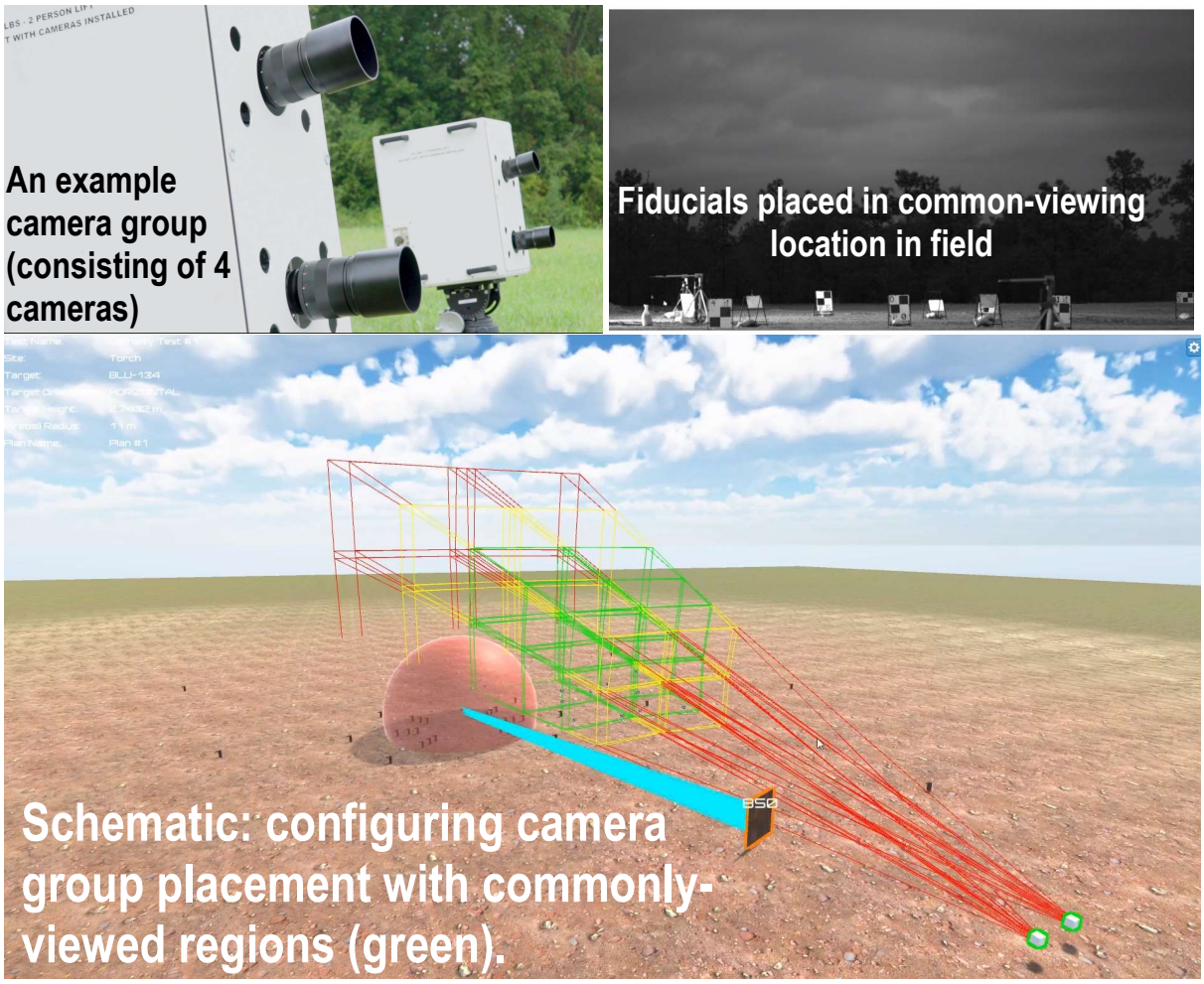

Figure 1. Representative Layout of the OWLSS Multi-Camera System

Semantic Segmentation as Enabler for Autonomous Geometric Calibration

This autonomous calibration capability can be implemented through using an ML model in the form of a NN which segments imagery according to the type of object detected. *Semantic* segmentation (segmenting into regions consisting of a detected object type) is frequently-used when conducting supervised-training of NNs. Compared to alternate segmentation schemes (e.g., segmentation via super-pixel clusters), semantic segmentation's linking of regions to object type is more useful for localizing objects according to type "fiducial". Figure 2 illustrates semantic segmentation from the perspective of truth data: a rendered image (left) and a truth segmentation by object type (right) are both produced by the simulation. This example illustrates segmentation according to four classes ("fiducial", "terrain", "background", "sky"); the application will involve only two classes ("fiducial", "not a fiducial"). This example produces segmented imagery from truth (i.e., through simulation capabilities); the next section describes the NN constructs which infer segmentation estimates.

Figure 2. Illustration of Rendered Imagery and Corresponding Segmentation Mask

SPECIFYING A SEGMENTATION NETWORK ARCHITECTURE

U-Net Architecture for Semantic Segmentation

Semantic segmentation image-processing models comprise two primary functions: object classification, and object localization. These two roles may be respectively enacted via two sub-networks operating in sequence: an *encoder* network (mapping an image or image region into a classification outcome) and a *decoder* network (mapping a classification to the image pixels most