Deep Learning Crash Course

Hui Xue

Fall 2021

www.deeplearningcrashcourse.org

Outline

- Set up the training, cont.
- Data pre-processing
- Model weight initialization
- Tips for model training

Model error and training setup

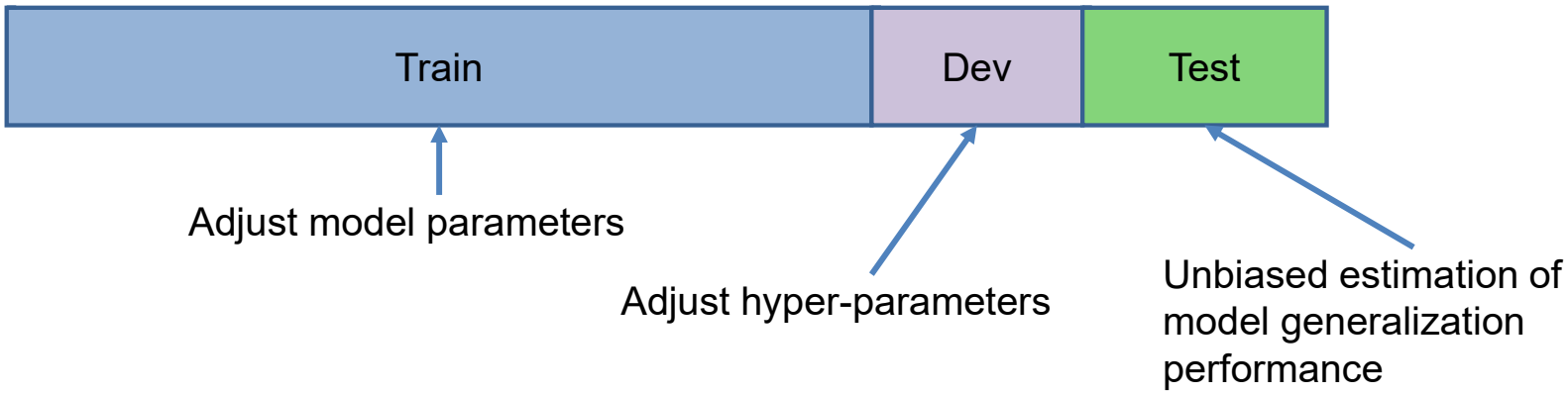
Given a data set, model error is:

$$E_{\boldsymbol{D}}\left[\left(y - M(\boldsymbol{x};\boldsymbol{D})\right)^2\right] = \{E_{\boldsymbol{D}}[M(\boldsymbol{x};\boldsymbol{D})] - f(\boldsymbol{x})\}^2 + E_{\boldsymbol{D}}\left[\left(E_{\boldsymbol{D}}\left(M(\boldsymbol{x};\boldsymbol{D})\right) - M(\boldsymbol{x};\boldsymbol{D})\right)^2\right] + \boldsymbol{\sigma}^2$$

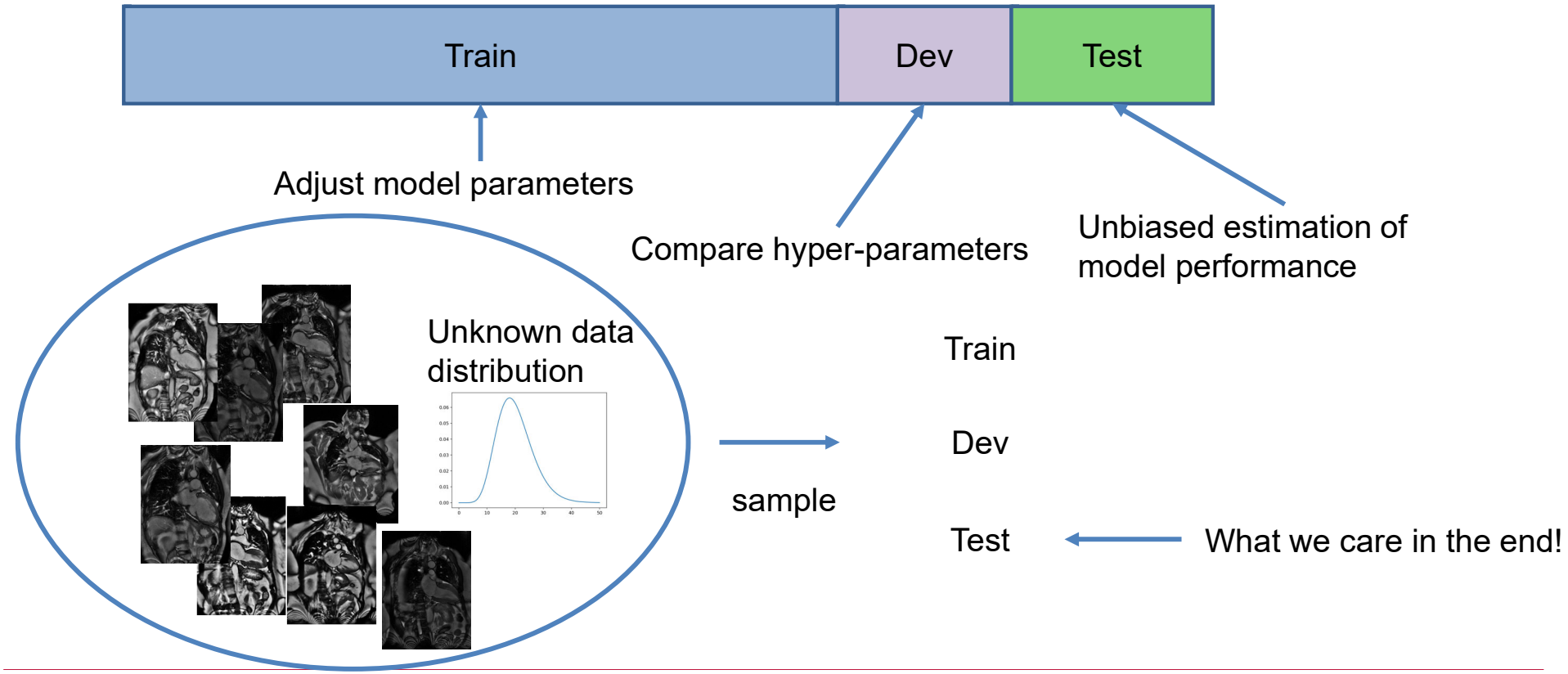
Bias

Variance

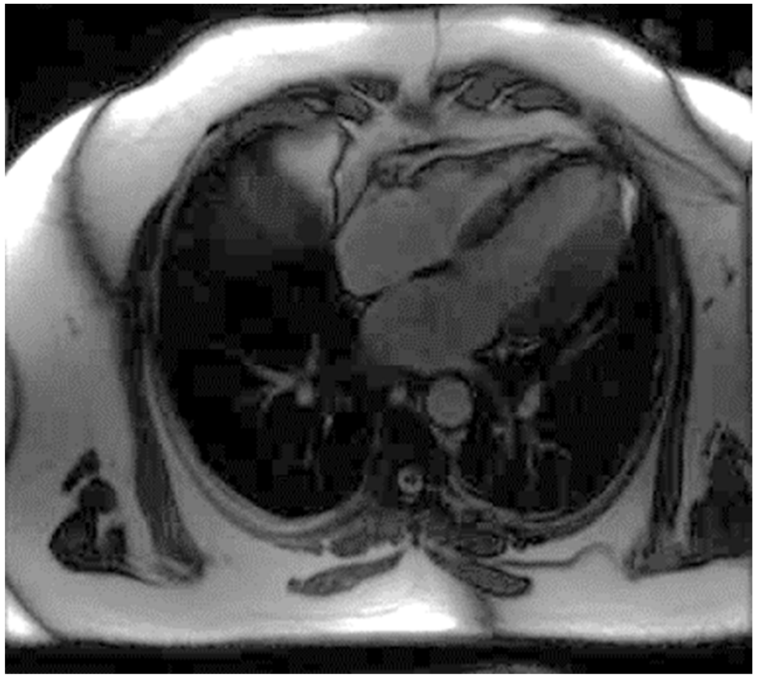
Bayes error



Ideally, Train, Dev, Test sets are from the same data distribution



Test and Train sets can be from different distribution

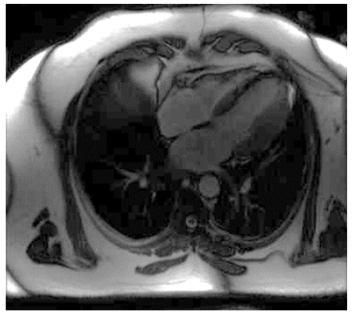


10K, MRI images

https://www.clinicalradiologyonline.net/article/S0009-9260(17)30058-2/fulltext

1K, CT images

Test and Train sets can be from different distribution



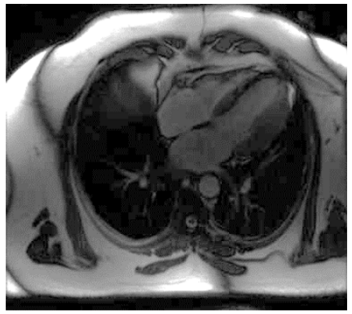
10K, MRI images

1K, CT images

Goal: Detect whether heart is imaged in MR images

	Option1	Option2	Option3	Option4
Train	8K MRI	8K MRI+1K CT	8K MRI	8K MRI
Dev	1K MRI	1K MRI	1K MRI+1K CT	1K MRI
Test	1K MRI	1K MRI	1K MRI	1K MRI +1K CT

Test and Train sets can be from different distribution



10K, MRI images

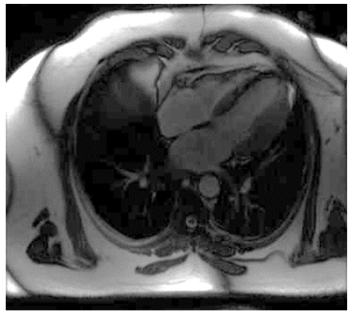
1K, CT images

Sample the Dev and Test sets from the same data distribution for the deployment time

Dev and Test sets define the target of the model

https://www.google.com/url?sa=i&url=https%3A%2F%2Ftowardsdatascience.com%2Ftradeoff-bias-or-variance-1409eec38caf&psig=AOvVaw0FzJ-R_8JxUkPlAiJQax7Y&ust=1620153718709000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCMCroImVrvACFQAAAAAdAAAAABAD

Test and Train sets can be from different distribution



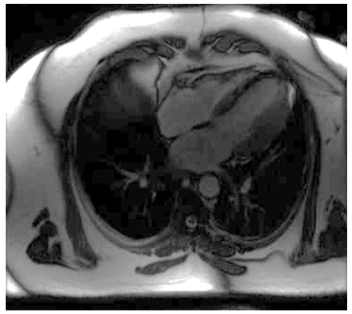
10K, MRI images

1K, CT images

Goal: Detect whether heart is imaged in **CT** images

	Option1	Option2	Option3
Train	500 CT	10K MRI	10K MRI+500 CT
Dev	250 CT	500 CT	250 CT
Test	250 CT	500 CT	250 CT

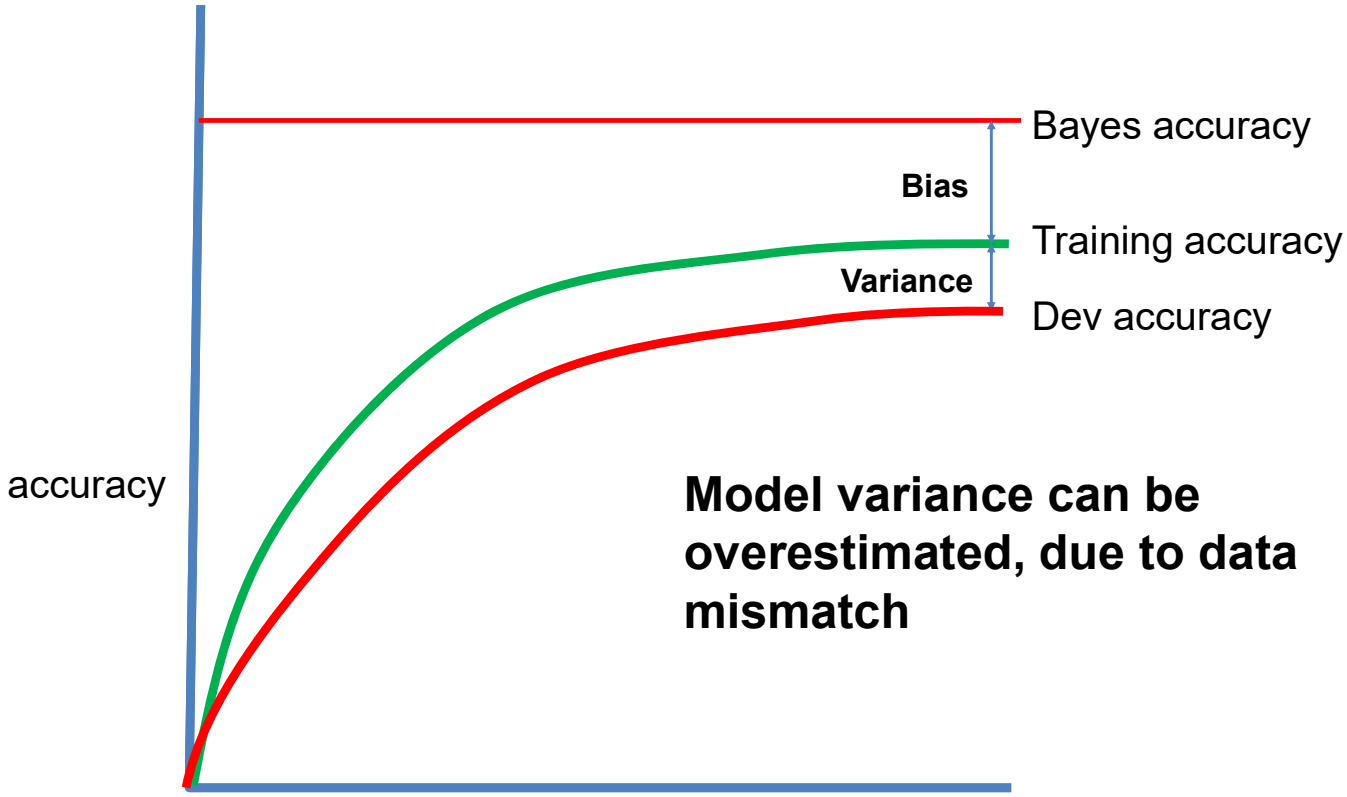
Dev and Train set are from the different distribution



