Postdoc First Year Projects and Progress

Audience/Presented to ???

Month Day, Year

Robin Yancey
First Year Applied Machine Learning PostDoc

LLNL-PRES-XXXXXX

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

About Me

- EE BS/MS CS PhD with focus in deep learning, computer vision, big data/parallel
- New PostDoc in Machine Learning
- Hired to help with ML in Laura's group on Damage Analysis (AMH* OMF*)
- Now also working on DGP* LDRD
- Thanks to everyone but especially Laura, Chris, Scott, and Connor for helping introduce me to LLNL, NIF, OI, and MUCH more!!
  - **And, especially for providing details on their projects and areas where they could use my help improving**
  - **So far best team I could ever ask for**

* See next slide for project acronyms

Brief overview of the various mechanisms/protocols employed within the NIF Optics recycle loop which include deep learning tools I've worked on

- **All Microscopy Hitlist (AMH):** After the Final Optics Damage Inspection (FODI) imaging system detects enough damage sites on an optic installed in the National Ignition Facility (NIF), it is removed for a full microscopic scan. The AMH analysis consists of images of everything from this scan above a given size.
  - Deep learning is used to determine which of these are actually damage (and not some other non-damage subclass [RAM cone, debre, particle, ect. eg. slide 15]) so that it can be sent to OMF to be repaired.

- **Optics Mitigation Facility (OMF):** The OMF is where damaged optics are repaired by placing a RAM (Rapid Ablation Mitigation) cone over each damage site, in order to prevent the site from absorbing energy required to power the (NIF)
  - Deep learning models are used after the repair process to determine when all remnant damage is fully removed by the repair machine.

- **Damage Growth Prediction (DGP):** We currently estimate the potential amount of growth of a damage site in order to make a decision on whether or not a site should be mitigated based on the site estimated Equivalent Circular Diameter (ECD) obtained from a Nikon image.
  - Deep learning techniques are being developed to see if we can make a better estimate of the amount of growth based on higher detail Laser Confocal Images (LCI), fluence, and Photoluminesence.

PROJECT SUMMARIES SLIDES

Main Year 1 Projects (so far)

1. AMH Deep Learning Model Training/Evaluation Framework
2. AMH Deep Learning Model Production Deployment Framework
3. AMH/OMF/ETC. Deep Learning Multi-Model Standardization
4. Damaged RAM Cone Data Processing & Model Training/Evaluation/Test Framework Development
5. DGP Data Processing & Model Training/Evaluation/Test Framework Development

AMH Deep Learning Model Training /Evaluation Framework can be improved

- **Problems:**
  - Due to changes in optics processing procedures data drift is a common problem
    - Can cause unexpected drops in accuracy can lead to downstream issues including damage not being mitigated
  - We have no knowledge of the specific types of damage/non-damage (eg. slide 15) that are on the optics
    - However, this info could help us make better decisions in the future regarding what protocols to use
  - Slight increases in accuracy are extremely helpful in production due to the high number of AMH classifications made per day
    - Improving overall accuracy (of Damage vs Non-damage classification) by changing the base model could save a lot of time/money and has yet to be explored

Solution:

Rewrite training code from scratch with the following capabilities

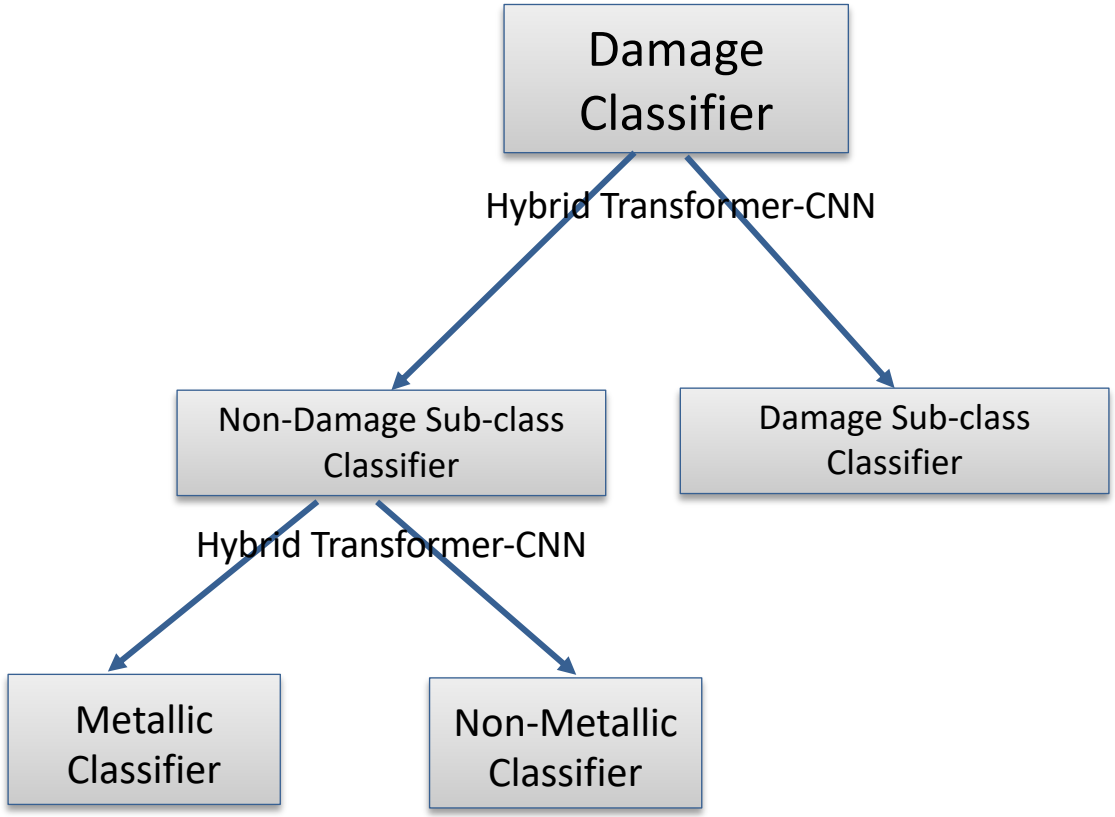
- Expand classification to Damage/Non-Damage subclasses, without significant loss in accuracy
- Include any possible increase in the accuracy/robustness of base model (currently using ResNet18)
- Uncertainty assessment and bad classification feedback would be helpful to avoid common issue of data drift

Model (Pretrained on)	Accuracy	Accuracy with Data Augmentation
ResNet50 (ImageNet1k)	0.975	0.979
ResNet18 (ImageNet1k)	0.978	0.980
Swin Transformer (ImageNet1k)	0.980	0.980
ConvNext (ImageNet1k)	0.980	0.980
Hybrid (ImageNet1k)	0.982	

PROGRESS: AMH Deep Learning Model Training/Evaluation Framework

Implementation:

- Built a transformer CNN ensemble model to predict damage vs. non-damage with over 98% accuracy
  - Improved accuracy over current base model *when tested with same training setup*
- Built a sub-model to classify non-damage *subclasses* with accuracy within 2-3% of binary class prediction accuracy
  - Discovered that a state-of-the-art transformer-based was model better at classifying non-damage subclasses over damage sub-classes (which were less important to classify)
  - Handled missing data, class minority/imbalance
- Wrote specialized AMH data-loader, train, evaluate, and test functions to run on LC GPU eg.
  - Easily reads proper data format, trains, and evaluates model with given set of input hyperparameters (saves best, last models)
  - Reports loss, accuracy, and confusion matrices
  - Provides class, subclass confidence, exports test data input predicted with lowest confidence to directory for further analysis



Block Diagram of Subclasses Evaluation

Machine Learning Standardization for OMF, AMH (and future models to be added)

Problem:

- All models, data and train/test code frameworks highly unorganized
  - Currently stored in all different places and use completely different codes/libraries/frameworks/data-structures
  - Inaccessible to group members besides the person who created it and to new members who join our group

- *It turned out we were also not able to reproduce a previously stated AMH top accuracy (within 1%) so I ended up rewriting the AMH model training code from scratch 4 different times, and implementing every possible detail in the code in every possible way*
  - Even tried running original AMH code (by the first author) a few months later when it had then eventually been provided afterwards (and it produced the same accuracy as mine)