responsible for it). The U-Net architecture (Ronneberger O. et al, 2015) is a popular choice for semantic segmentation for Convolutional Neural Networks (CNN) encoders. U-Nets use the layer-specific feature activations (for each convolutional layer in the encoder) to subsequently map the inferred classification to the image plane (in each convolutional layer of the decoder). This scheme provides the convenience of not requiring separate specification and training of encoder and decoder networks – it effectively models the decoder as the encoder's convolutional layers executed in reverse, with each feature-layer "step down" (e.g., from the rightmost 196×196-pixel layer's up-convolution to the 392×392-pixel layer) performed using the activations from the corresponding encoder layer. U-Net training was performed by extending a classifier-network's (cross-entropy) scoring metric to each mask pixel and combining the matrix of metrics into a single scalar for back-propagating parameter updates. Figure 3 schematically outlines the U-Net architecture and processing sequence for the case of a ResNet34 encoder (described next).

CNN Architecture for Classification Encoder

The encoder network is functionally responsible for image classification. CNNs are presently a frequently-used class of image-processing architectures. Residual Networks (ResNets) – a CNN sub-type –use layer-skipping "residual" blocks to achieve effective learning in many-layer (or "deep") architectures by preventing the diminishing of parameter-updates computed at progressively-deeper layers (He K. et al, 2016). For this study, a ResNet34 architecture was selected, offering the advantages of deep-networks' rich inferential capabilities while requiring only moderate storage (i.e., network parameters fit on available RAM).