10K, MRI images

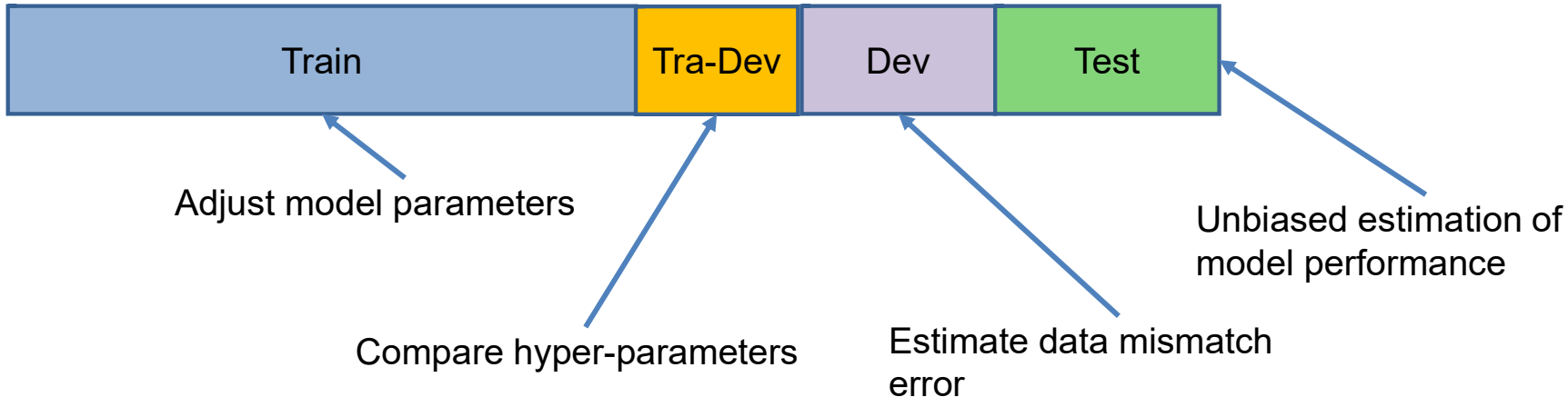
1K, CT images

Goal: Detect whether heart is imaged in **CT** images



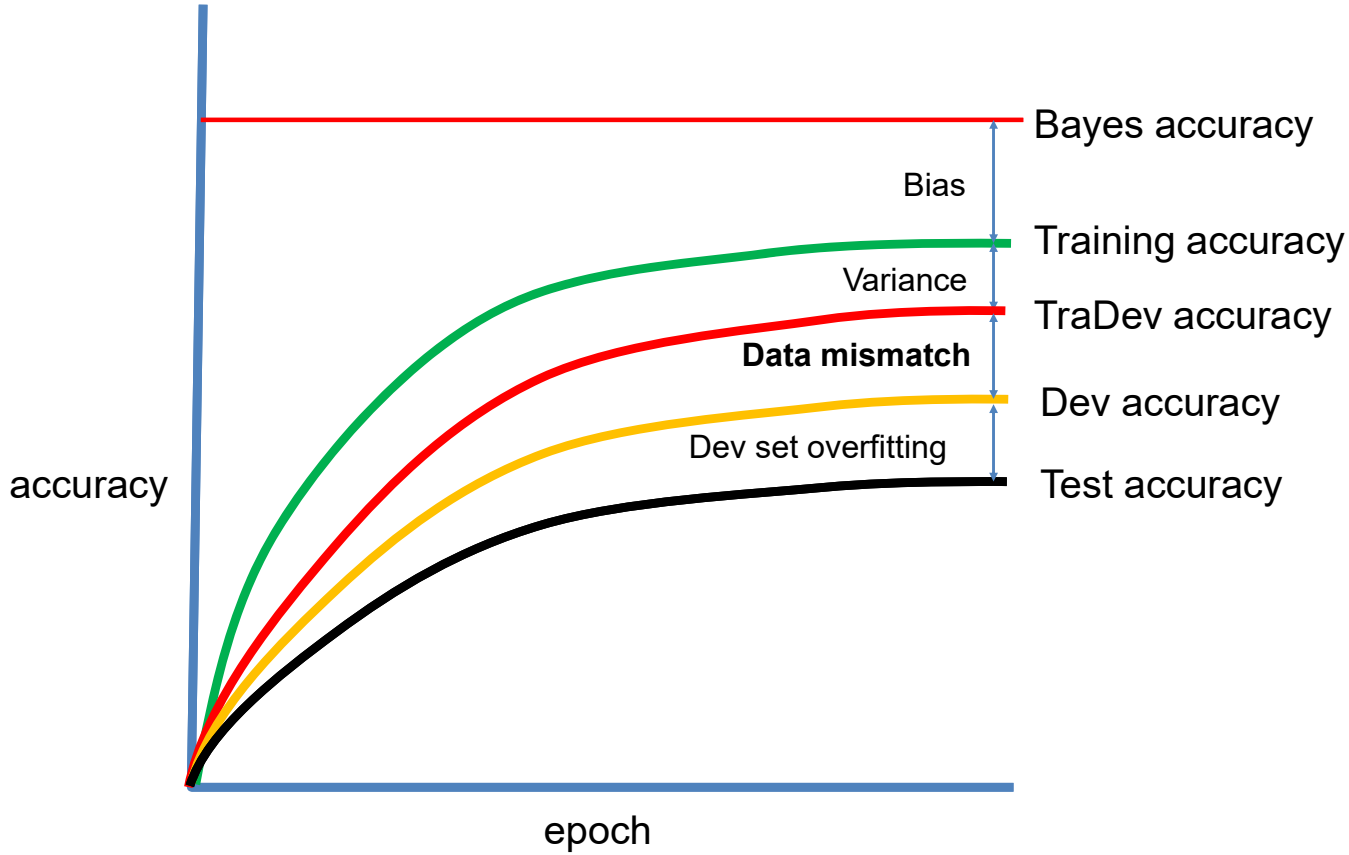
Model variance can be overestimated, due to data mismatch

Dev and Train set are from the different distribution



- Split the train set further to get a Tra-Dev set
- Train and Tra-Dev set have the same data distribution
- Used to estimate model variance

Train, Tra-Dev, Dev and Test sets



Spot problems : Error analysis

Error analysis : exam the incorrect/less-perfect samples in Dev set to spot the problems

Typical sample
in Train set

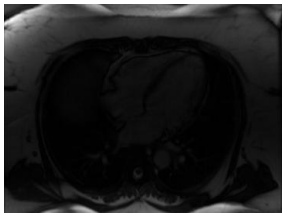
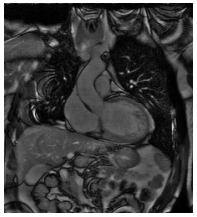


Image is too dark 5%



New image view 11%

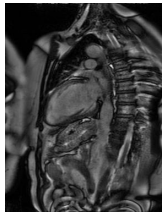
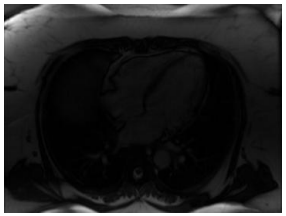
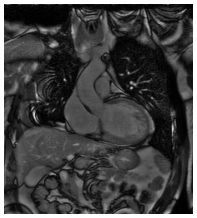


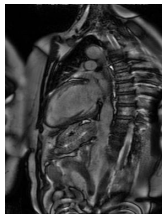
Image is flipped and with different contrast 14%

Spot problems : Error analysis

Error analysis : exam the incorrect/less-perfect samples in Dev set to spot the problems

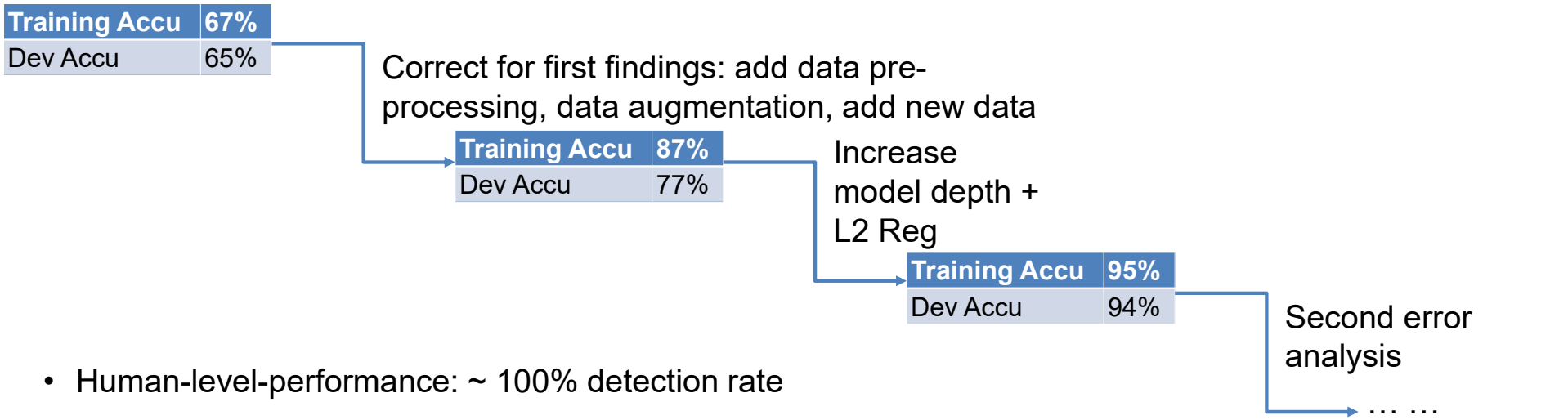






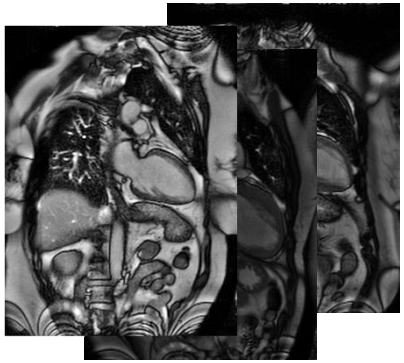
Findings	Error rate	Remedies
Image is too dark	5%	Add pre-processing
New image view	11%	Acquire more data
Image is flipped and with different contrast	14%	Add data augmentation

Iterate error analysis with training

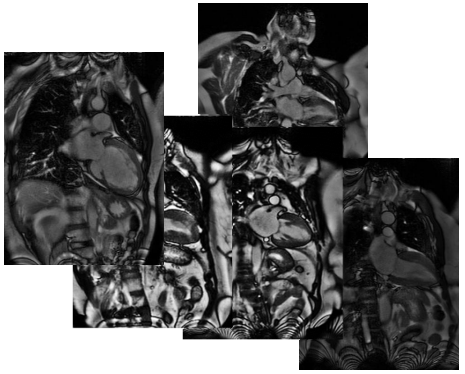


- Human-level-performance: ~ 100% detection rate
- Iterative process
- The key for deep learning is to decide what to do at each step