- **Solution:**

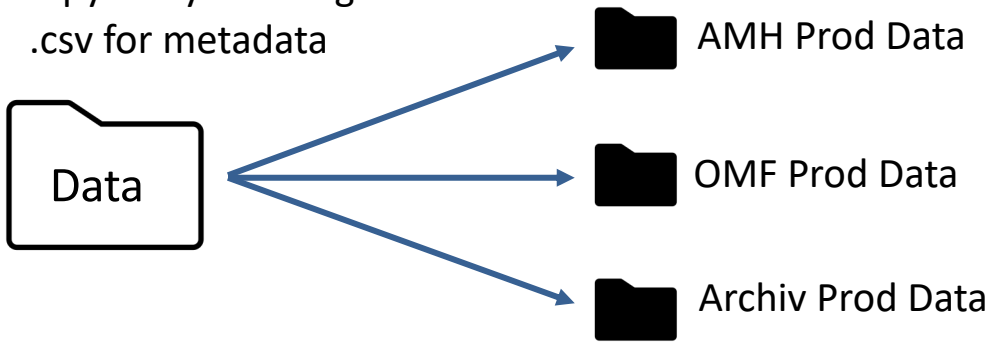
- Use code rewrites to help with 3 main things:
  - Final version was a simple model I could use to give to the intern as a starting point for ML standardization
  - Gave me a ton of experience running multiple/simultaneous tests on LC compute cluster (GPU's) to verify best setups
  - Helped solidify nearly all and ***exactly*** what we need in our standardized ML code framework (see details slides [in progress 2 or 4 done])

- Created a high-level organizational infrastructure we can use/follow for each project (eg. next slide)
  - Detailed list of requirements are to long of a list to add here (see details slides [in progress 2 or 4 done])

- Our new intern (Anand) is helping us move OMF into the new standards
  - He has standardized the data formats and the data loading methods
  - He also created an initial train/test code framework in Pytorch

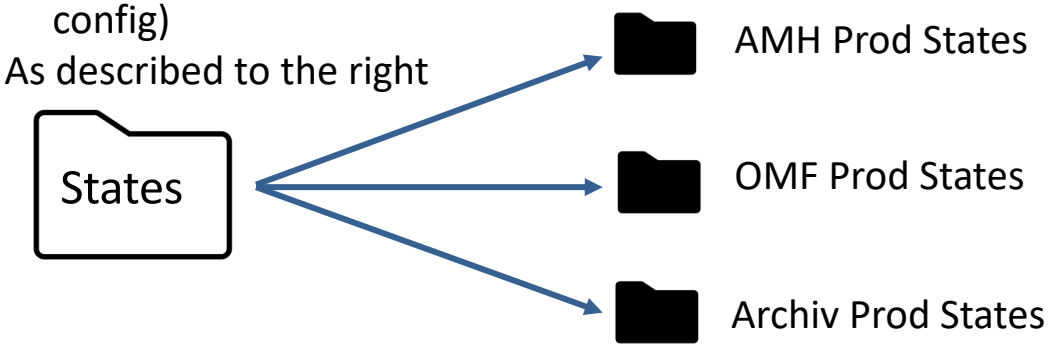
NIFOI ML Standardization Structure

DATA: Contains versioned input training and test data:
- .npy array for images
- .csv for metadata

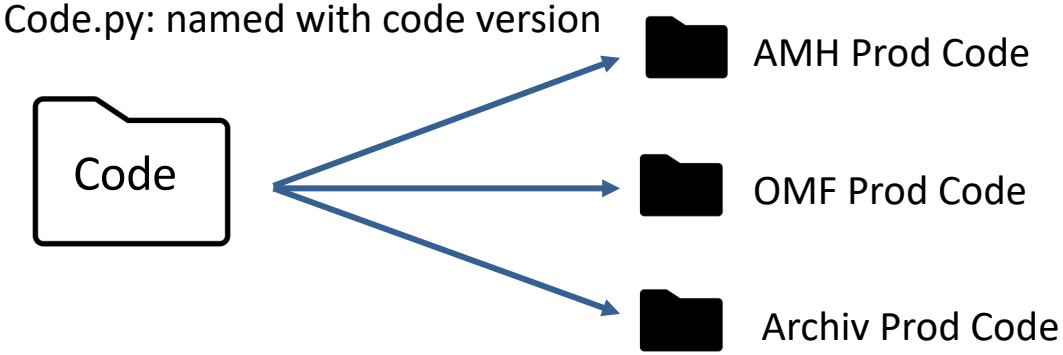


STATES: Contains directories which in turn contain:
- .pt files
- .txt files (or copy of code, config)

As described to the right



CODE: Contains train/test methods actually being used
Train-config.txt : any inputs that may actually need to vary in prod code
Batch.cmd: simply runs code on LC node
Code.py: named with code version



State Directory: (output from code), containing:
1. First, Last, Best states as .pt file:
   - {model.state_dict, optimizer.state_dict, epochs}
2. Associated .txt file:
   - Start state directory path (must be contained in states directory)
     - If starting from scratch list exact methods
   - Train/Test Data paths (must be contained in data directory)
     - Named with versions
   - Code version (eg. In case it is ever archived)

Automated production system AMH was Cyber blocked due to NVIDIA® CUDA™ (GPU and parallel processing toolkit)

Problem:

- Cyber/IT blocked AMH PROD version of CUDA
- Needed monthly exemptions to bypass block
- CUDA upgrade failed to load the DL models

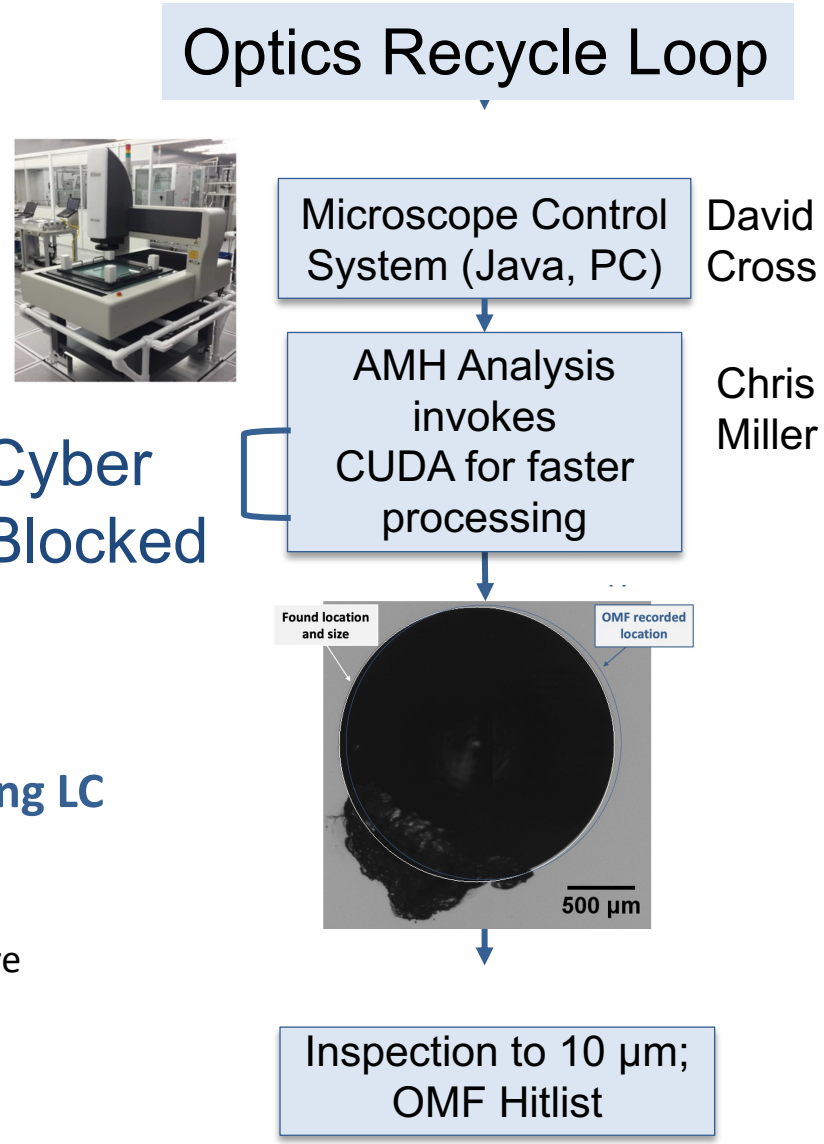
Solution: ***Re-wrote the multi-threaded C++ frontend (model application to data) code from scratch** to work with new version of CUDA

- Read image, average pixels in each (RGB) channel, merge ring/top/back-lit
- Crop, create image tensor, permute, un-squeeze dimensions
- Concatenate tensors into batch
- Send model and data to GPU, apply model, get returned values
- Get batch class & confidence with tensor ops, return list of all results combined

Implementation:

- ***Developed & Tested solution on multiple CPU's/GPU's including LC high-performance computer, "Pascal"**
- Wrote detailed reusbale test code to compare predicted classes and confidences
- While porting to Windows PC, found error in interface to Java
- Working with David and Chris to investigate the Java and PC interactions to resolve problem.
  - 72 different test models built (some worked but not the one we needed).