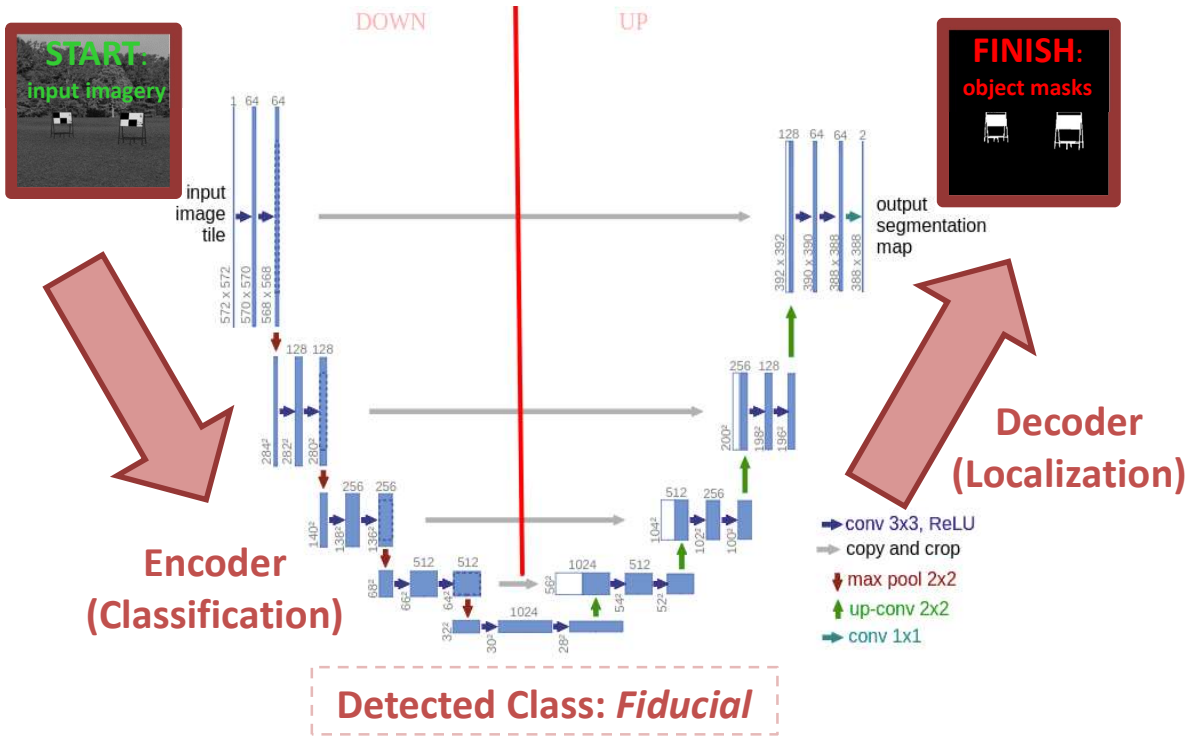

Figure 3. U-Net Architecture (with ResNet34 Encoder) and Operational Workflow

SCENE GENERATION: METHODS, TOOLS, AND OVERKILL AVOIDANCE

Image-rendering scene-generation may be done through a variety of computational schemes whose relevance depends upon the context's physics, the spatial scale of the structures involved, and the applicable electromagnetic spectral regions. For rendering *visual* scenes, ray-casting and ray-tracing represent two general approaches. *Ray-casting*, a low/moderate-fidelity rendering, linearly-propagates rays which transition from light "sources", reflecting/transmitting surfaces, and image-plane "sinks". *Ray-tracing,* a high-fidelity rendering, propagates rays according to more-comprehensive physical models of the passage medium, the surface models, and the light source. Ray-casting may be more appropriate when physical detail is a lower priority than maximum frame-rate. Ray-tracing applies when detailed shadow-models (from multi-surface reflection computation), passage through complex volumetric media (e.g., the *scattering* of fog/smoke or the *refractive-inhomogeneity* of atmospheric turbulence) warrants increased computation. These rendering methods may be considered two options for specifying fidelity – ray-casting for large-volume/low-fidelity data pools, ray-tracing for small-volume/high-fidelity.

This study selected Blender – a free/open-source 3D computer graphics software toolset – as the source of visual scene-generation. Blender natively contains both ray-cast (Blender Render) and ray-trace (Cycles Render) engines, enabling CAD model re-use when transitioning between ray-casting/ray-tracing. Additional relevant Blender capabilities include:

- *scripting backend (Python)* which allows generation of a Monte Carlo image set (varying fiducial number/placement, sky model/brightness, terrain model/structure, background texture/placement, and camera placement/perspective);
- *scene lighting*, which determines how sky-based illumination is done (e.g., through high dynamic range imagery – HDRI – models, or mapping equirectangular JPG image to the enclosing hemisphere as an emissive surface);
- *truth segmentation*, performed by modeling object surfaces as emitters colored according to object type (see right image of *Figure 2* as example); and
- *selective activation of high-fidelity features*, e.g., ambient-occlusion shadows and detailed surface models (maps specifiable for diffuse/specular reflectance, bump, gloss, normal, etc.).