Speedup the error analysis

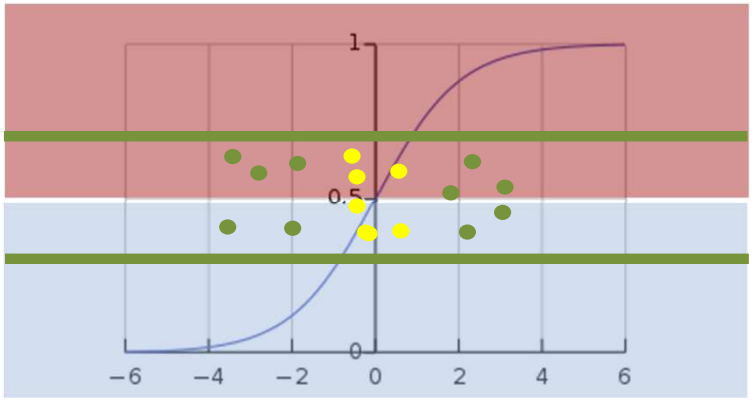


Eyeball set: a portion of dev set to check



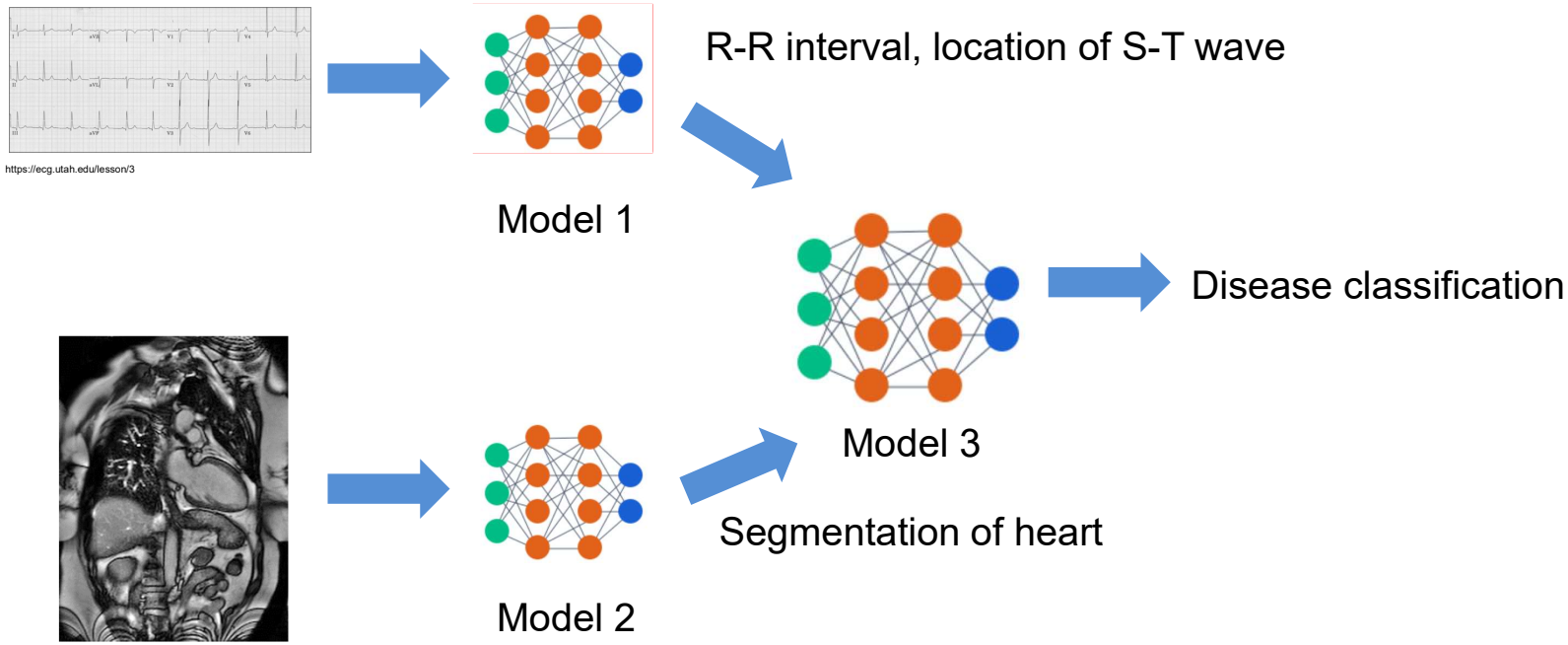
Blackbox set: a portion of dev set to apply model, but not exam for error

Active selection : use uncertainty

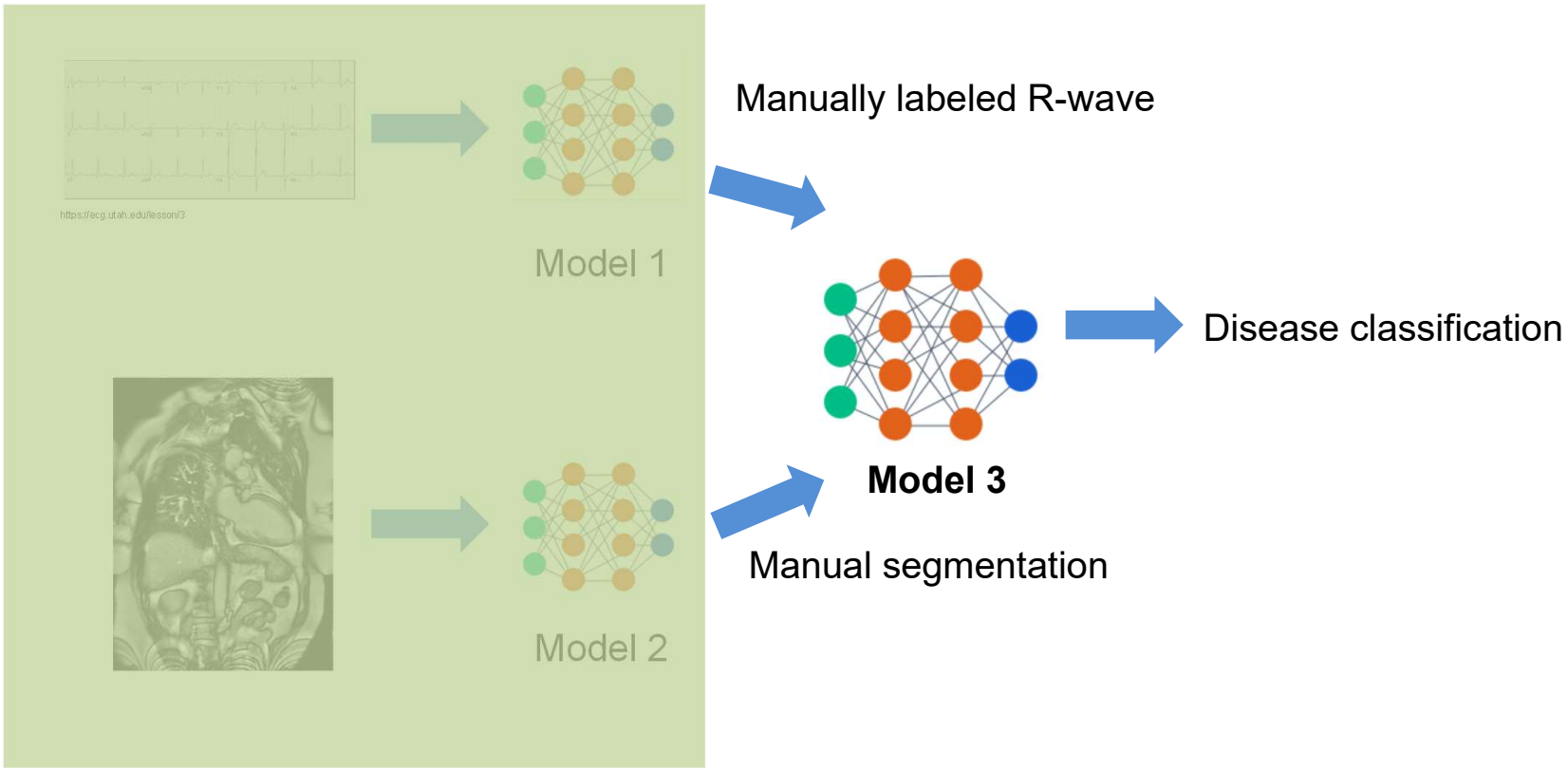


- Use the model output to pick samples to check
- Consider to use different metrics
- Combine domain knowledge, e.g. signal-noise-ratio in different imaging protocols, image appearances in different diseases (such as hypertrophic cardiomyopathy)
- Automate this process and optimize tools