*** Learning Curve for me!**

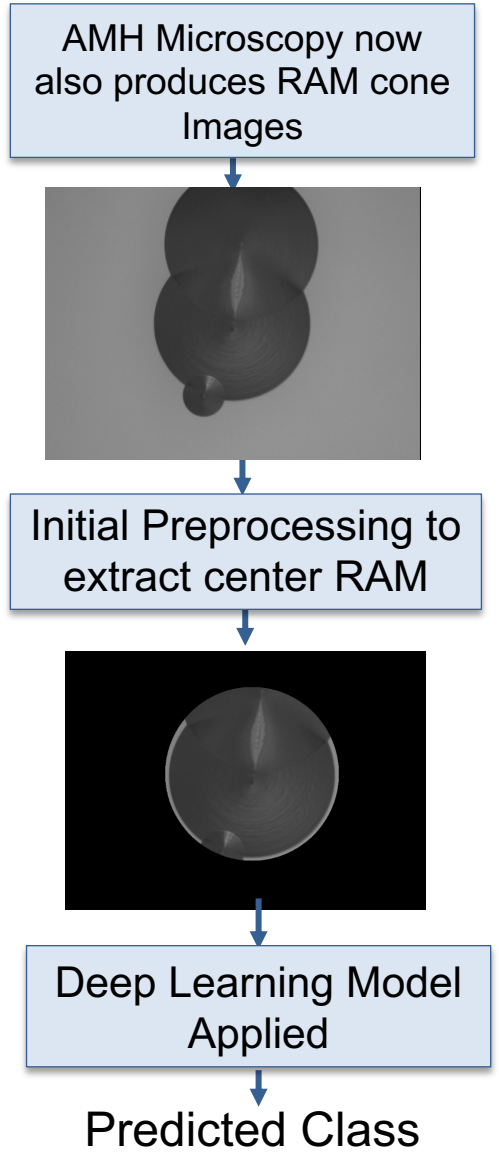


Understanding why RAM cones damage could help us avoid it in the future

Problem: RAM cones get damaged and often we don't know the cause
Solution: Develop a deep Learning model to quickly classify damaged cones on a large scale. This information will help us to find correlations (such as phase objects, protocol size, and/or fluence)

Implementation:

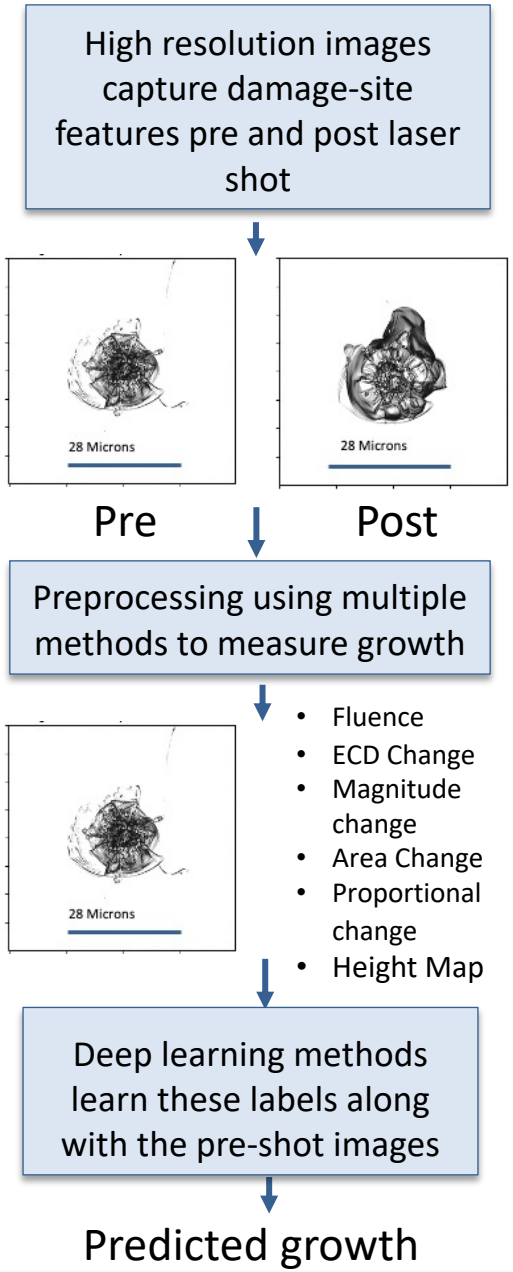
- Deep learning train, evaluation, and test code (described on separate slides) has been implemented
- Specialized data pre-processing code:
  - Input: CSV with image paths and corresponding classifications for training OR set of input filters to metrology to scan Metrology server log for RAM images of interest for testing
  - Algorithm: find center cone and count all overlapping cones (RAMEOs), check protocol (with input meta)
  - Output: 1. Array of images segmented center cone , 2. CSV of corresponding metadata for each image
- Test Progress:
  - K-fold testing
  - Per protocol testing
  - Protocol count added as metadata to model final layers
  - Data augmentation on minority class to improve (class averaged) accuracy
  - Test on >10k unlabeled server data (reported confidence, predictions for each in CSV)
  - Exported eval images missed and correct class labels
  - Next:
    - ROC curve
    - More test data for creating more classified training data



Machine learning model for better predicting damage growth

Problem: Some damage sites we mitigate which never would have grown large enough to cause harm
Solution: Develop a deep Learning model to estimate growth behavior of damage sites
Implementation:

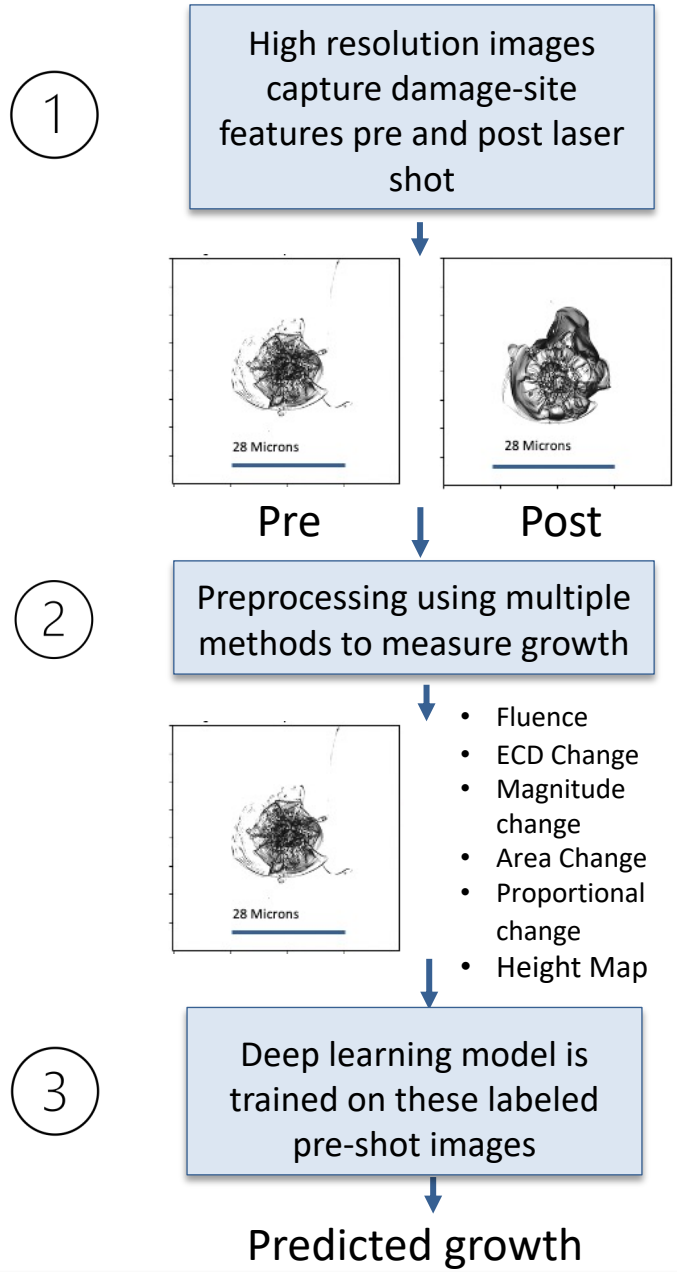
- Deep learning train, evaluation, and test code (described on separate slides) has been implemented
- Specialized data pre-processing code:
  - Input: vk4 files of pre and post LCI and height images, difference images (eg. next slide)
  - Algorithm: find/crop center damage site, remove noise (from each with special methods depending on dataset, image type), sum magnitude of pixels and sum pixels greater than threshold contributing to damage site for
    - Post – pre (height and LCI)
    - Difference image
  - Output: 1. Array of cropped pre-shot images, 2. CSV of corresponding metadata for each image
- Test Progress:
  - Height map image data included in all model layers alongside LCI
  - Compared various hyperparameters and accuracy measures
  - Proportional Growth vs Magnitude Growth vs Area growth comparison
  - Fluence metadata added into model final layers (adjusted to minimize MAE)
  - Test on other datasets (after lots of detailed analysis only *one* other dataset could be combined)
    - Tested various methods of normalization between train/test data
  - Compared various methods of noise removal for both LCI and height data
  - Next:
    - Compare MAE in microns with various labels



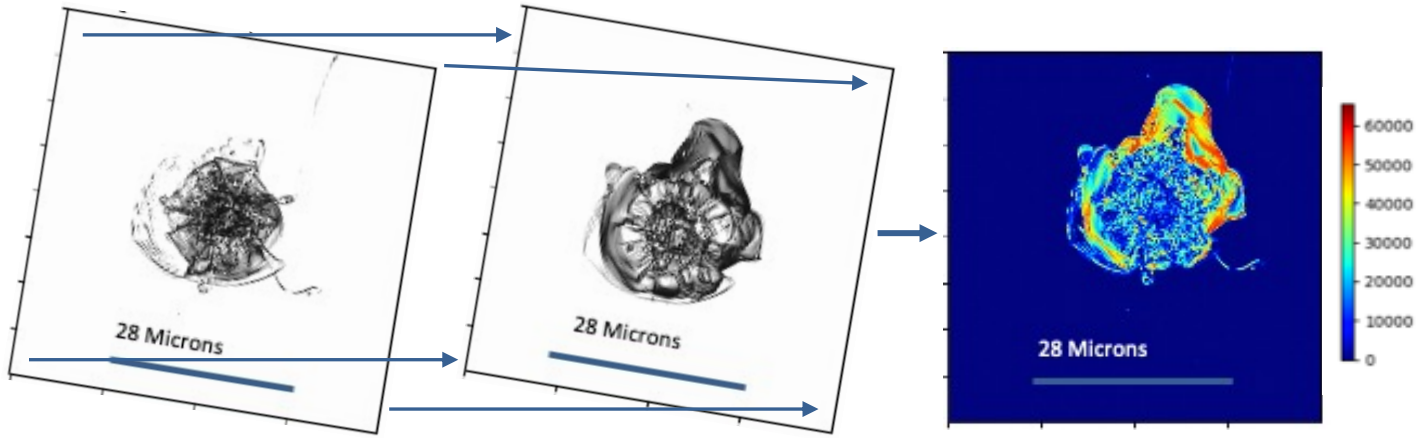
Deep learning model for prediction of laser-induced damage growth on NIF optics

Robin E. Yancey, Christopher F. Miller, Jae Hyuck Yoo, Yoonsoo Rho, Christopher W. Carr, Ted A. Laurence

- Recycling of the National Ignition Facility (NIF) fused silica final optics allows regular laser operation above the exit surface damage growth threshold.
- **Problem:** While large damage sites tend to grow exponentially upon additional laser exposures – requiring repair when an optic is recycled – smaller sites grow in a stochastic manner, making the necessity of repairing an individual site ambiguous.
- **Solution:** Develop a deep Learning model to estimate growth behavior of damage sites
- **Implementation:**
  1. Gather Data, 2. Preprocess data for ML, 3. Train/Test Deep Learning Model



DGP pre-processing uses precise measurement of damage growth to produce growth labels for each pre-shot LCI image



Pre-shot Image | Post-shot Image | Absolute Difference Image

2

Growth Magnitude Label (Y_{True}) =

$$\frac{1}{\max(|I_{post}-I_{pre}|)} \int |I_{post} - I_{pre}|\, dA_{I>Imin}$$

After the background and noise is removed from the difference image, the summation of the absolute value of all pixel intensities leftover from within the damage site is calculated. This is used to precisely quantify growth behavior associated with each pre-laser-shot image and effectively train the machine learning model using this information.

1

Corresponding pre and post laser shot damage images are aligned with Fourier Mellin Method and then subtracted from each-other element-wise to capture both positive and negative change information as a "difference image" (shown above).

3

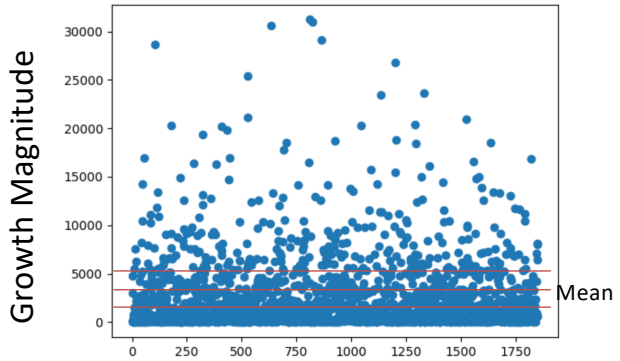
- Damage site size contributes to higher likelihood of growth, so the MAE (Mean Absolute Error) is also calculated with *growth with respect to the pre-shot size* (right) as the label in order to ensure test accuracy is maintained.
- Proportional Growth is equal to Growth Magnitude divided by the Pre-Shot damage size ($\frac{Y_{True}}{Y_{Pre\_Size}}$).