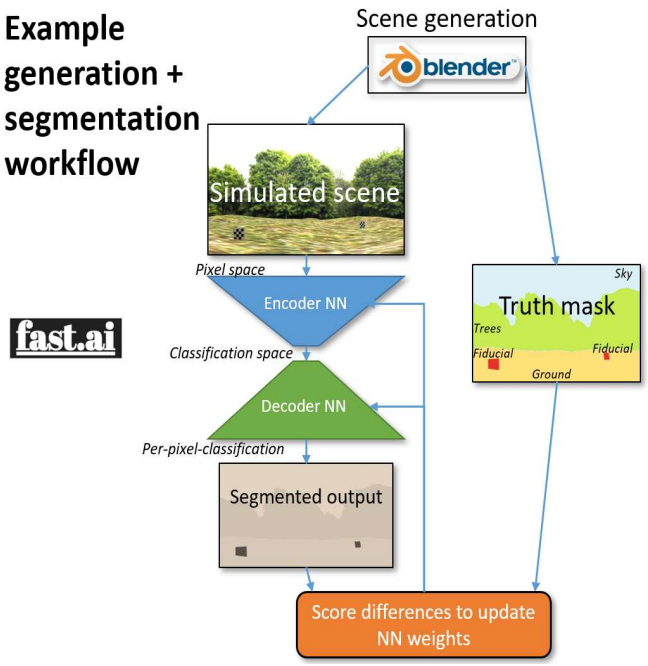

Figure 4. Use of Sim-Generated Scenes & Masks in Model Training

Simulating a single scene instance results in two outputs – a rendered scene and a truth mask (colored by object type). Figure 4 illustrates how these outputs are processed when training the segmentation model – the model converts the scene image into a mask estimate which is compared to the truth mask for updating the model.

Modeled scenes had three types of objects with high amounts of structural detail: background (trees), terrain (grass/gravel), and sky (clouds). As discussed earlier, polygonal meshing of every branch, leaf, stone, and grass shoot is likely to be impractically excessive. As an alternative, these three objects were modeled as *textures* (namely, a physical public-domain image overlaid on a primitive structure such as – for the background – a rectangular sheet facing the camera). Texture parameters (scale, location, brightness, selected sample) were set to Monte Carlo variables. Approximately 10 instances existed for each (10 skies, 10 tree backgrounds, and 10 terrain models). The terrain's primitive was structured to model rippling (small hills) by passing a randomized 2D array through multiple 2D Gaussian convolution kernels (at various kernel widths) and summing each into a crumpled square scaled to cover camera field of view. The sky primitive comprised the environment's 2D hemisphere.

Modeling artefacts observed from the rendered imagery include absence of fiducial shadows, an artificially abrupt terrain-to- background transition, background scaling at values non-realistically large and/or incompatible with ground structure, low-resolution terrain-texture artefacts, absence of hyperfocal object blurring, and infrequent cases of fiducial geometry intersecting other fiducials. Perhaps the most egregious artefact was that fiducial textures had a different texture than that used in Blender (Blender scenes had "backslash" black-square pattern compared to typically "forward-slash" fiducials in field imagery). In practice, a prudent course of action would be to address each detected artefact possessing a feasible mitigation approach. For the purposes of this study, the image database with these artefacts was used in network training, to examine the extent to which sim-artefacts inhibit performance when models perform on real-world imagery.

IMPLEMENTING AN APPROPRIATE TRANSFER-LEARNING CONFIGURATION

This study's scope involves implementing a representative transfer-learning procedure – spanning multiple (physical/simulated) data pools – to train a segmentation network. Due to time-constraints, the final step – fine-tune training upon physical field imagery – was omitted for now, owing to the time-intensive/subjective procedure of manually specifying fiducial pixels. Therefore, the transfer-learning will be demonstrated by training across two data pools, then testing performance on fiducial field imagery, examining for the presence of modeling-artefact susceptibility.

In this application, two training objectives are required – the ResNet34 encoder learns classification, and the U-Net segmenter learns localization. These were accomplished by a two-step transfer-learning procedure:

(1) The ResNet34's parameters were initialized to values obtained from training on ImageNet – an image database for visual object recognition comprising ~14 million images classified into ~20,000 categories (Deng J. et al, 2009).
(2) The U-Net parameters were updated through training on 1000 512x512 Blender-simulated fiducial scenes, with each output comprising two images – scene and truth mask.