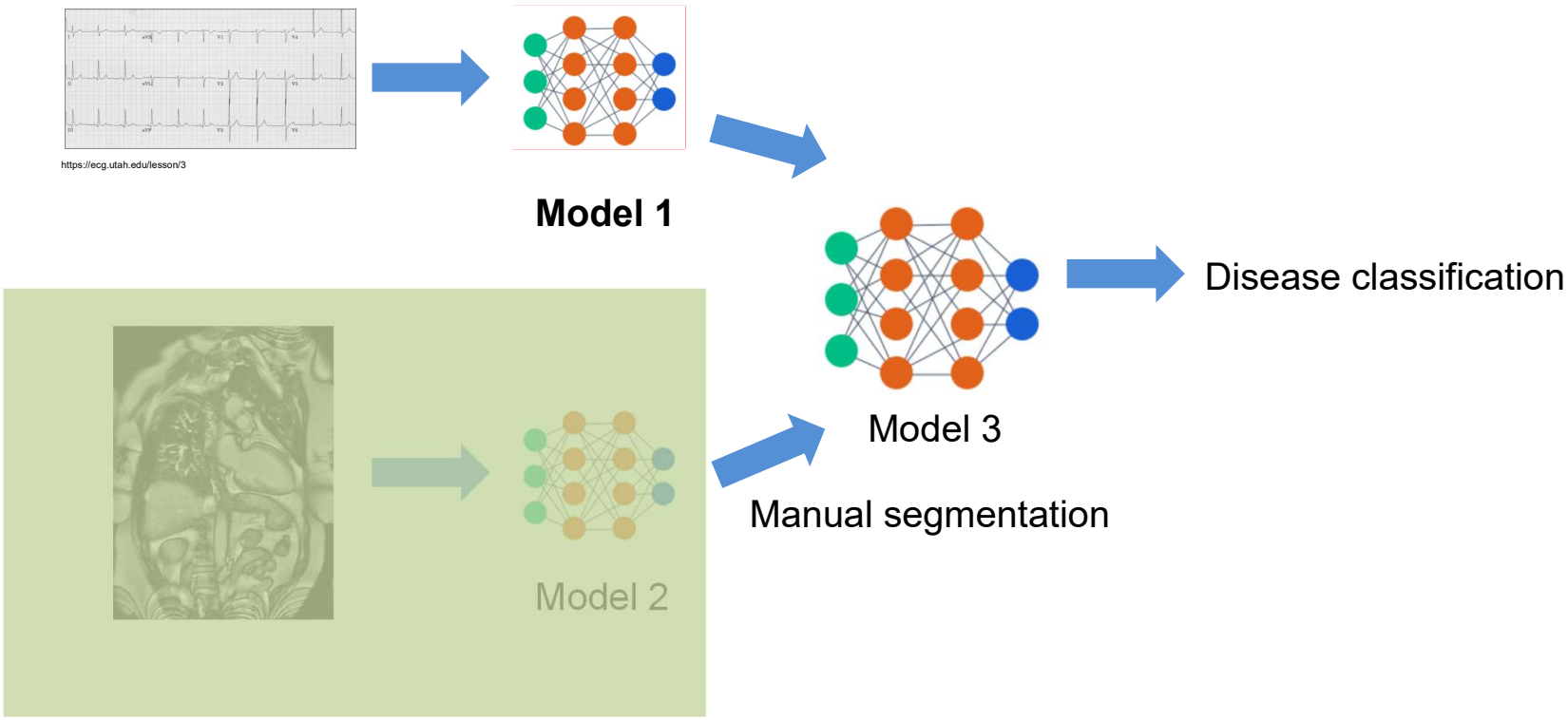
Error analysis of a complex system



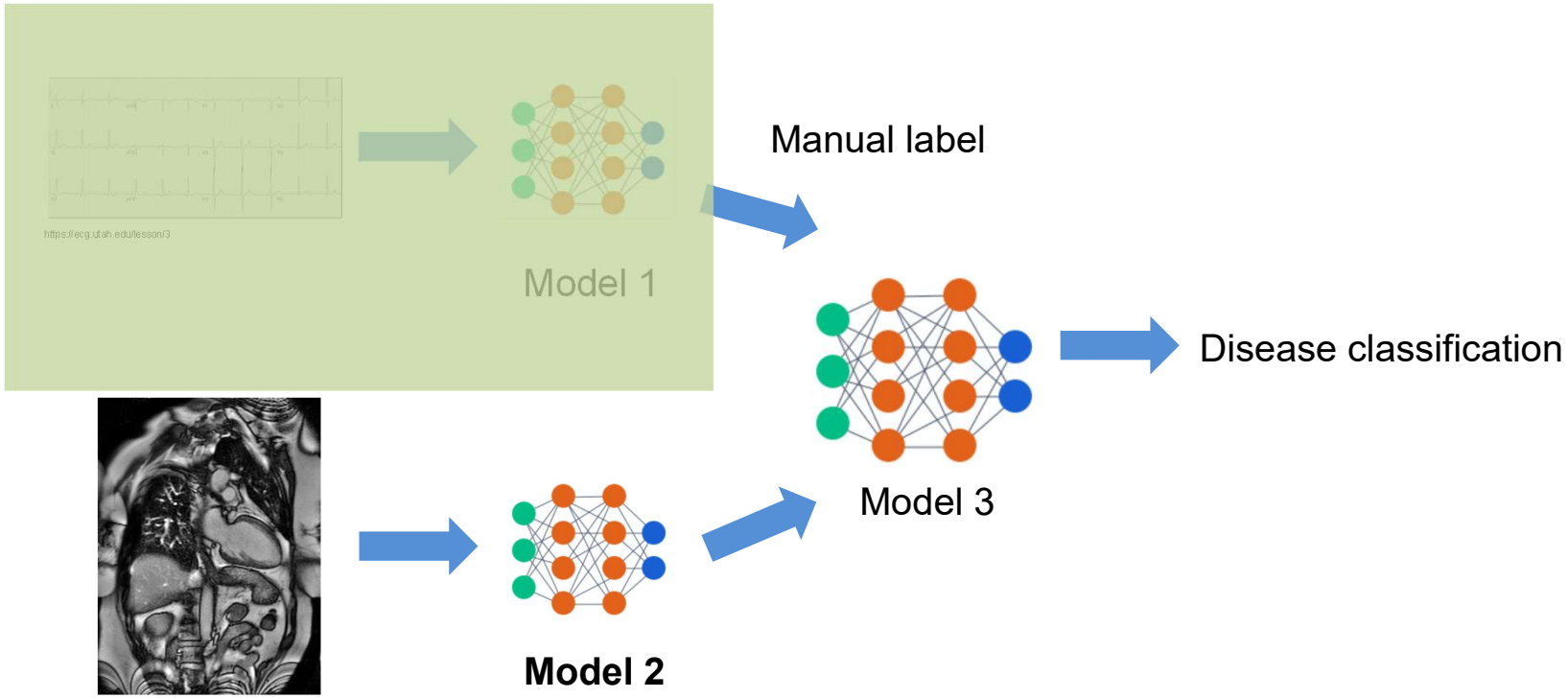
Error analysis of a complex system



Error analysis of a complex system

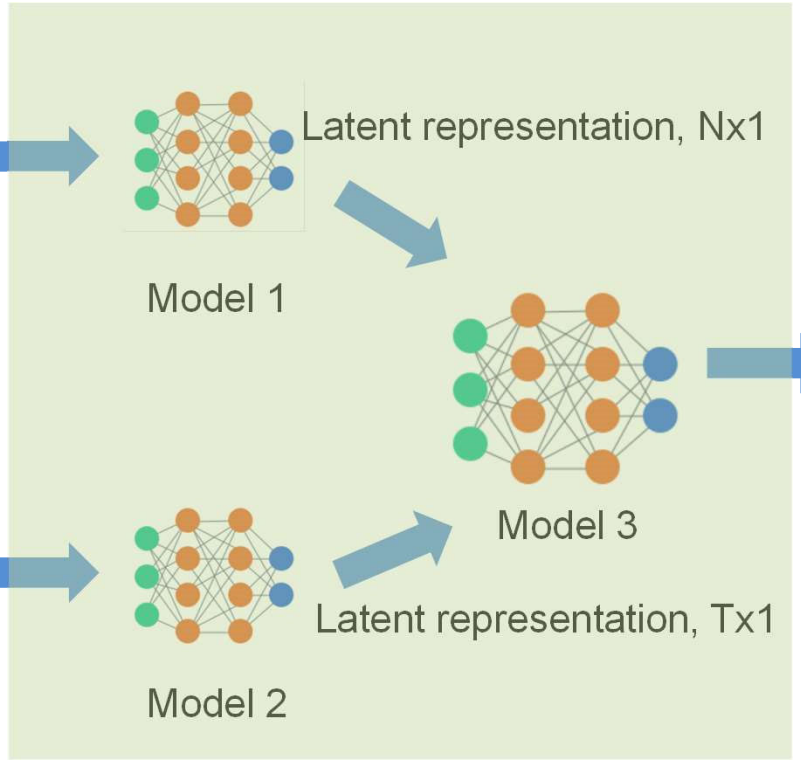


Error analysis of a complex system



Error analysis of a complex system

https://ecg.utah.edu/lesson/3



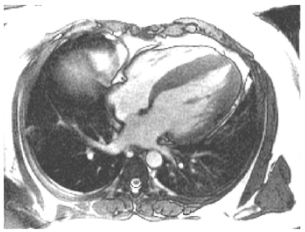
- Debug as one big model
- End-to-end training
- Harder to isolate bugs

Outline

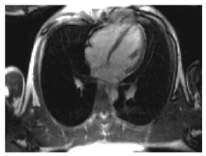
- Set up the training, cont.
- **Data pre-processing**
- Model weight initialization
- Tips for model training

Data pre-processing to reduce variation

Standardizing the input data and reducing the variation can significantly help training

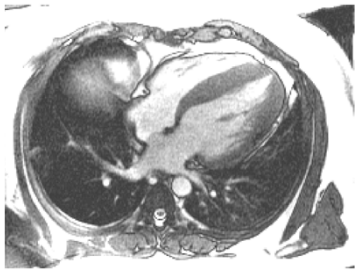


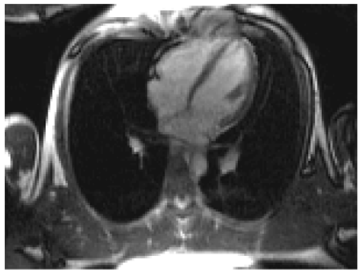
High spatial resolution



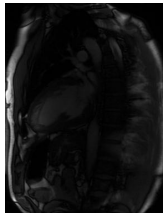
Low spatial resolution

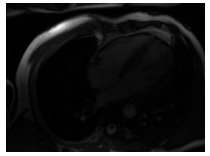
Resample to fixed resolution



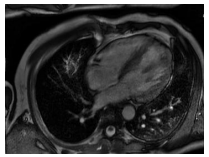


Fixed high resolution of 1mm^2





Adjust image windowing

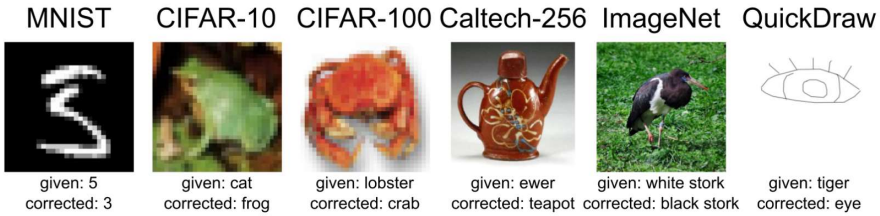


Data pre-processing to understand

Understand the “edge” samples, exclude the corrupted samples and correct wrong labels

Noisy with artifacts

Signal loss due to devices



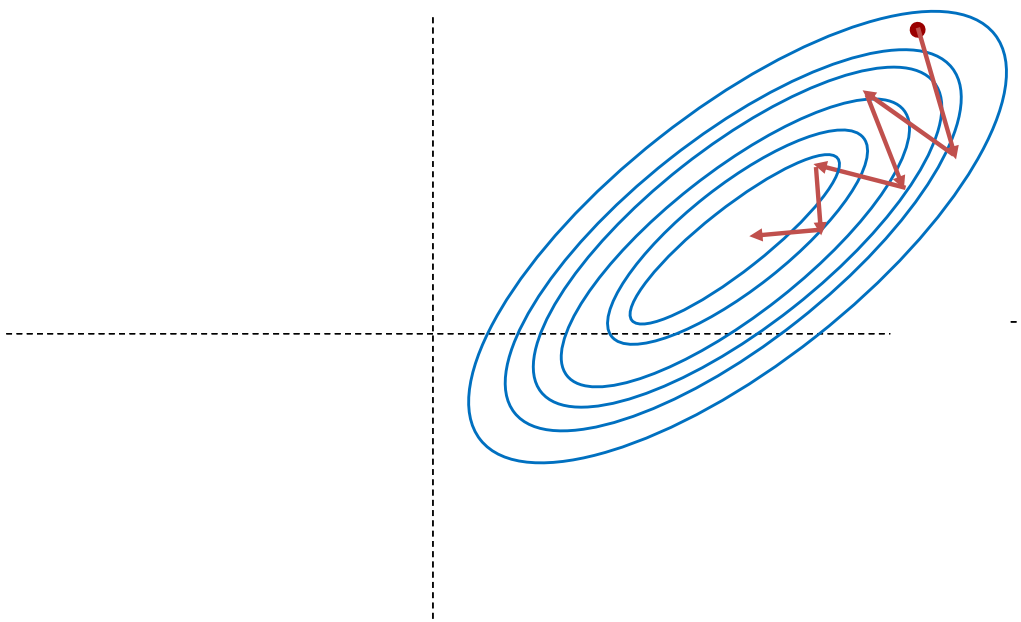
An average of 3.4% errors across the 10 benchmark ML datasets

Has impact on evaluating different models

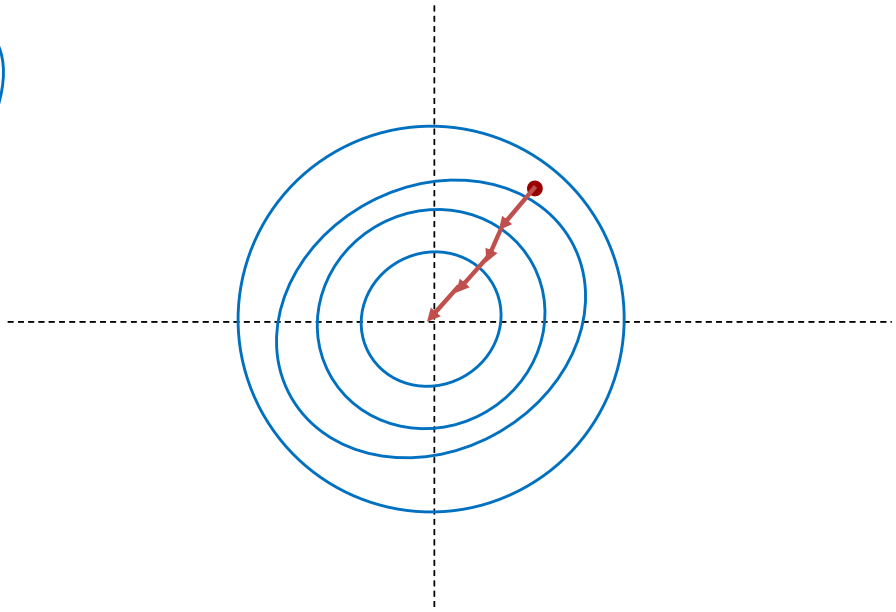
https://labelerrors.com/
https://l7.curtisnorthcutt.com/confident-learning

Data pre-processing

Normalize different dimension of input data, help optimization



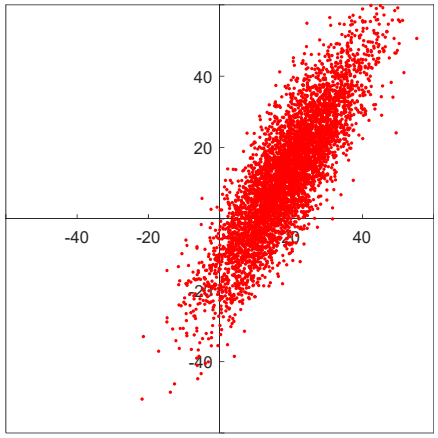
Features with varying magnitude



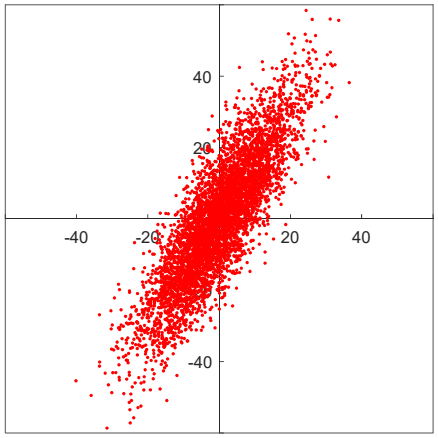
After scaling and centering

Scale and centering

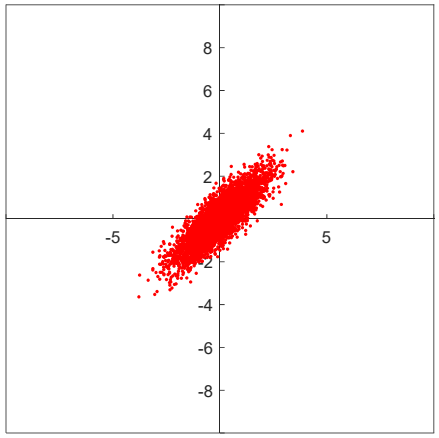
Normalize different dimension of input data, help optimization