Pre-shot Damage Size ($Y_{Pre\text{-}size}$)=

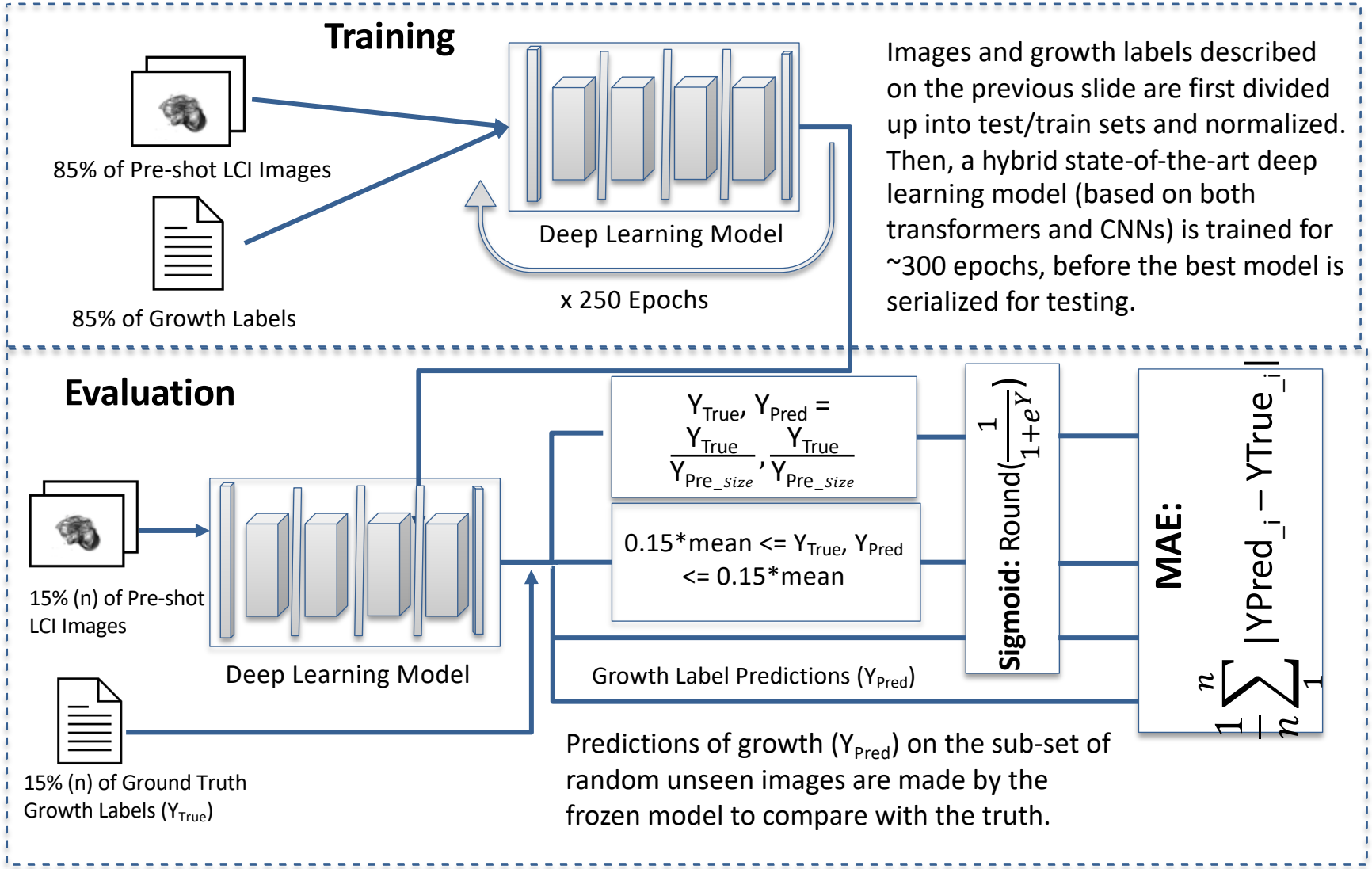
$$\frac{1}{\max(I_{pre})} \int I_{pre}\, dA_{I>Imin}$$

- MAE (Mean Absolute Error) for a mean cutoff are used for model verification and will be defined on the next slide.

Overall Distribution of Sub-dataset



DGP state of the art deep learning is used to estimate growth level



Method	2340 Data
Proportional/Binary Accuracy	**~0.83**
Bandpass Filtered Test Data (excludes points near threshold) Accuracy	**0.93**
Binary Accuracy	**0.88**
MAE	**0.31**

PROJECT DETAILS SLIDES

AMH DETAILS SLIDES

Can we predict sub-classes of damage and non-damage in AMH?

Issue	Specific	Solution
Missing Data Labels	Missing sub-class labels for most of the data (only labeled Damage/Non-Damage)	First separate *all* data into one of the 2 and then train a sub-model to predict sub-class from reduced set of possibilities
Overlap of sub-classes	Some classes contained a combination of 2+ different labels (eg. 'Particle' class contained 'Non-Metallic Particle' and 'Metallic Particle')	First separate *all* data into one of the 2 and then train a sub-model to predict sub-class from reduced set of possibilities
Model architecture	Higher accuracy obtained by hybrid models for sub-class groups	Employ a hybrid model with multiple sub-architectures

The main obstacles to overcome were the missing data labels and decreased ability for the current classifier to distinguish various shapes.

Can we make use of ‘un-labeled’ data?

Test/Valid images total: 11085

Damage: {'Damaged RAM': 1, 'E. Fillament': 86, 'Growing Mussel': 1418, 'Mussel': 1725, 'Damage': 1828, 'Pansy': 2004}

Non-damage: {'!': 1, 'Coating Flake': 2, 'RAM Cone': 3, 'Input': 5, 'Ram Cone': 8, 'RAM': 19, 'Particle': 53, 'Coating Defect': 53, 'Blank': 70, 'Exit': 98, 'Metallic Particle': 182, 'TBD': 285, 'NonMetallic Particle': 621, 'AMP Scratch': 1201, 'Non-Damage': 1422}

Train images total: 44340

Damage: {'Damaged Scratch': 1, 'ISBE': 1, 'E. Fillament': 310, 'Growing Mussel': 5909, 'Mussel': 6695, 'Damage': 7076, 'Pansy': 8030}

Non-damage: {'!': 1, 'Mussel': 2, 'Input': 9, 'RAM Cone': 9, 'Ram Cone': 16, 'Coating Flake': 22, 'RAM': 59, 'Blank': 216, 'Coating Defect': 243, 'Particle': 246, 'Exit': 362, 'Metallic Particle': 812, 'TBD': 1091, 'NonMetallic Particle': 2502, 'AMP Scratch': 5067, 'Non-Damage': 5661}

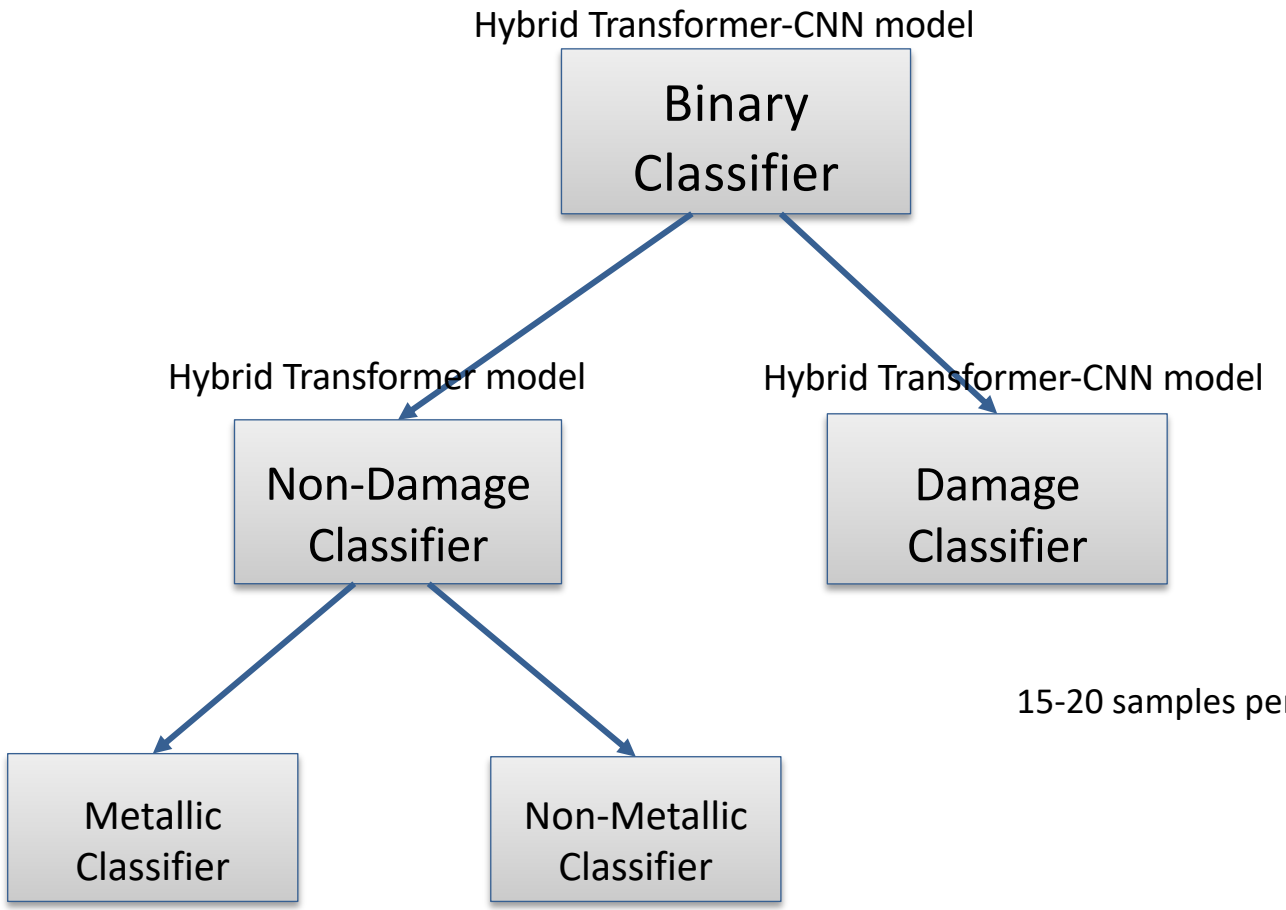
	Non-Damage Sub-classes	Damage Sub-classes	NA/Unlabeled
Train	10373	20947	13020
Test	2537	5233	3315