This transfer-learning procedure relates two databases (high-volume/low-relevance ImageNet and low-volume/high-relevance Blender scenes) *and* two capabilities (classification and localization), illustrating how a transfer-learning scheme can be flexibly tailored to meet the demands of specific applications.

Figure 6 in "Conclusions" schematically represents the transfer-learning procedure's progression from ImageNet to sim-produced data pools.

TRAINING SEQUENCE: RESULTS AND FINDINGS

Training on sim output (consisting of 1000 pairs of grayscale scenery & binary-valued truth masks) was performed with a (88%)/(12%) partitioning of samples used in training/validation. The selected loss metric was Multi-Class Binary Cross-Entropy (2D). Training was executed on a multicore CPU in 8-image "batches" which parallel-processed the gradient computations before combining those (through the ADAM optimization algorithm) into a single update of each model parameter. Cyclical learning rates were employed in two stages: first when freezing all but topmost layers (16 epochs), then for fully-unfrozen layers (16 epochs). Training duration was ~24 hours and the ~95% accuracy had not yet exhibited signs of encountering the overfit limit of achievable performance.

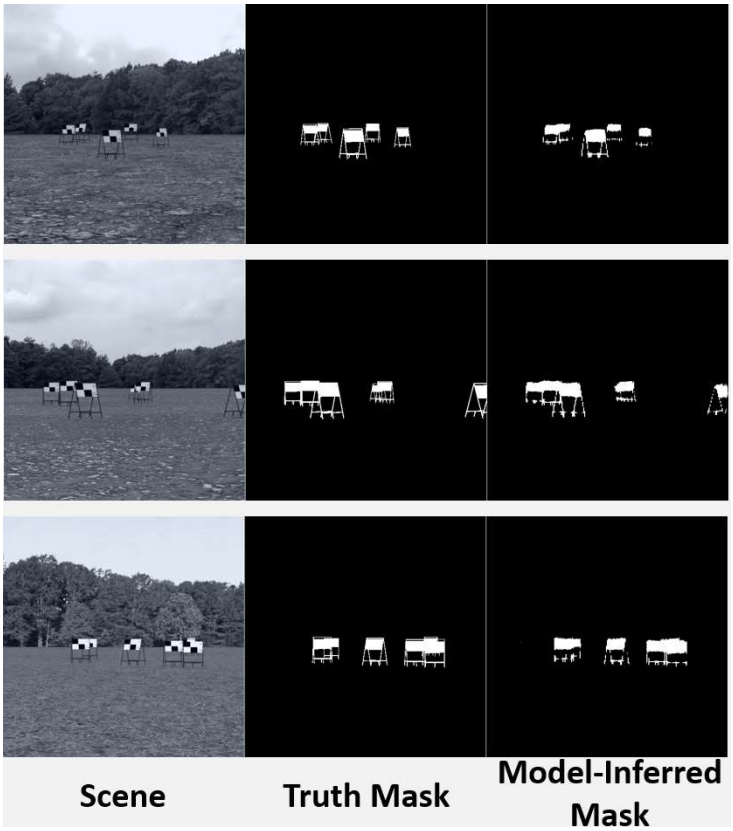

Figure 5. Model Validation Examples (Synthetic Images)

Figure 5 displays three examples of model performance on images from the validation set. The ~95% quantitative accuracy is observed to still carry visible qualitative degradations in model-inferred masks (compared to sim-provided truth mask). Examples of qualitative findings include (1) the tendency to omit fiducial struts for fiducials placed larger distances from the camera and (2) difficulty in determining the horizontal top strut when a fiducial is placed against a dark background. These findings agree with what a human would be expected to encounter if faced with the same challenging context.

RUNNING SIM-TRAINED MODEL ON FIELD IMAGERY

Following training on sim-data, the model was provided imagery from a limited-volume (~10 samples) field imagery data set. Because a form of "truth" segmentation did not exist for this set, assessment focused on qualitative findings from visual comparison over these limited number of image samples. Since field test imagery was not cleared for

public release, results are described in terms of performance upon specific imagery features (e.g., background scenery, brightness levels, etc.) observed in these limited number of tests.

