Pytorch Visual Transformer for InSAR APS removal

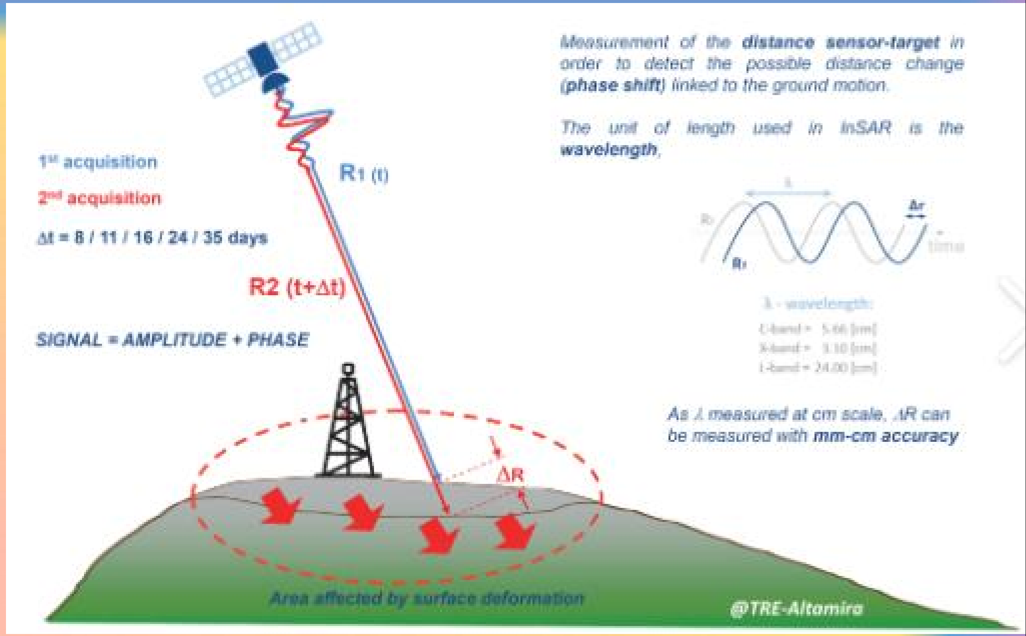
Luke Fairbanks

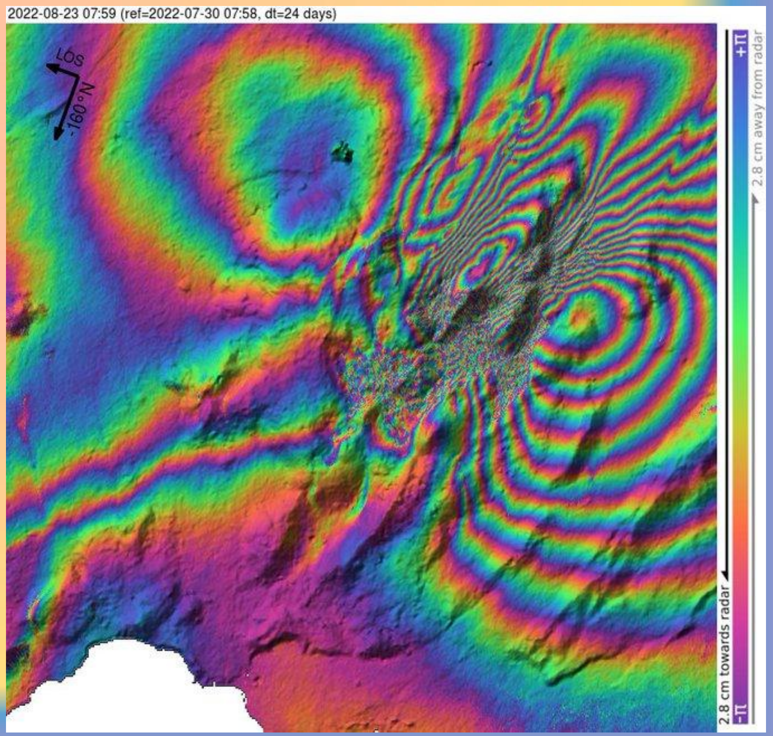
28/3/2024

Contents

APS in InSAR

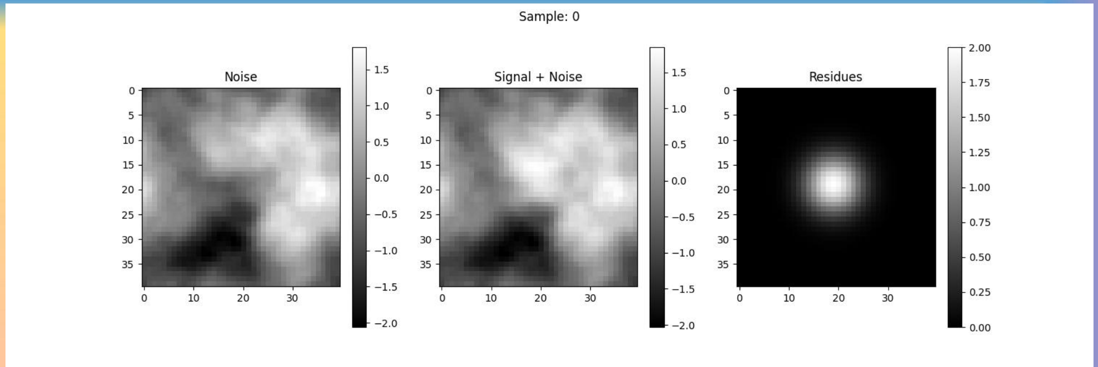
- Interferometric Synthetic Apeture Radar
- Atmospheric Phase Screen
  - Variations of refractive index of gas => noise
    - variable in time