	Metallic Particle	Non-Metallic Particle	NA/Particle
Train	812	2502	246
Test	182	621	53

Dividing up classes and handling them individually increases accuracy with minimal increase in inference time

This method allows:

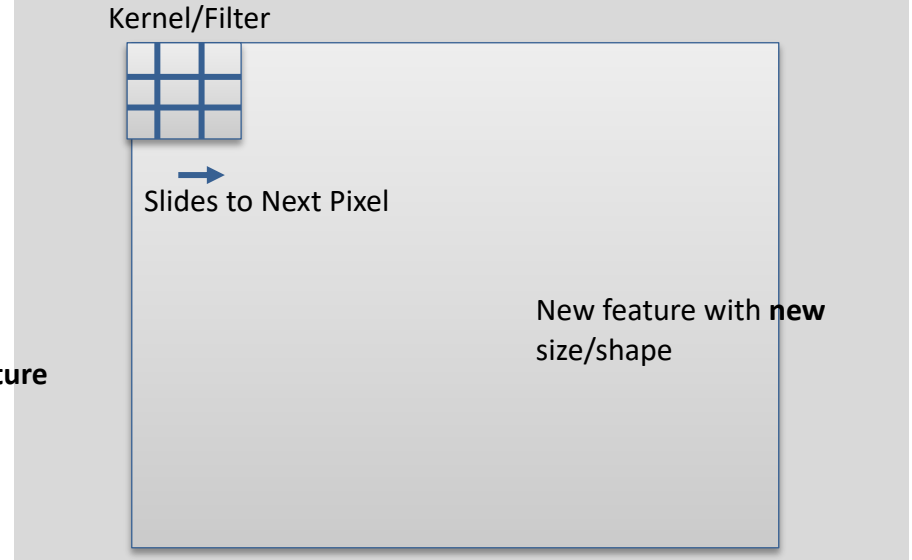
1. Use of multiple sub-architectures,
2. Reporting only Non-Damage subclasses
3. Increased accuracy by reducing possible classes.



15-20 samples per second test time

Convolution vs Attention for Feature Extraction

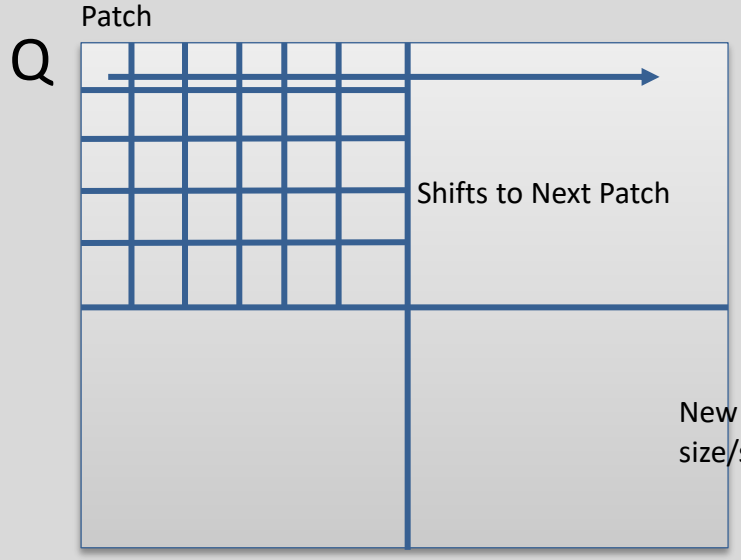
Convolutional Layer



Better at local/texture

$Pixel_i$ $Filter/Kernel_j$ Similarity

Attention Layer



Better at shape

Don't need successive convolutions to get distant interactions

Patch Q_i to Patch K_j Similarity

Where i is the location in the patch/kernel and j is the corresponding location in the next patch or next slide of the kernel.

Major difference is pixels are replaced by sum of SM(dot product) of corresponding patch locations from the entire image instead of just the kernel locations.

Non-Damage/Damage Classifier Model Comparison

	ResNet	Swin	ConvNext	Multi-Hybrid
Non-Damage Accuracy	97.6%	98.0%	97.8%	**98.1% (swin/conv)**
Damage Accuracy	88.9%	88.7%	89.1%	**89.4%**

Approx. 15 samples per second test time

text.

Non-Damage CFM: 98.1% accuracy

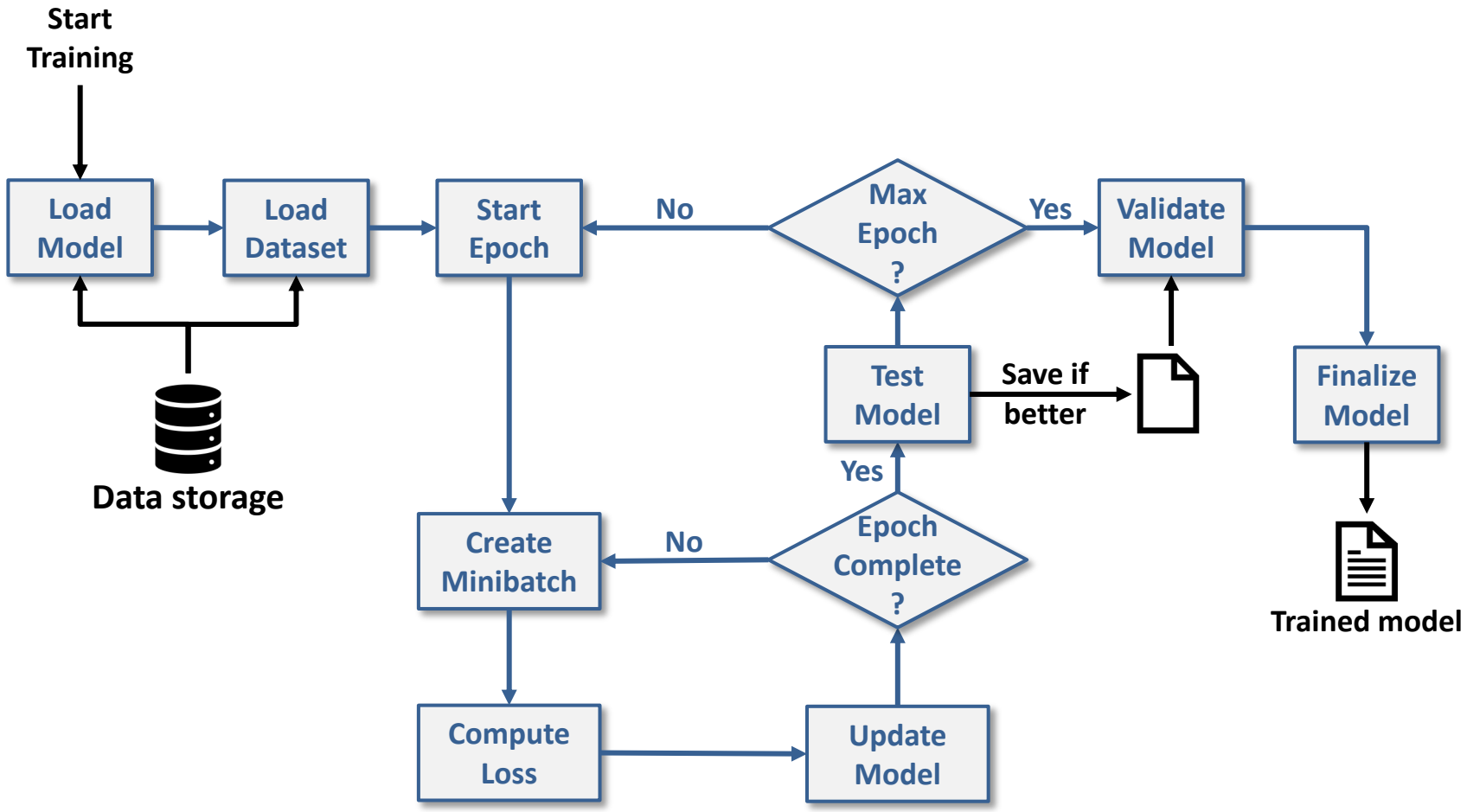
Predicted / Actual	Amp Scratch	Particle	TBD	RAM	Exit/Input
Amp Scratch	1196	3	2	0	0
Particle	3	834	18	0	1
TBD	1	15	269	0	0
RAM	0	0	0	22	0
Exit/Input	3	3	0	0	167