$X \in \mathbb{R}^{N\times 2}$



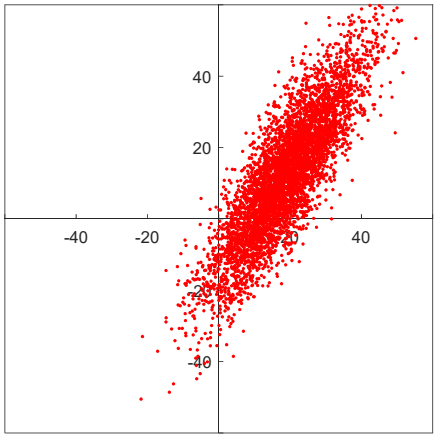
$X = X - np.mean(X, axis = 0)$



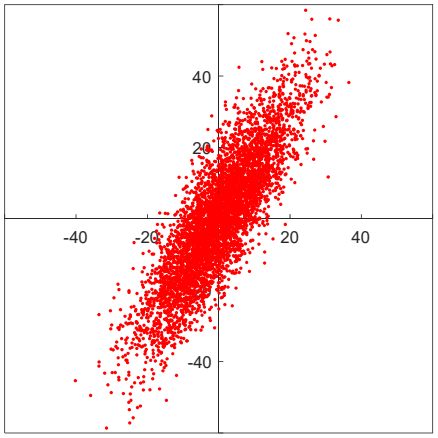
$$X = \frac{X - np.mean(X, axis = 0)}{np.std(X, axis = 0)}$$

Another approach often used: $X = \frac{X - np.mean(X, axis=0)}{np.amax(X, axis=0) - np.amin(X, axis=0)}$

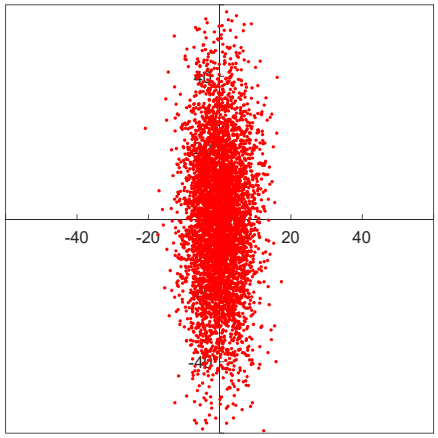
PCA: Principal Component Analysis



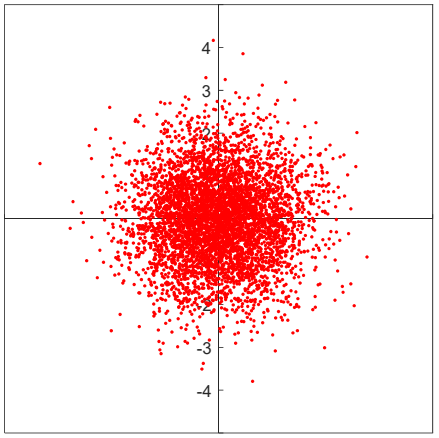
$X \in \mathbb{R}^{N \times 2}$



$X = X - np.mean(X, axis = 0)$



$[E, V] = np.linalg.eig(np.dot(X.T, X)/N)$
$X = np.dot(X, V)$



$X = X/np.sqrt(X \cdot diagonal())$

PCA: Principal Component Analysis

$X \in \mathbb{R}^{N \times D}$ If the input dimension D is large, dimension reduction can be performed

$$[E, V] = np.linalg.eig(np.dot(X.T, X)/N)$$
$$X = np.dot(X, V)$$
$$X = X[:, 0:K]$$

Only keep the top K dimension, $X \in \mathbb{R}^{N \times K}$, can filter out some noise

These steps are also called “Whitening” :

the covariance matrix of X after applying the eigenvector and divided by sqrt(eigenvalue) becomes identity matrix

Tips for data processing

- For image, often only apply centering and scaling

$$X = \frac{X - np.mean(X, axis = 0)}{np.amax(X, axis = 0) - np.amin(X, axis = 0)}$$

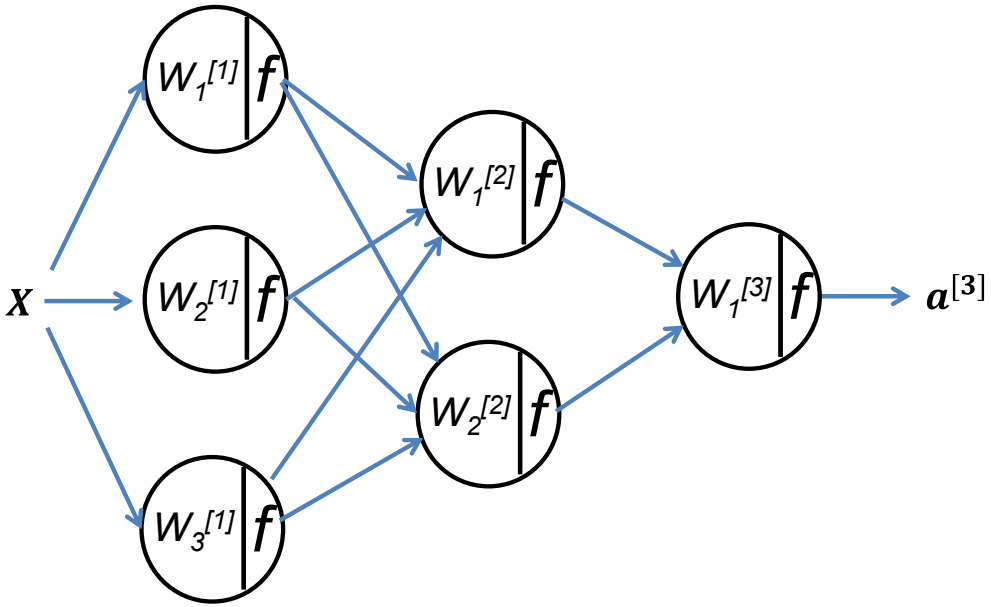
Or simply, X=X/np.max(X) ← scale image to be [0, 1]

- For time signal or multi-channel signals, PCA can be useful
- Start simple and plot results from every preprocessing steps
- Including the **identical** preprocessing in the regression test
- Make sure the inference data go through the **identical** pre-processing steps
- Computing the pre-processing statistics (e.g. dataset mean, eigen vectors, eigen values …) on the training set **ONLY** and applying them in inference
- Complicated pre-processing can make model more vulnerable to data distribution shift

Outline

- Set up the training, cont.
- Data pre-processing
- **Model weight initialization**
- Tips for model training

Need a strategy to give weights some initial values



$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$a^{[1]} = f(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = f(Z^{[2]})$$
$$Z^{[3]} = W^{[3]}a^{[2]} + b^{[3]}$$
$$\hat{y} = a^{[3]} = f(Z^{[3]})$$

How do we initialize weights and biases?

Need a strategy to give weights some initial values

How about W=0 and b=0?

$$\boldsymbol{Z}^{[1]} = \boldsymbol{W}^{[1]}\boldsymbol{X} + \boldsymbol{b}^{[1]}$$
$$\boldsymbol{a}^{[1]} = \boldsymbol{f}(\boldsymbol{Z}^{[1]})$$
$$\boldsymbol{Z}^{[2]} = \boldsymbol{W}^{[2]}\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}$$
$$\boldsymbol{a}^{[2]} = \boldsymbol{f}(\boldsymbol{Z}^{[2]})$$
$$\boldsymbol{Z}^{[3]} = \boldsymbol{W}^{[3]}\boldsymbol{a}^{[2]} + \boldsymbol{b}^{[3]}$$
$$\hat{\boldsymbol{y}} = \boldsymbol{a}^{[3]} = \boldsymbol{f}(\boldsymbol{Z}^{[3]})$$

All $\boldsymbol{Z}^{[l]}$ will be zero

All $\boldsymbol{a}^{[l]}$ will be a constant, depending on the non-linear activation function used

When performing backprop, then gradient will be zero …

So SGD cannot update network parameters …

Need a strategy to give weights some initial values

How about W=σ ×np.random.rand(input_D, output_D), zero-mean, gaussian random values

$$\boldsymbol{Z^{[1]} = W^{[1]}X + b^{[1]}}$$

$\boldsymbol{W^{[1]}}$:3xN matrix

$$Z_k^{[1]} = \sum_{i=0}^{N-1} W_{k,i}^{[1]} x_i + \boldsymbol{b^{[1]}}$$ The k-th element of score Z

Let's whitening the input X, so it has same variance for all features.

$$\boldsymbol{Var}(Z_k^{[1]}) = \sum_{i=0}^{N-1} \boldsymbol{Var}((W_{k,i}^{[1]})\boldsymbol{Var}(x_i) + \boldsymbol{Var}(b^{[1]}) = N\sigma^2\ \boldsymbol{Var}(x)$$

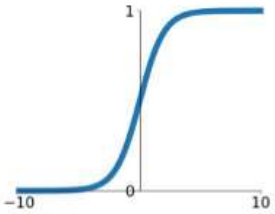
Since all x_i and $W_{k,i}^{[1]}$ are independent upon initialization, with zero mean

$$\boldsymbol{Var}(wx) = \boldsymbol{E}(w^2)\boldsymbol{E}(x^2) - \boldsymbol{E}(w)^2\ \boldsymbol{E}(x)^2 = \boldsymbol{E}(w^2)\boldsymbol{E}(x^2) = \boldsymbol{Var}(w)\boldsymbol{Var}(x)$$ If w and x are independent and have zero mean

For deep network, activation function can be saturated

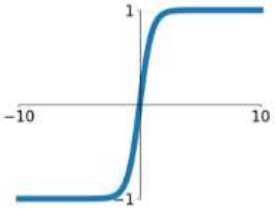
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



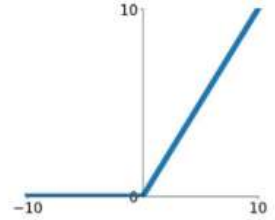
tanh

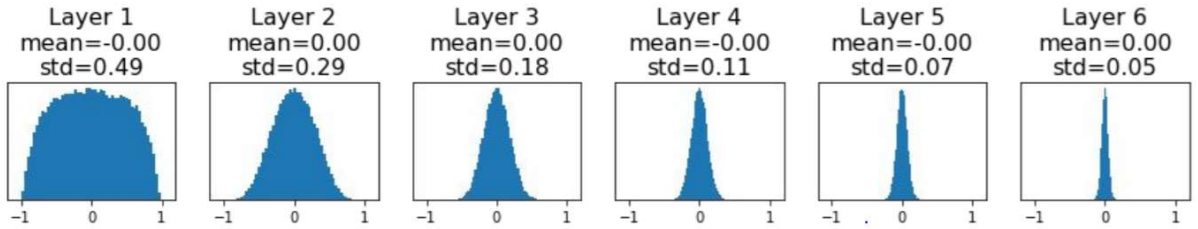
$$\tanh(x)$$



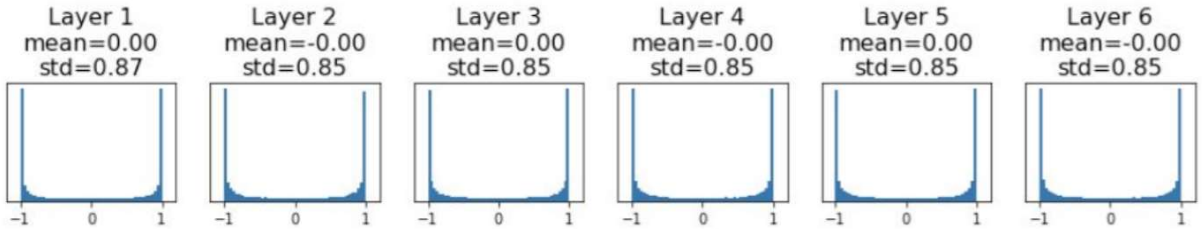
ReLU

$$\max(0, x)$$





σ=0.01



σ=0.05

tanh activation was used

Figure credit: Stanford CS231N notes

Xavier Initialization

$$\boldsymbol{Var}(\boldsymbol{Z_k^{[1]}}) = \boldsymbol{N}\sigma^2\ \boldsymbol{Var}(x)$$

For a deep network, the non-linear activation can be saturated for native initialization

- Make the learning harder
- Sensitive to hyperparameter configuration

To keep the output variance unchanged:

W=np.random.rand(output_D, input_D)/np.sqrt(input_D) $\sigma = \frac{1}{\sqrt{N}}$

$$\boldsymbol{Var}(\boldsymbol{Z_k^{[1]}}) = \sum_{\boldsymbol{i=0}}^{\boldsymbol{N-1}} \boldsymbol{Var}((\frac{\boldsymbol{1}}{\sqrt{\boldsymbol{N}}} W_{\boldsymbol{k,i}}^{[\boldsymbol{1}]})\boldsymbol{Var}(x_{\boldsymbol{i}}) + \boldsymbol{Var}(\boldsymbol{b}^{[\boldsymbol{1}]}) = \boldsymbol{Var}(x)$$