    - variations possibly larger than displacement signal

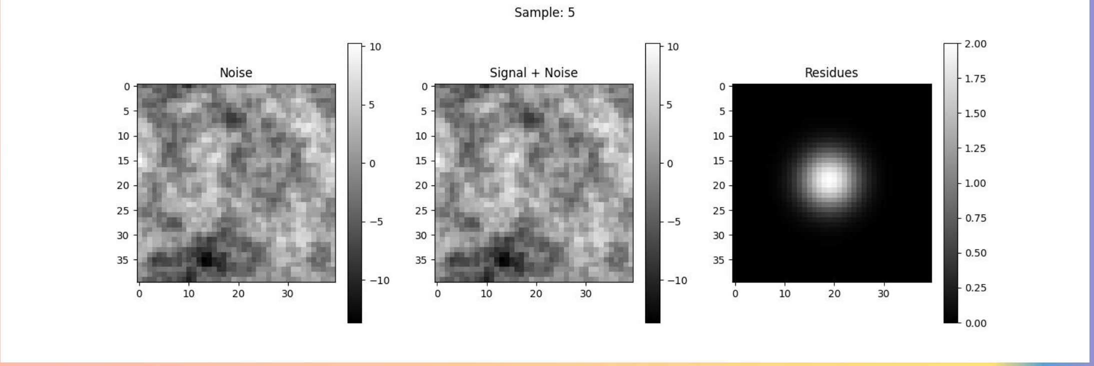


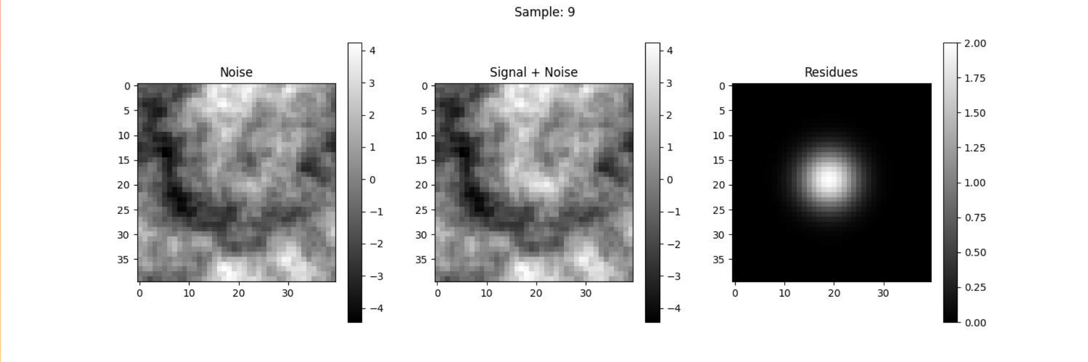


Test Dataset

- produced via
- signal + noise - ‘datm’, noise - ‘gatm’
  - signal - gaussian bump
  - noise - correlated power spectral, see ref
- [40x,40y,channel=1]
- 500 samples
- .grd files => xarray, numpy => png (torchvision)
- Credit Yuri Fialko