The selected field imagery tested the algorithm operation by presenting a span of imaging features/conditions, with some going beyond those covered in the simulated training data sets. By assessing performance in these conditions, an initial indication of performance sensitivity may be obtained (providing guidance on how closely the simulated data must align to test conditions). Examples of field imagery conditions – with noticeable departure from sim training conditions – include the following.

- *Background scenery*: Field imagery included desert landscapes with mountain backgrounds and rocky terrain in addition to the forested backgrounds in simulation training.
- *Lighting*: Field imagery exhibited more diversity in lighting models, including cases in which the sun was directly-overhead. This resulted in increases to both background level and background contrast relative to that of fiducials.
- *Additional objects*: Non-fiducial test apparatus were visible in most field imagery cases. In some cases, apparatus was placed at ranges near (but without obstructing) fiducials; in others, apparatus were placed behind fiducials at distances sometimes exceeding the sensor's focal range.
- *Fiducial configuration*: Field imagery fiducials varied in the checkerboard pattern displayed: some had the black squares in a "back-slash" configuration as in ***Figure 5*** while others had the pattern flipped ("forward-slash"). Also, due to the presence of other camera groups, some images included fiducials not facing the imaging sensor. Most fiducial stands were ground-secured with sand-bags (absent in sim training data). One image set had near-range fiducials with 1D span of ~300 pixels.
- *Optical effects*: Field sensors were focused at the range in which fiducials were generally placed. Cases with distant backgrounds or far-range apparatus had degraded resolution of such objects. Recall that simulated data was performed without range-based blurring effects.

The segmentation masks resulting from processing field imagery through the sim-trained NN were visually compared to the field imagery through overlaying the semi-transparent mask onto the original image. From this ~10 sample set of images, the following observations were made:

- *Relative features*: Throughout all cases, the fiducial frame was more reliably segmented than the checkerboard pattern. This is believed to be primarily due to the discrepancy in the checkerboard "slashed-ness".
- *Impact of contrast*: Performance was degraded when fiducial brightness level and contrast were comparable to the scene background. This indicates a parameter in which the training data should be augmented.
- *Additional objects*: The segmentation performed generally well in rejecting non-fiducial objects such as test apparatus. One image included an apparatus of pipes arranged with a structural portion in a rectangular, "Π"-shape for which the segmentation algorithm classified as fiducial for a few pixels near the rightmost joint. Due to this shape's similarity with fiducial frames, this behavior appears consistent with the network having learned the relative arrangement of fiducial features (such as vertical/horizontal bars in the fiducial frame).
- *Fiducial size*: For the single image with fiducials at a 1D span of ~300 pixels, the segmentation algorithm failed to classify the fiducial. Because the algorithm was trained on images having ~50 pixel maximum 1D extent, this result indicates where test/training departures significantly impact performance.

For cases in which lighting and the fiducial pattern most closely aligned with training conditions, both the fiducial pattern and the fiducial frame were found to be segmented with higher accuracy. Lighting appeared to be the dominant factor as evidenced by cases having lighting similar to that used in training – but with forward-slash checkerboarding instead of back-slash patterns – resulting in an estimated 90% accuracy for total pixels in each fiducial.

The suitability of this algorithm's results will depend upon the application context. For baseline multi-sensor registration procedures which only require selecting a visual anchoring point for each fiducial, this segmentation algorithm's results suggest adequate performance for the existing training data/procedure. This is due to the algorithm's ability to capture fiducial macro-features (i.e., spatial extent, through the reliable extraction of structural frames). Should one extend this network architecture to also estimate fiducial pose (for precision registration), it would be recommended to conduct another training iteration on augmented conditions, primarily accounting for (1) more general fiducial patterns (e.g., both back-slash & forward-slash checkerboards) and (2) more general lighting (especially for cases in which fiducials and background levels are similar).

RETROSPECTIVE AND OUTLOOK

This study has illustrated workarounds to roadblocks (exorbitant computational expenses and limited size data pools) when training ML models:

- training procedures which take advantage of dynamic learning rates to expedite development of proficiency,
- using simulated scenes as an intermediate transfer-learning step (when limited field-data is available),
- modeling fine-detail scene content through textures instead of polygons,
- feeding-back the assessment of a sim-trained model's performance on field imagery into the next-iteration of training on sim data, and/or
- performing fine-tune training upon physical imagery.