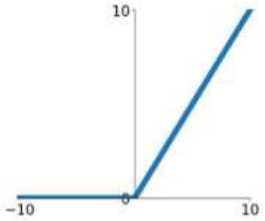
Xavier Glorot and Yoshua Bengio, “Understanding the difficulty of training deep feedforward neural networks”, AISTAT 2010

He initialization: Xavier Initialization for ReLU activation

W=np.random.rand(output_D, input_D) × **2**/np.sqrt(input_D)

ReLU

$\max(0, x)$



$$\sigma = \frac{2}{\sqrt{N}}$$

To compensate the ReLU function for the negative inputs

$$\boldsymbol{Var}(wx) = \boldsymbol{E}(w^2)\boldsymbol{E}(x^2) - \boldsymbol{E}(w)^2\ \boldsymbol{E}(x)^2 = \boldsymbol{E}(w^2)\boldsymbol{E}(\boldsymbol{x}^2) = \boldsymbol{Var}(w)\boldsymbol{E}(\boldsymbol{x}^2) = \frac{1}{2}\boldsymbol{Var}(w)\boldsymbol{Var}(x)$$

x is the response after RELU, which has positive mean

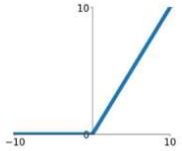
- Use Xavier or He initialization
- Bias is initialized to zero; randomness in weights often is sufficient to drive the training
- With Batch Normalization, network training is much less sensitive to parameter initialization

He et al, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”, ICCV 2015

He initialization: Xavier Initialization for ReLU activation

ReLU

$\max(0, x)$



y is the score before ReLU:

x = max(0, y)

y has zero-mean and symmetric distribution,
if the network is in good condition before this layer

$$\boldsymbol{Var}(wx) = \boldsymbol{E}(w^2)\boldsymbol{E}(x^2) - \boldsymbol{E}(w)^2\,\boldsymbol{E}(x)^2 = \boldsymbol{E}(w^2)\boldsymbol{E}(\boldsymbol{x^2}) = \boldsymbol{Var}(w)\boldsymbol{E}(\boldsymbol{x^2}) = \frac{1}{2}\boldsymbol{Var}(w)\boldsymbol{Var}(x)$$

x is the activation after RELU, which has positive mean

$$E[x^2] = \int_{-\infty}^{+\infty} \max(0, y)^2 p(y)dy$$

where the part $y < 0$ does not contribute to the Integral

$$= \int_{0}^{+\infty} y^2 p(y)dy$$

which we can write as half the integral over the entire real domain (y^2 is symmetric around 0 and $p(y)$ is assumed to be symmetric around 0):

$$= \frac{1}{2}\int_{-\infty}^{+\infty} y^2 p(y)dy$$

now subtracting zero in the square we get:

$$= \frac{1}{2}\int_{-\infty}^{+\infty} (y - E[y])^2 p(y)dy$$

which is

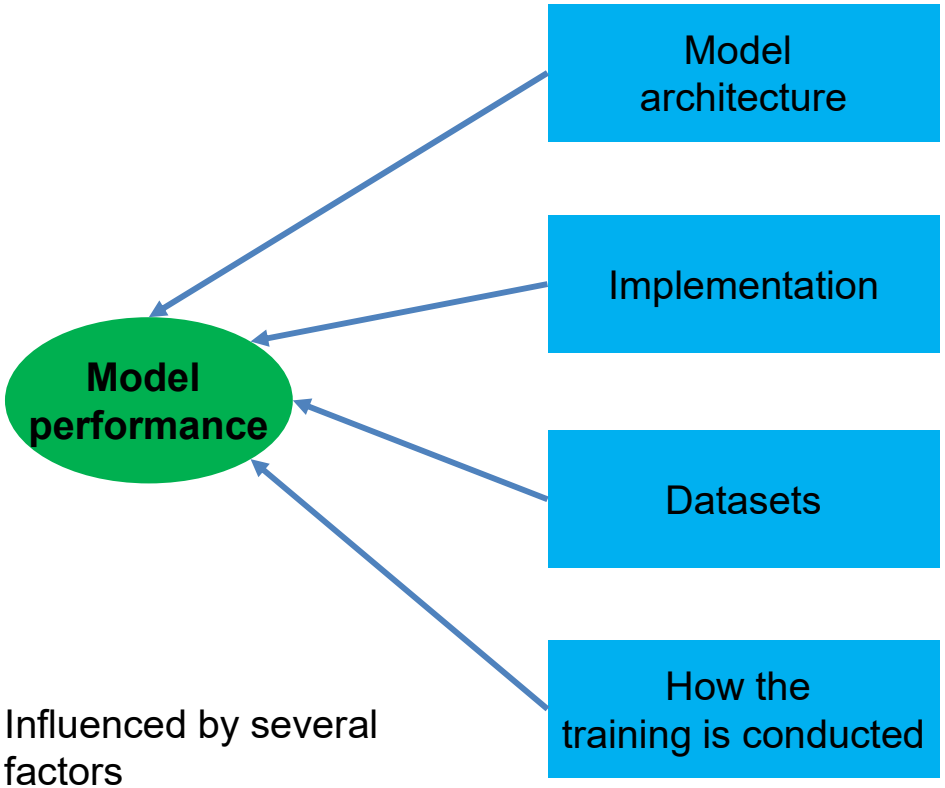
$$= \frac{1}{2}E[(y - E[y])^2] = \frac{1}{2}Var[y]$$

https://stats.stackexchange.com/questions/138035/variance-calculation-relu-function-deep-learning

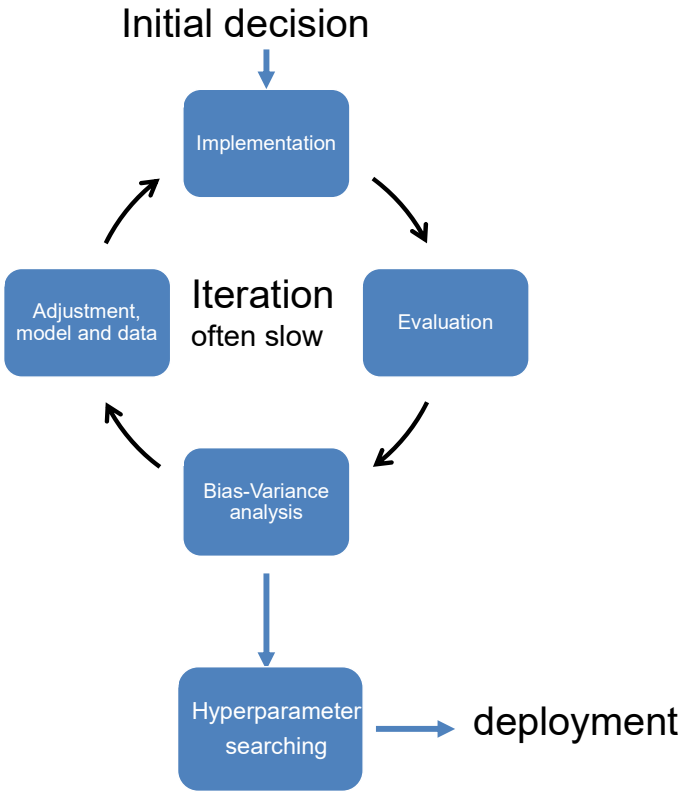
Outline

- Set up the training, cont.
- Data pre-processing
- Model weight initialization
- **Tips for model training**

Debug deep neural network model can be difficult

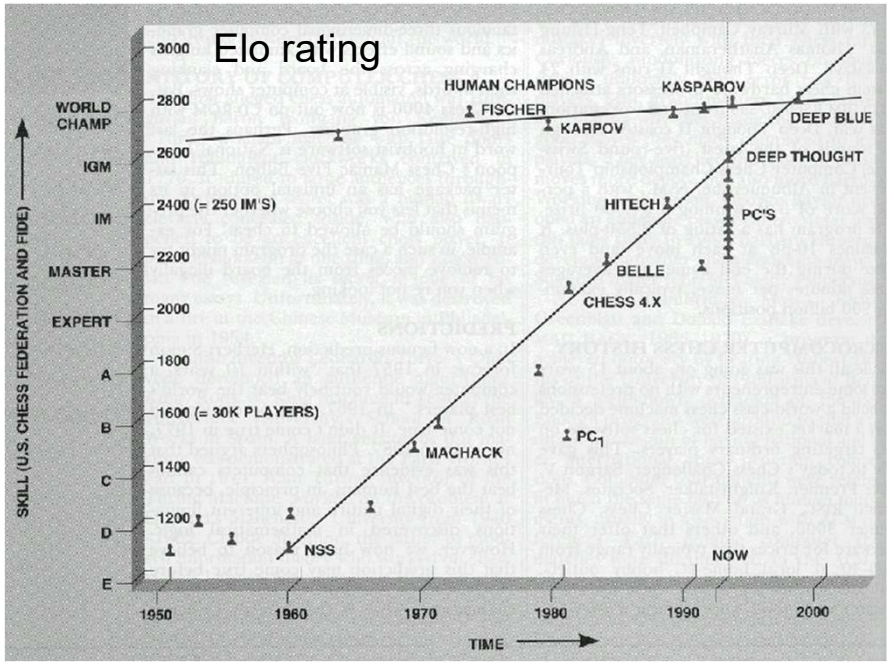


Influenced by several factors



Before you started …

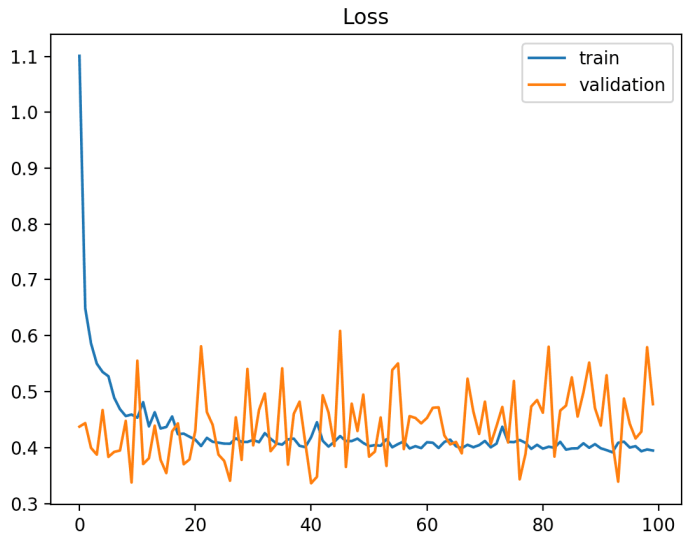
- Have an estimation of Bayes accuracy
  - Human level performance as a surrogate
  - Expert performance – experienced operators vs. less experienced
  - Voting strategy
  - Baseline and results from publications
  - Whether it has related competition



https://www.drdobbs.com/parallel/computer-chess-the-drosophila-of-ai/184405171

Before you started …

- Have the Dev and Test set established



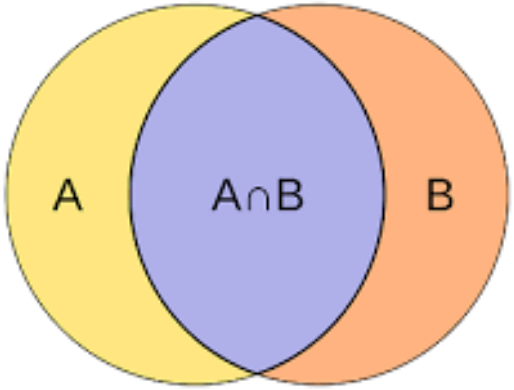
- Dev and test sets should be large enough to give a reliable estimation of model performance
- If the total amount of data is small, get more data and use cross-validation

https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

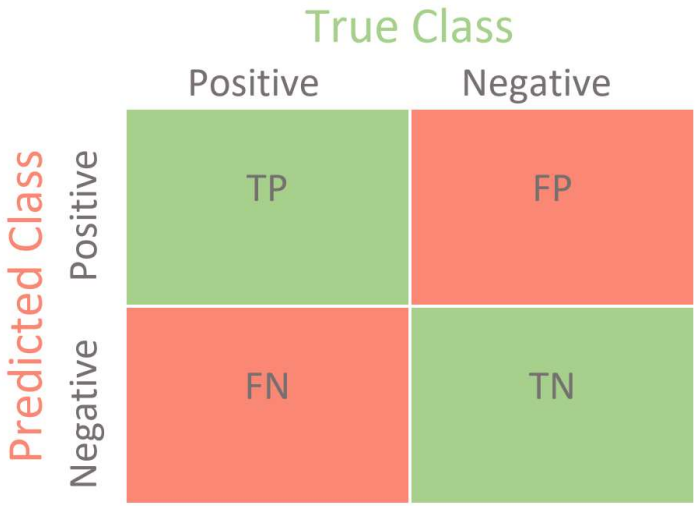
Before you started …

- Have the evaluation metric

  Accuracy, Dice ratio, L2 distance, business specific metrics?
  Pick the key metric



$$\text{Dice coefficient}(A, B) = \frac{2 \times |A \cap B|}{|A| + |B|}$$



Accuracy=(TP+TN)/(TP+TN+FP+FN)