OG Noise

```
for i=1:num
 phase = make_synth_igram(n,m);
 aps(:,:,i)=C*(phase-mean(phase(:)));
```

```
uav=5*rand(1,1);
ftz=.4*rand(1,1);
ftx=.4*rand(1,1);
k=[0 10.^[-3:0.1:5]];
nexp=1.5+rand(1,1);
lc=5*rand(1,1);

Pk=1./(k+2*pi/lc).^nexp;
[igram,UK,K,UKv,Kv]=slipsyntr(X,Z,uav,ftx,ftz,k,Pk);
```

```
NZ=2*round(NZ0/2); NX=2*round(NZ0/2); % even numbers
% start with white noise with a unif. pdf:
U=rand(NZ,NX);


% wavenumber matrix (with 0 frequency at the center, even number of points)
dxm=dx*1000; % grid size, m
Kny=(1/dxm/2)*(2*pi);   % nyquist frequency * 2pi = max wavenumber
K=zeros(NZ,NX);
KX=[-Kny  : (Kny*2)/NX : Kny-  (Kny*2)/NX];
KZ=[-Kny  : (Kny*2)/NZ : Kny-  (Kny*2)/NZ];
[KXX KZZ]=meshgrid(KX,KZ);             % KXX and KZZ are nz*nz matrix
K=sqrt(KXX.^2+KZZ.^2);  % K in m^-1
Kv=reshape(K,NX*NZ,1);    % matrix to vector
```

Machine Learning Architecture: SimpleViT

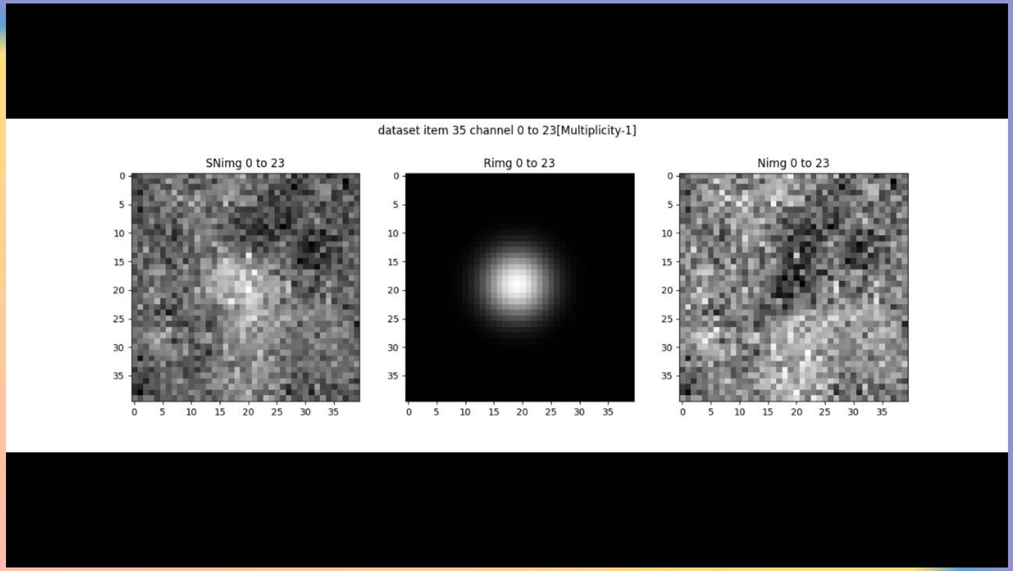
- Patch Embedding - linearly embedded into a high-dimensional vector space
  - non-overlapping patches
- Positional Encoding - 2D sinusoidal positional embedding added to the patch embeddings
  - capture spatial information
- Transformer Encoder Blocks - multiple transformer encoder blocks. block submodules:
  - Attention Mechanism - multi-head self-attention, captures global context
  - Feed-Forward Network - feed-forward neural network, processes attended features
- Layer Normalization - Layer normalization is applied before each AM & FNN
  - improves robustness & stabilized training
- Pooling Strategy - global average pooling to summarize the features
- No Dropout - Unlike the original ViT, SimpleViT does not use dropout
  - more simple, but worth considering other models TBD
- Linear Projection: A linear layer projects the pooled features to the desired number of output classes
- see references for more info regarding ViT architecture & variants