AMH Damage/Non-Damage Classification – Base Model Comparison

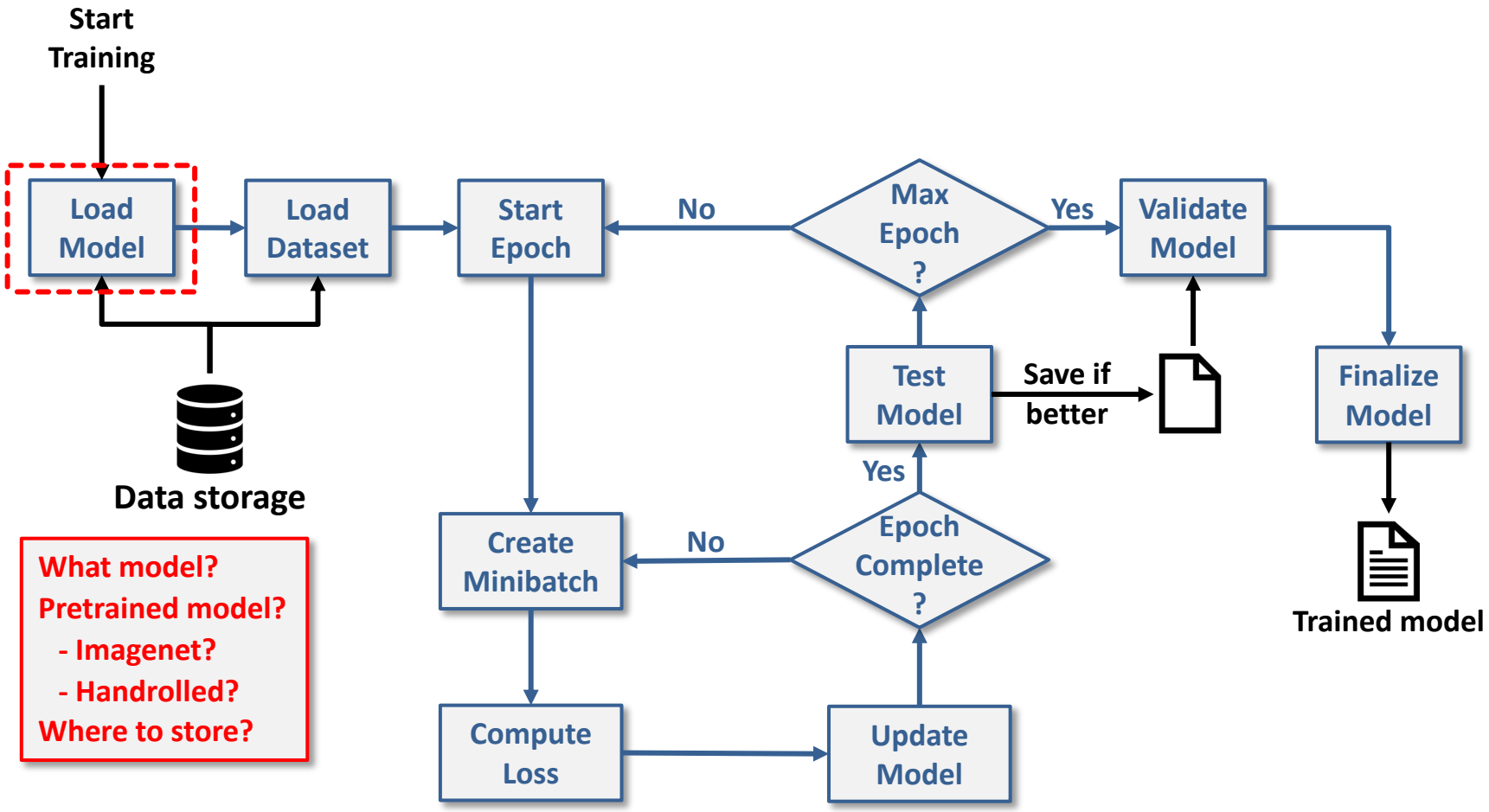
- Vanilla models each with same
  - hyperparameters (epochs 15, optimizer Adam, learning rate: 2e-5, batch size 8, drop-out, etc.)
  - feature extraction (eg. crop 448, resizing 224, and normalization of data)

Model (Pretrained on)	Accuracy	Accuracy with Data Augmentation
ResNet50 (ImageNet1k)	0.975	0.979
ResNet18 (ImageNet1k)	0.978	0.980
Swin Transformer (ImageNet1k)	0.980	0.980
ConvNext (ImageNet1k)	0.980	0.980
Hybrid (ImageNet1k)	0.982	