Precision=TP/(TP+FP)

Sensitivity=TP/(TP+FN)

Specificity=TN/(TN+FP)

F1 score=2 x Pre x Sen / (Precision+Sensitivity)

https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826

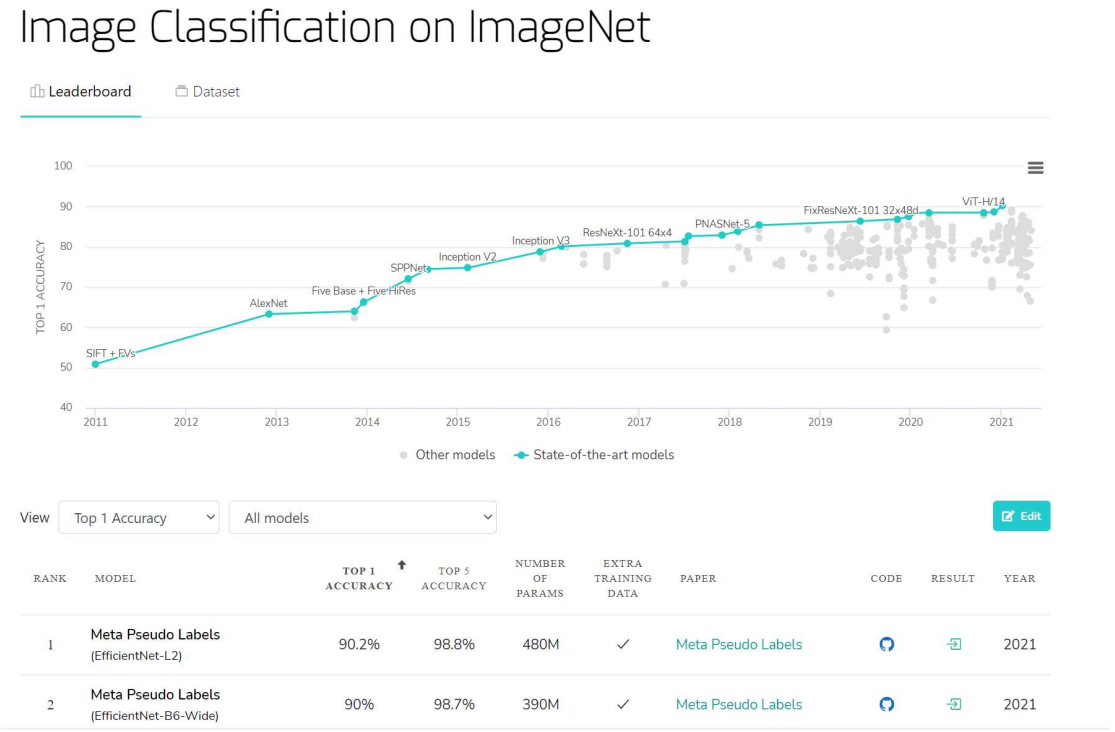
Before you started …

- Have the expectation
  - Know when to stop
  - Be the state-of-the-art?
  - Meet the business needs?

Consider the resource limitation

Data and computing

90% and 90.2% may be the same for your deployment



https://paperswithcode.com/sota/image-classification-on-imagenet

Model architecture

- Start from standard choices and make changes if needed
- Look for available resources
- Look for related publications
- Read the code if possible and do not take it blindly

Tasks	First try
Image classification	Standard ResNet, e.g. implementation in Pytorch repo
Segmentation	Unet architecture
Language, NLP	Pre-trained GPT family models
Time series signal	Bidirectional RNN with LSTM, transformer
Tabular data	Dense MLP, DropOut, BatchNorm

Implementation

- Pay attention to output layers

  With the same base network, different functionalities can be achieved by changing the output layers:
  - Pooling and dense linear for classification
  - CONV layer for segmentation
  - Dense linear for regression

- Pay attention to loss function – need logits or need probabilities?

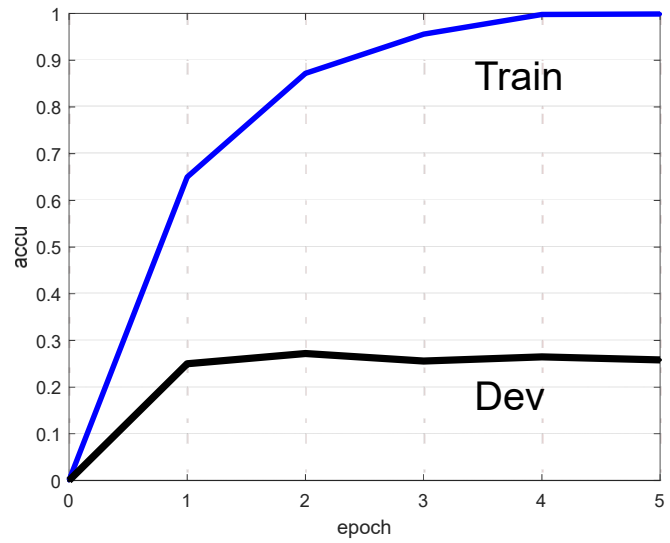
  different implementation may pack extra computation into loss function

```
>>> m = nn.Sigmoid()
>>> loss = nn.BCELoss()
>>> input = torch.randn(3, requires_grad=True)
>>> target = torch.empty(3).random_(2)
>>> output = loss(m(input), target)
```

```
>>> m = nn.LogSoftmax(dim=1)
>>> loss = nn.NLLLoss()
>>> # input is of size N x C = 3 x 5
>>> input = torch.randn(3, 5, requires_grad=True)
>>> # each element in target has to have 0 <= value < C
>>> target = torch.tensor([1, 0, 4])
>>> output = loss(m(input), target)
```

Overfit a small dataset, e.g. one mini-batch

- For classification, accuracy should quickly reach 100%
- For segmentation, Dice ratio should quickly increase to ~1.0



Very useful to find implementation bugs:

Loss goes up

learning rate too high, error in output layer for class order…

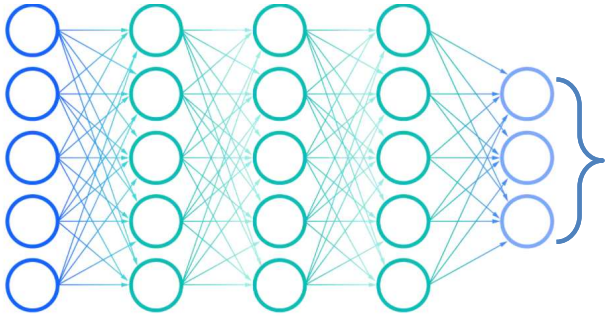
Loss is NaN

Overflow or underflow or divided-by-zero

Accuracy plateaus

learning rate too low, bad initialization, regularization too high, data/label mismatch …

Check the initial loss function, if making sense



These nodes will give roughly same scores for every class

- Random initialized
- All output classes are treated equally

- For the binary classification, initial loss will be close to –log(0.5)
- For the multi-class classification, initial loss will be close to –log(1/C), C is the number of total class

First turn off regularization/drop out/Batch Norm

$$L = \frac{1}{B}\sum_{i=0}^{B-1} L^{(i)} + \frac{\lambda}{2}\|\boldsymbol{W}\|_2^2$$

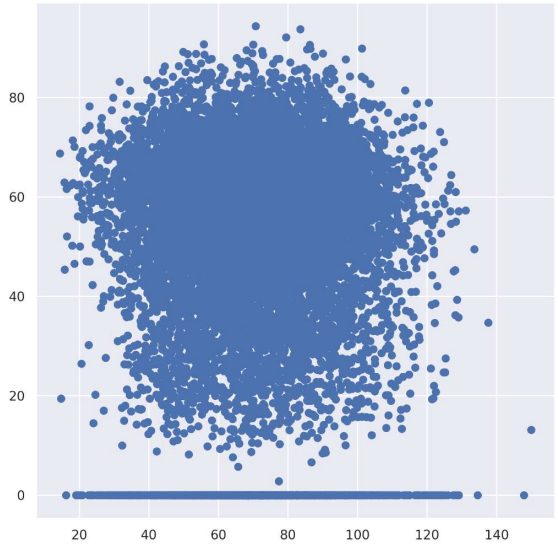
- Set $\lambda = 0$ to begin with
- Maker sure regularization loss not dominate total loss
- Turn off operations to introduce randomness to training, e.g. drop out
  Add controls to allow turn them on/off easier
- Consider to turn off batch norm at first trials, since BN layer can hide problem such as NaN in computation, explosion/vanishing gradients, and make debug harder
- With BatchNorm, make sure Batch size large enough (>16)

Check every steps, starting from pre-processing

Errors can happen in pre-processing, such as
forget to change label after processing images (e.g. image scaling for segmentation)

Errors can happen in datasets:
forget the shuffle the samples, some samples are corrupted …

Errors can happen when loading data:
image/label mismatch



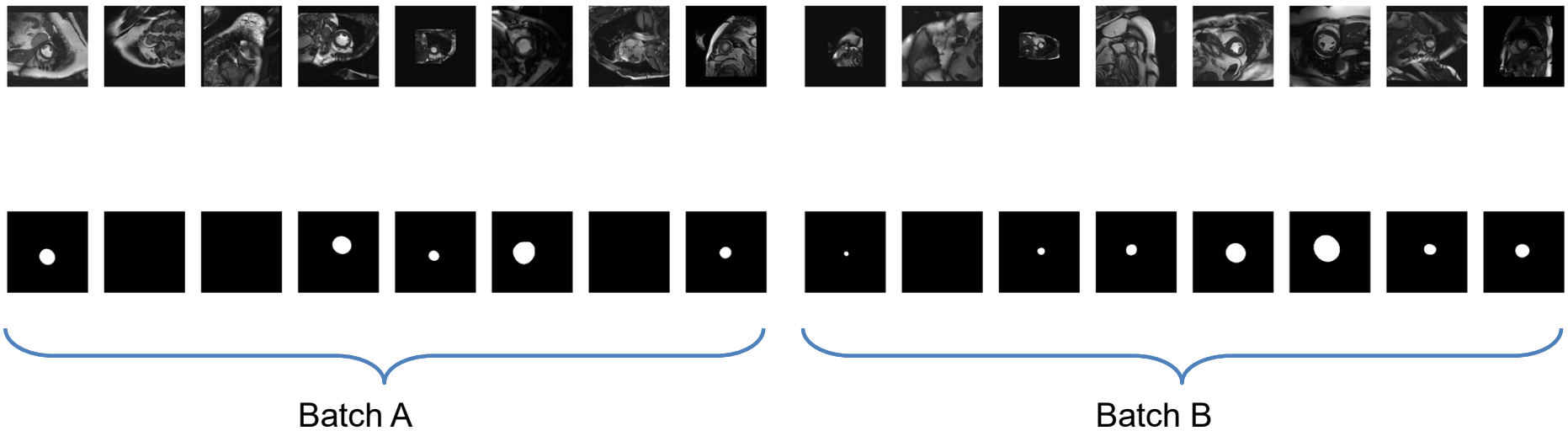
L2 norm of all samples

Some images are blanket!