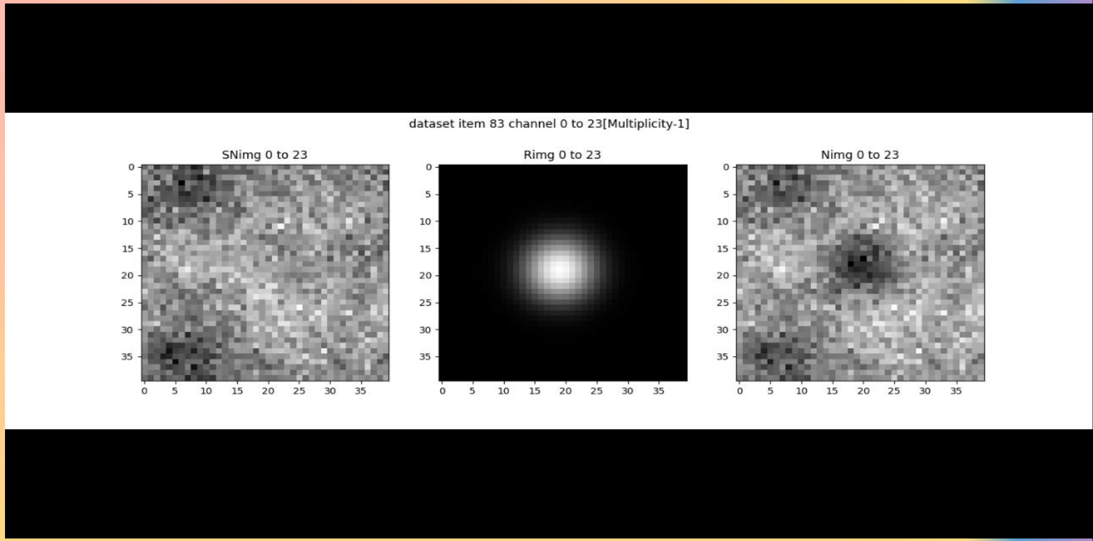
My main change? change from classification to transform/filtration by changing linear head @ end of network

- such that output of network is same dimensionality as original image tensor => loss fit to Res img tensor

Multiplicity param

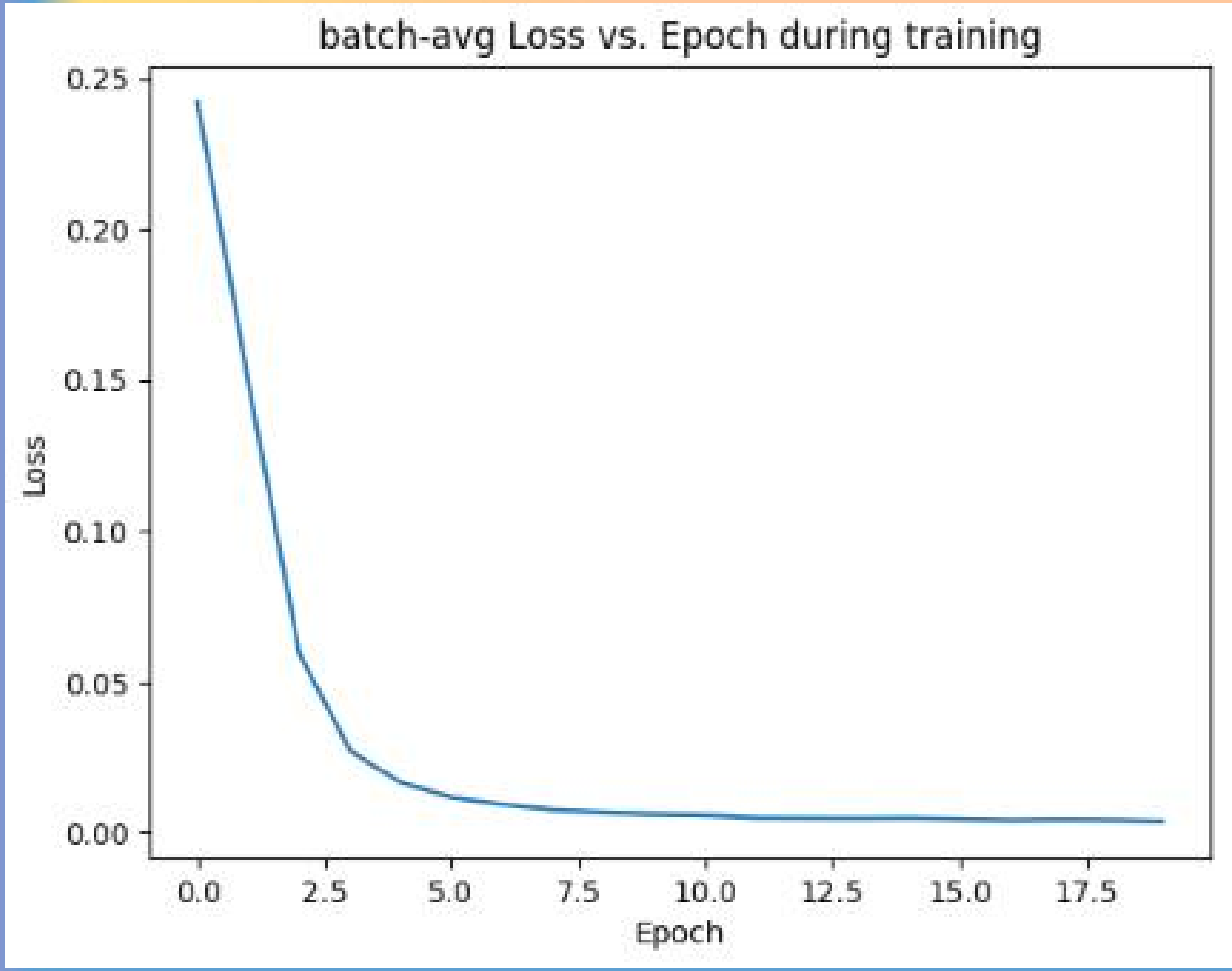
- CustomDatasetAPS.__getitem__
  - grabs single pair of [Signal+Noise, Residue = (Signal+Noise)-Noise]
  - simulate CSS methodology by noise resampling
    - single channel image to N channel
    - Residue copied (persistent signal, variations very small)
    - noise resampled
    - std = torch.std(SNimg)
    - noise_samples = torch.randn(self.multiplicity,*SNimg.shape)
    - noisy_channels = SNimg + noise_samples * std
- advise training on real data w/ moving window parametrized by ‘multiplicity’