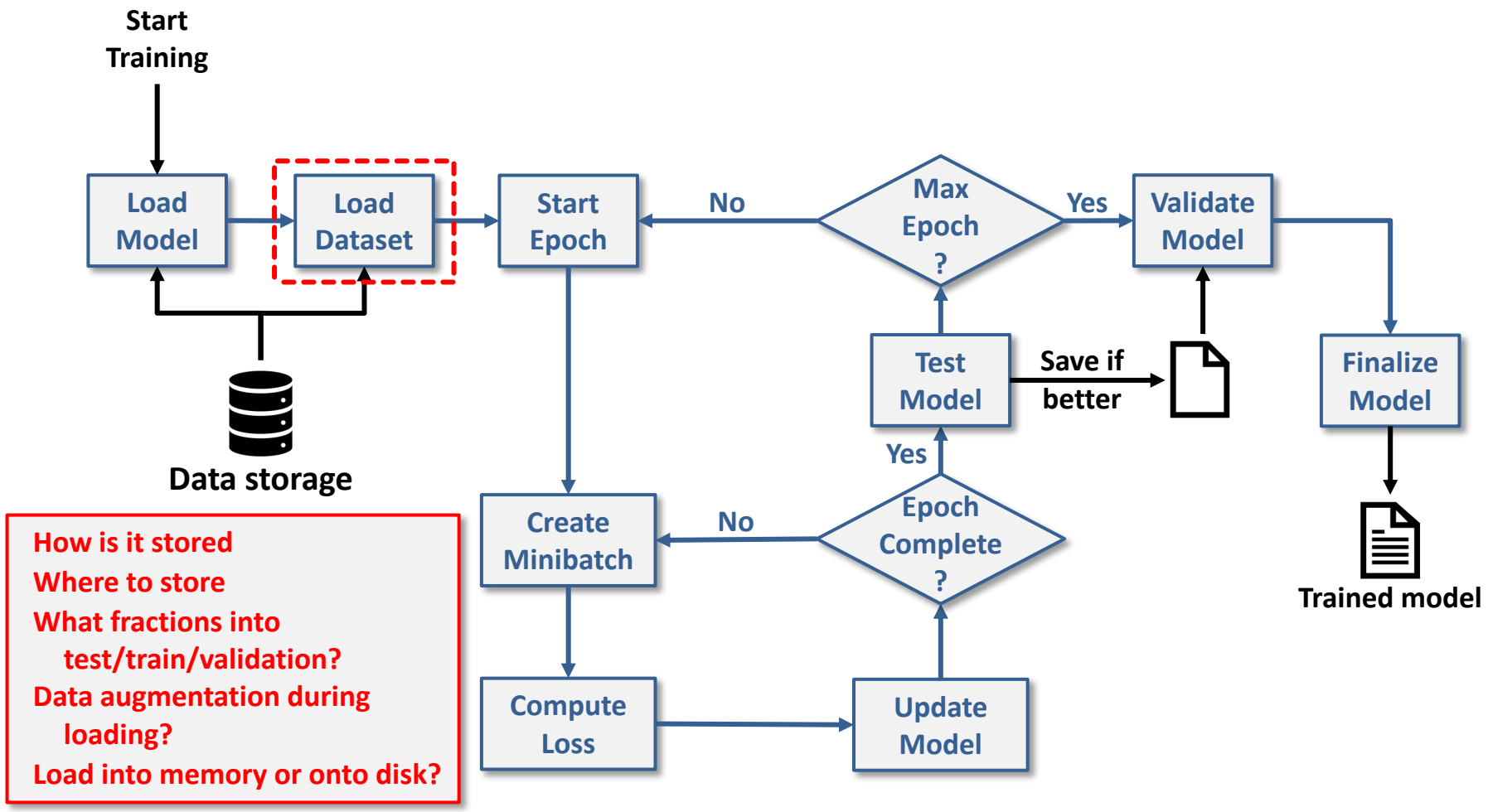
Basic classifier training process



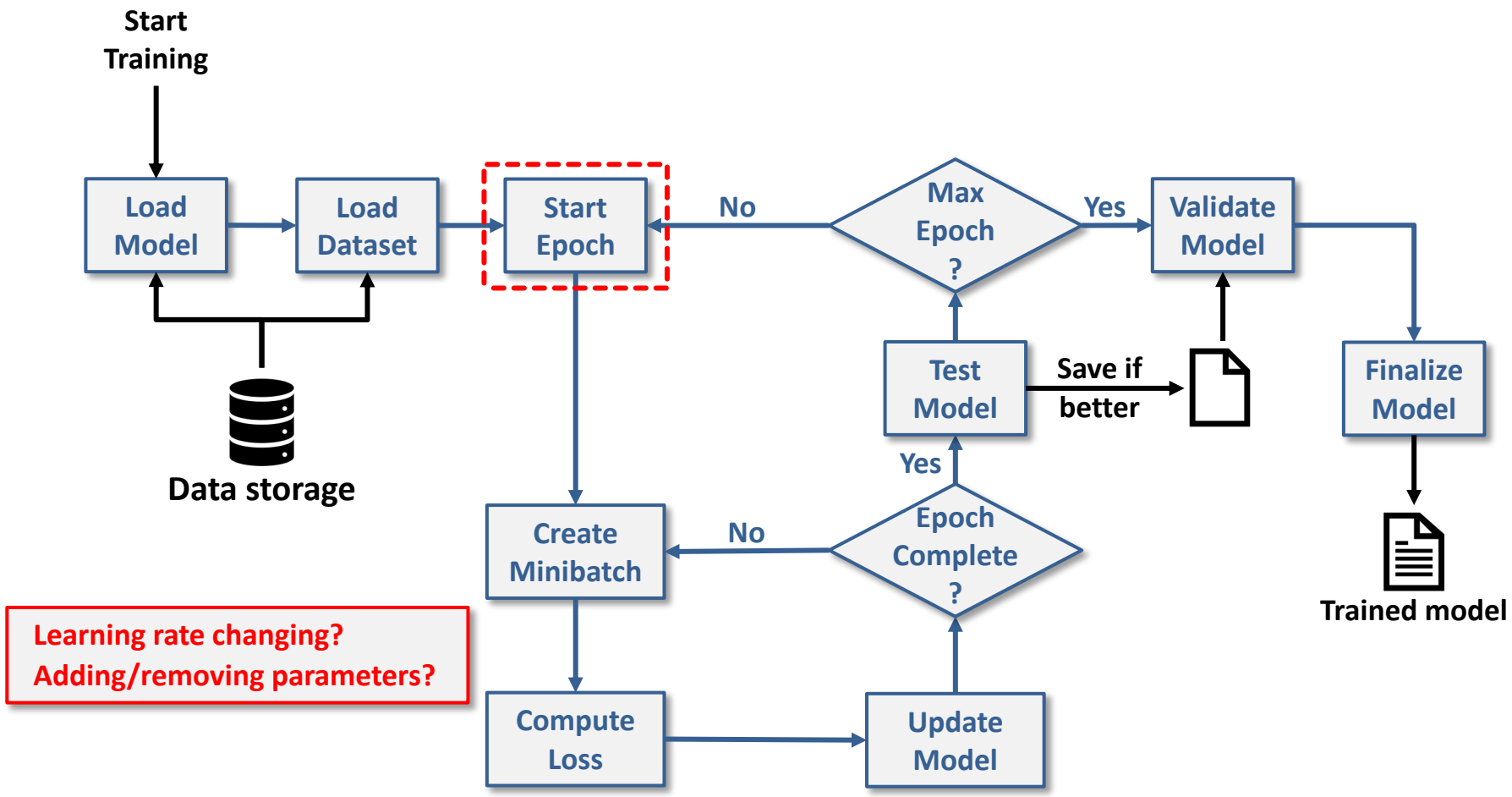
Basic classifier training process



Basic classifier training process



Basic classifier training process



Deep learning models standardized code must include 1. a data loader function

Input:

- Paths to 1 or 2 CSVs of metadata and corresponding NUMPY arrays of images
  - Test set is 2nd input array, CSV (if any)
  - Image index in array corresponds to metadata CSV row
    - CSV (column 0 index, column 1 image name/path, last is label if training data)
- Selections of Train, Test, and Evaluation sets to use
  - K and split size are the size and number fold of the train set to use for evaluation
    - Must include some of the minority class
    - Default: split # 0 15%
  - Input queries to CSV columns
    - May be different for Train, Test, and Evaluation split
- Batch size
- Data Augmentation Types:
  - In-training and upfront options
    - For upfront: may query specific CSV columns

Algorithm:

- Create data-loaders from image, label (meta) tuples converted to tensors
- Get mean, standard deviation of sub-data sets
- Create a function for all in-training data augmentation and normalization

Output:

- Data loader functions for train, test, and evaluation sets
- In-training Data Augmentation and Normalization functions

Deep learning models standardized code must include 2. a train setup function

Input: model, dataloader,

Algorithm:

- Detect class length to determine softmax or sigmoid
- Detect if label is continuous or scalar/class to determine accuracy measure
  - Need ex pred for testing only?
- Testing:
  - Write lowest confidence images for each class to directory
  - Write CSV (image, prediction, confidence [if classification])
- Evaluating:
  - Save first, best, last model
    - Optimizer state, epochs, params, output type
  - Confusion matrix, accuracy (or MAE, MSE for regression)
  - write low confidence and incorrectly predicted with class label as name to directory

Output:

- score

Deep learning models standardized code must include 2. a train setup function

Input:

Algorithm:
Output:

APPENDIX SLIDES

ViT (Vision Transformer) vs Swin (Shifted Window Transformer)

- ViT: Divides image into patches and creates a linear projection of patches which are a learnable embedding (each includes a position), to be fed into transformer encoder which includes self-attention, normalization, and residual layers before an MLP head
  - Attention layers create a hierarchical representation to encode structure and global information
    - *Self-Attention* compares every token pair in a sequence to get a score for each
      - Current token updated to be the weighted average of all input tokens (weights given by attention scores)
        - Queries key value to find similarity and get weight
      - Representation of tokens adapts so it becomes based on most relevant tokens
      - Input/output same dim
  - Shape bias (similar to humans); CNN texture bias

- Swin: Top performing transformer (~2021-2022); uses *shifted windows* which limit self-attention to non-overlapping windows (window shifts between layers, then output of neighbors merged)
  - Takes shifted window concept from CNN to capture more **local** self-attention (ViT more global)
  - Allows cross-window connections
    - helps with high res pixels, less shape bias, more generalizable
  - Patch number fixed so linear not quadratic with input like ViT

- *ConvNext* takes ResNet50 and morphs toward ViT design (making SWs more like patches)

Hybrid models Convnext

- Depth wise convolution
- Patchify layer
- BN -> LN
- AdamW optimizer
- RELU -> GELU