An additional approach (not explored here) would be to obtain two sources of sim data (e.g., high-volume/low-fidelity ray-*cast* imagery vs. low-volume/high-fidelity ray-*trace* imagery) and stage the transfer-learning procedure to progress through these in order (i.e., break-out the sim-training step into two). A suggested heuristic would be to ratio the number of images produced in these pools according to each sim's relative duration in generating a single scene/mask output (e.g., if ray-tracing took 100x increase in duration, it would have 100x fewer samples).

Though class-activation and saliency maps are recommended for general use, it is unsure whether they would perform effectively in resolving small structural features (such as presence of discrete polygons in the structure of CAD models). A variety of saliency & class-activation map algorithms are being evaluated for potential use in identifying ties of model performance to simulation structure.

CONCLUSIONS

This study has identified – through an example implementation – approaches for using simulation data to support a multi-stage ML training procedure. Guidance was also provided on approaches for detecting and managing the significance of sim-introduced artefacts. Figure 6 highlights the general procedures of this study's implementation.

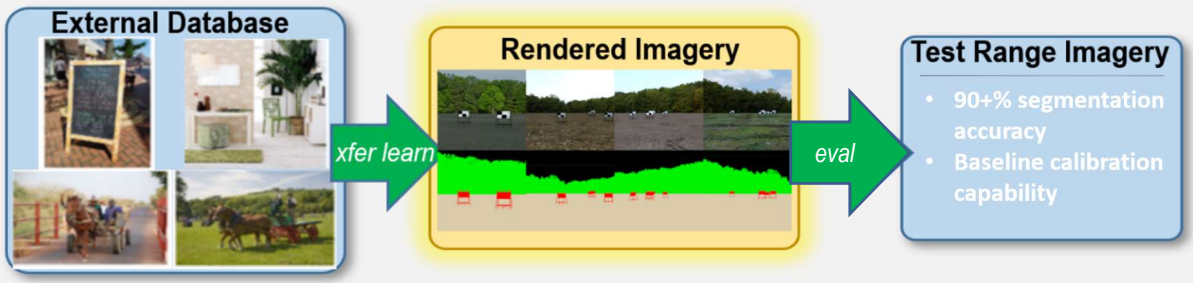

Figure 6. Training Workflow Implemented in This Study

This study highlights the fact that a transfer learning implementation – in the form of intermediate simulations at with appropriate combinations of fidelity and volume of simulated output – may serve as an effective alternative for training ML solutions than requiring large volumes of high-fidelity data exclusively.

REFERENCES

Bengio Y., Practical Recommendations for Gradient-Based Training of Deep Architectures, eprint arXiv:1206.5533, Retrieved June 14, 2019, from https://arxiv.org/pdf/1206.5533v2.pdf

Deng J., Dong W., Socher R., Li L. J., Li K., Fei-Fei L., ImageNet: A large-scale hierarchical image database, *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*

He K., Zhang X., Ren S., Sun J., Deep Residual Learning for Image Recognition, *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*

Howard J., and Ruder S., Universal Language Model Fine-Tuning for Text Classification, eprint arXiv:1801.06146, Retrieved June 14, 2019 from https://arxiv.org/pdf/1801.06146.pdf

Ronneberger O., Fischer P., Brox T., U-Net: Convolutional Networks for Biomedical Image Segmentation, *Medical Image Computing and Computer-Assisted Intervention – MICCAI*, 2015

Smilkov D., Thorat N., Kim B., Viegas F., Wattengerg M., SmoothGrad: Removing Noise by Adding Noise, eprint arXiv:1706.03825, Retrieved June 14, 2019, from https://arxiv.org/pdf/1706.03825.pdf

Smith L. N., Cyclical Learning Rates for Training Neural Networks, *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*

Zhou B., Khosla A., Lapedriza A., Oliva A., Torralba A., Learning Deep Features for Discriminative Localization, *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*