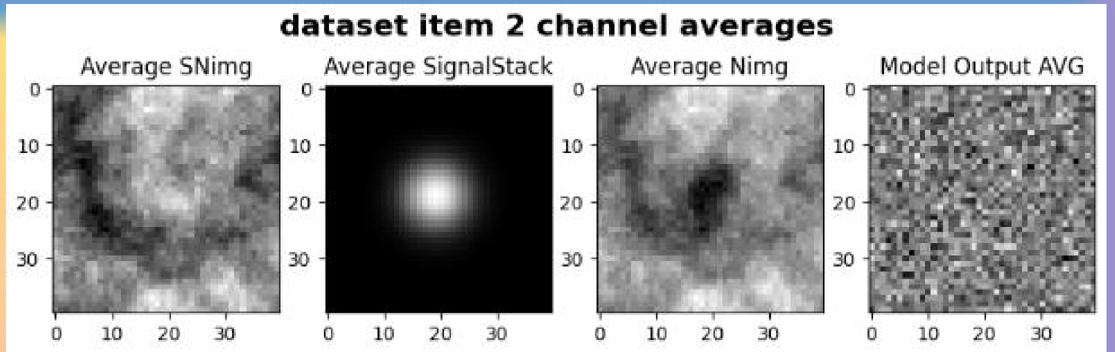


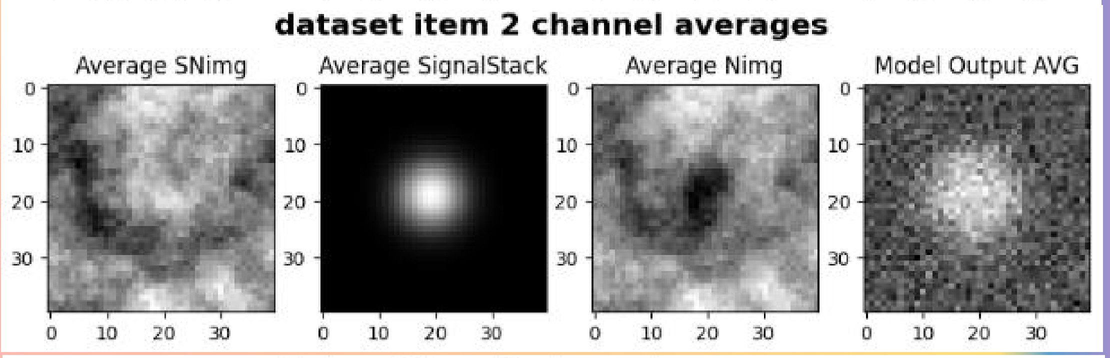


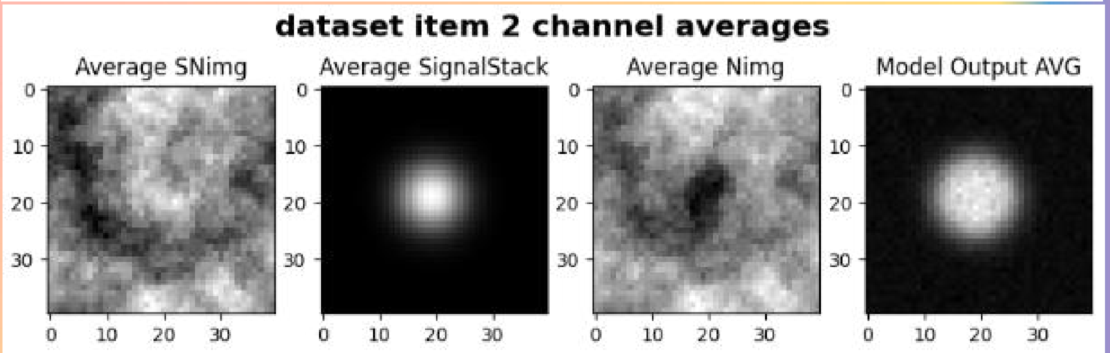
Results

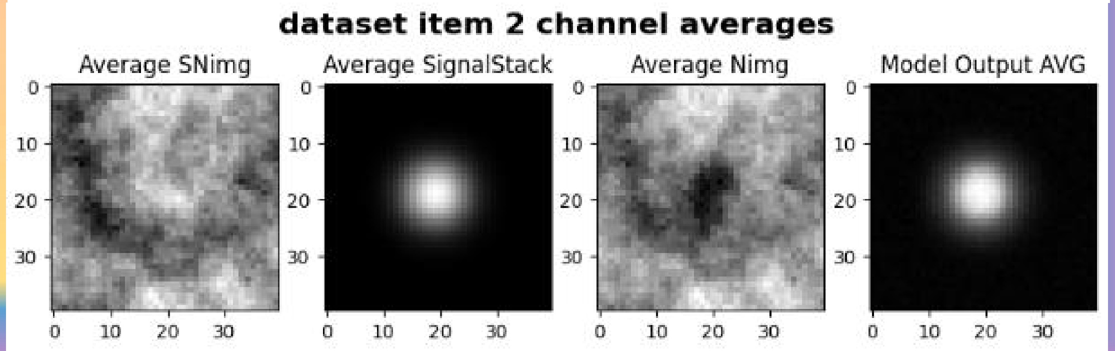
20 epoch, 50 batch, 400 sample, multiplicity=24, [40,40,1]











Current UI

- Jupyter notebook
  - run imports
  - run class/functional block
  - single function run
    - address param to root/gatm,datm OR root/raw,css
    - hyperparams and user-end variables
    - converts data, organizes subdirectories, creates dataloaders & model, sets params, runs model, saves end-case model & plots
  - handful of examples
  - ‘what next’ write-up @ end

what next? (more info in jupyter notebook)

- model architecture (currently simpleViT)
- modify 'multiplicity' parameter - analog for CSS stacking
- hyperparameter optimization such as by grid search over order of magnitude ranges, or more sophisticated methods
- insert model validation within training, couple lines of code in data prep & training
- modify initial data transform - images are transformed when loading in, many transforms available to test through
- Data
    - use on real data, best model for task will be one trained as close to real conditions as possible
    - modify test dataset from MATLAB fake_igrams code
- modify noise level to test robustness to variable noise
    - test dataset or during noise resampling step
- compartmentalize code further
- hardware changes to help with training on high resolution or high multiplicity
- Visualization - utilize existing plotting tools or others
- Train on different task? classification or another transform
- Generative adversarial network - one to mimic APS, other to determine underlying signal
- ?

References

- https://sioviz.ucsd.edu/~fialko/
  - data code & guidance
- https://github.com/lucidrains/vit-pytorch
  - examples of ViT models
- https://github.com/LukeFairbanks/APS_removal_via_Pytorch_ViT
  - final model, results
- https://www.geostockgroup.com/en/interferometric-synthetic-aperture-radar-insar-technology/
- https://twitter.com/VOLCAPSE/status/1562501704133750784