Scale the image and forget to scale mask

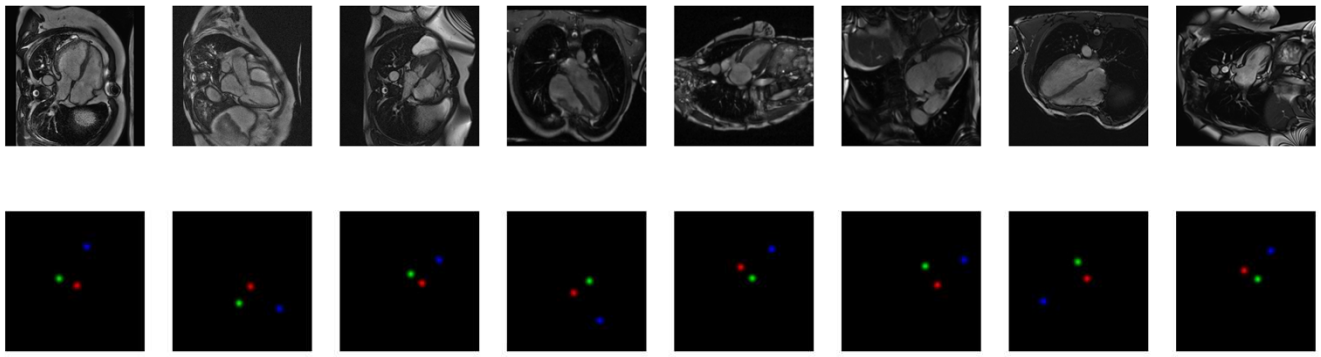
Check immediate inputs to your model training code

Print/plot/save a few mini-batch and check them before training

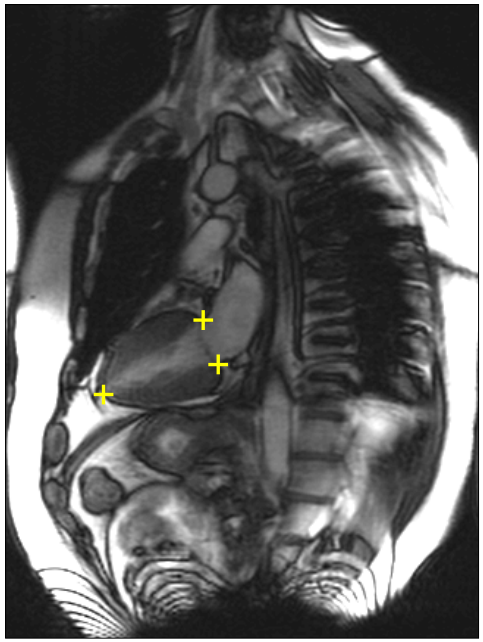


Check immediate outputs of your model before any post-processing

```
scores = model(images)
```



Model outputs are spatial probability maps



Goal : detect key landmarks

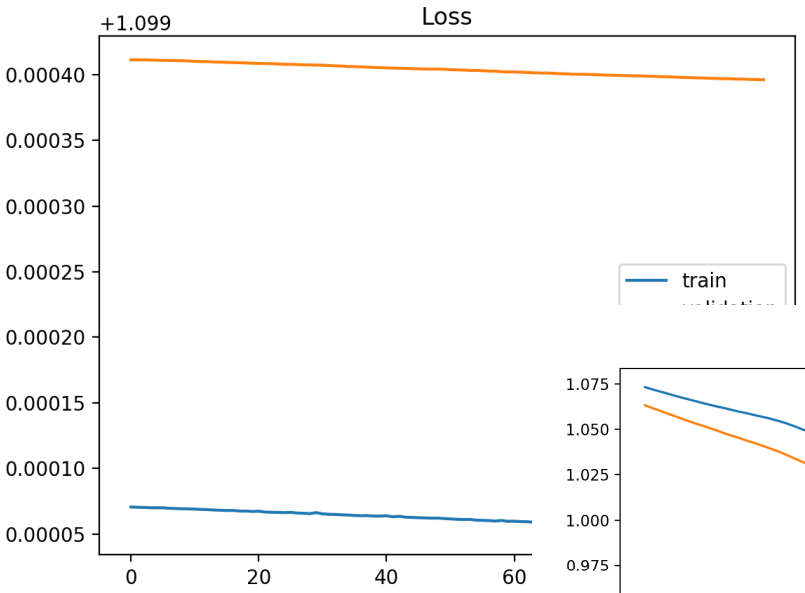
Postprocessing: extract landmark location

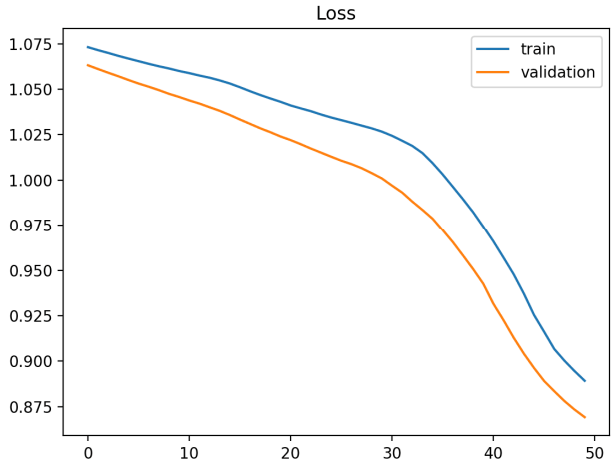
Add Debug mode and verbose mode, add working_dir

```
if(self.working_dir is not None):
    self.save_batch(self.working_dir, images, batch_no)

scores = model(images)

if(self.working_dir is not None):
    self.save_output(self.working_dir, scores, batch_no)
```

- Easily enable or disable
- Fast, not slow down training if disabled
- Save intermediate results with its index/name/time stamp/data role …
- Keep these code lines in your repo

Monitor the learning curves





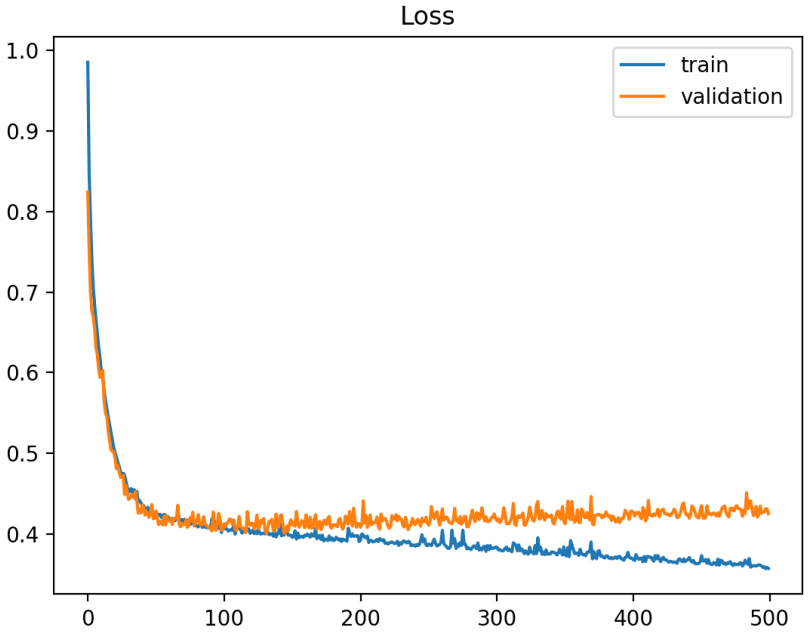
Loss is not decreasing quickly

learning rate too low
model underfitting

try to fix the problem by fitting to a tiny dataset

https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

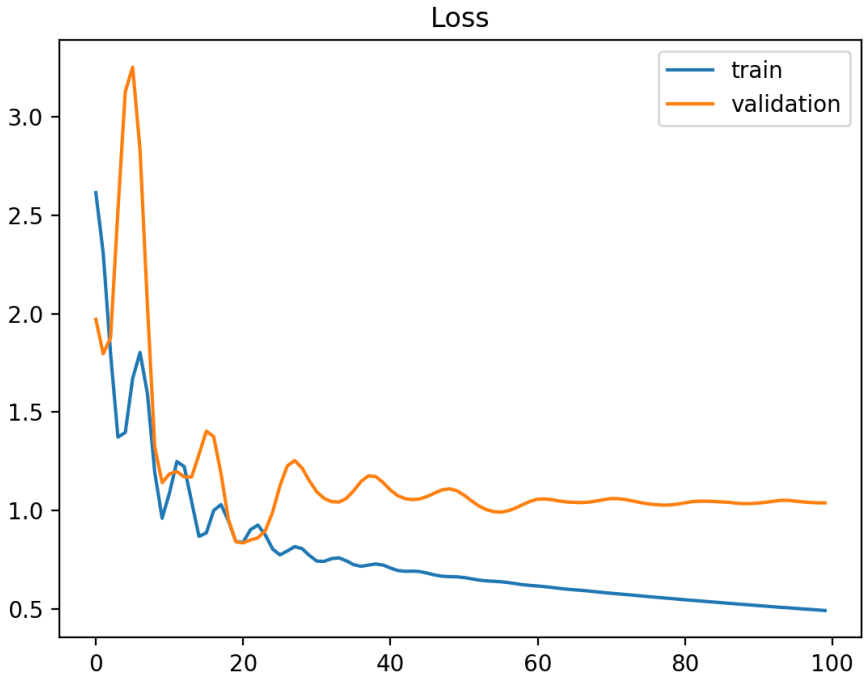
Monitor the learning curves



Loss goes up for dev set and goes down for training set

model overfitting
add regularization
reduce mode capacity
add more training data

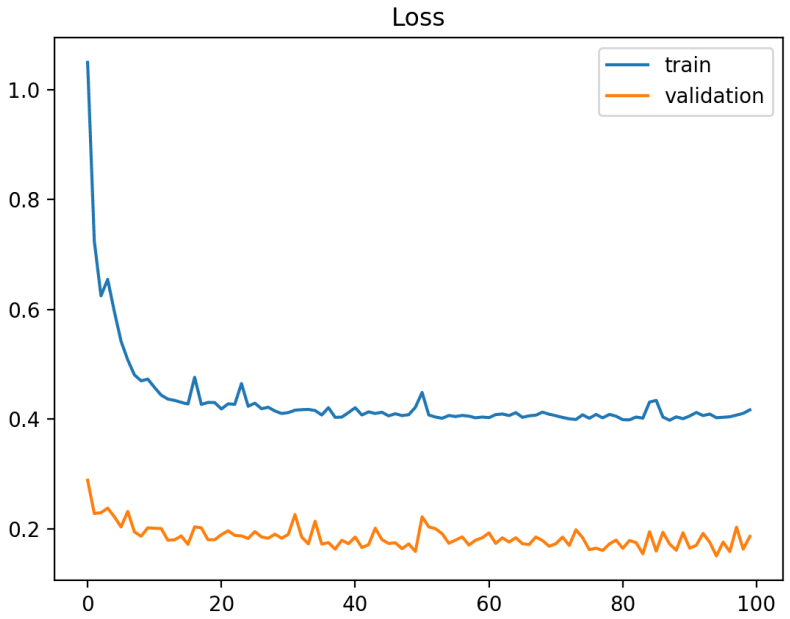
Monitor the learning curves



Loss shows strong variance in dev set

model overfitting
training set too small
Add more sample to training set

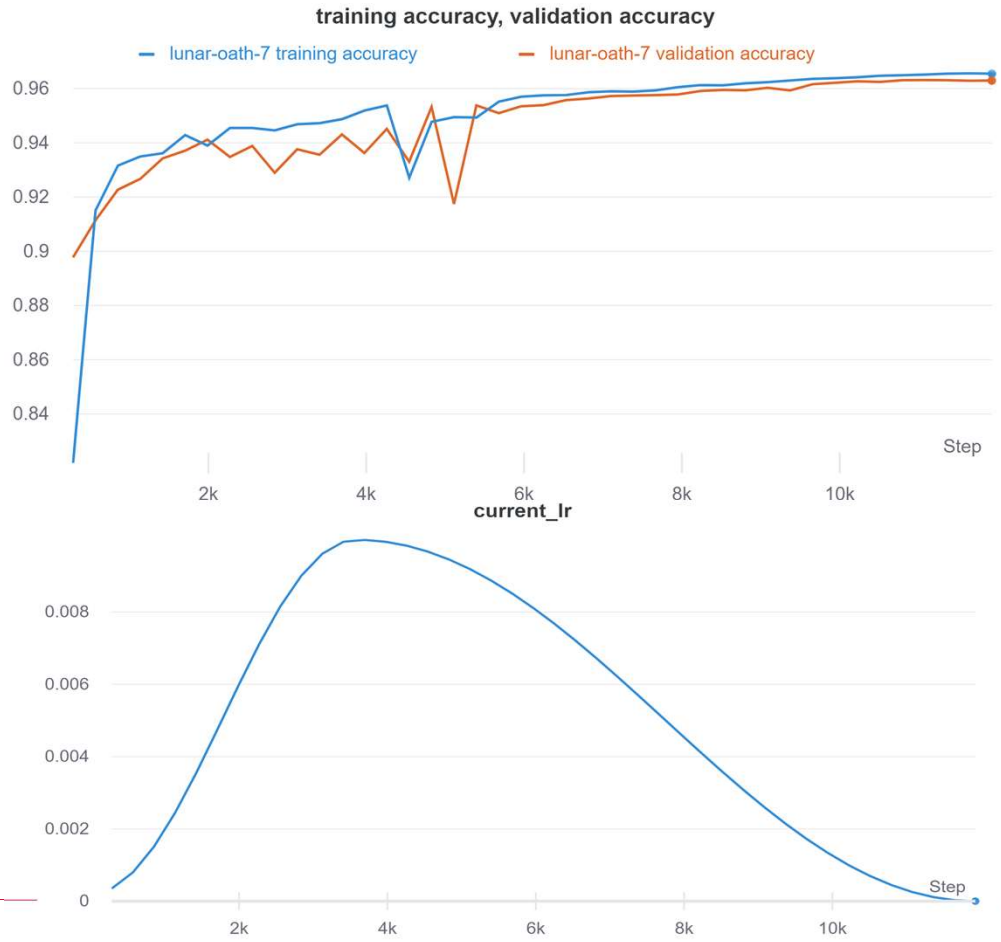
Monitor the learning curves



Loss is lower for dev set than train set

dev set is easier than train set
too much data augmentation
dev/tra set distribution mismatch
exam dev set

Monitor the learning curves



Good training

Accuracy is high, but still increasing
Consider to train longer

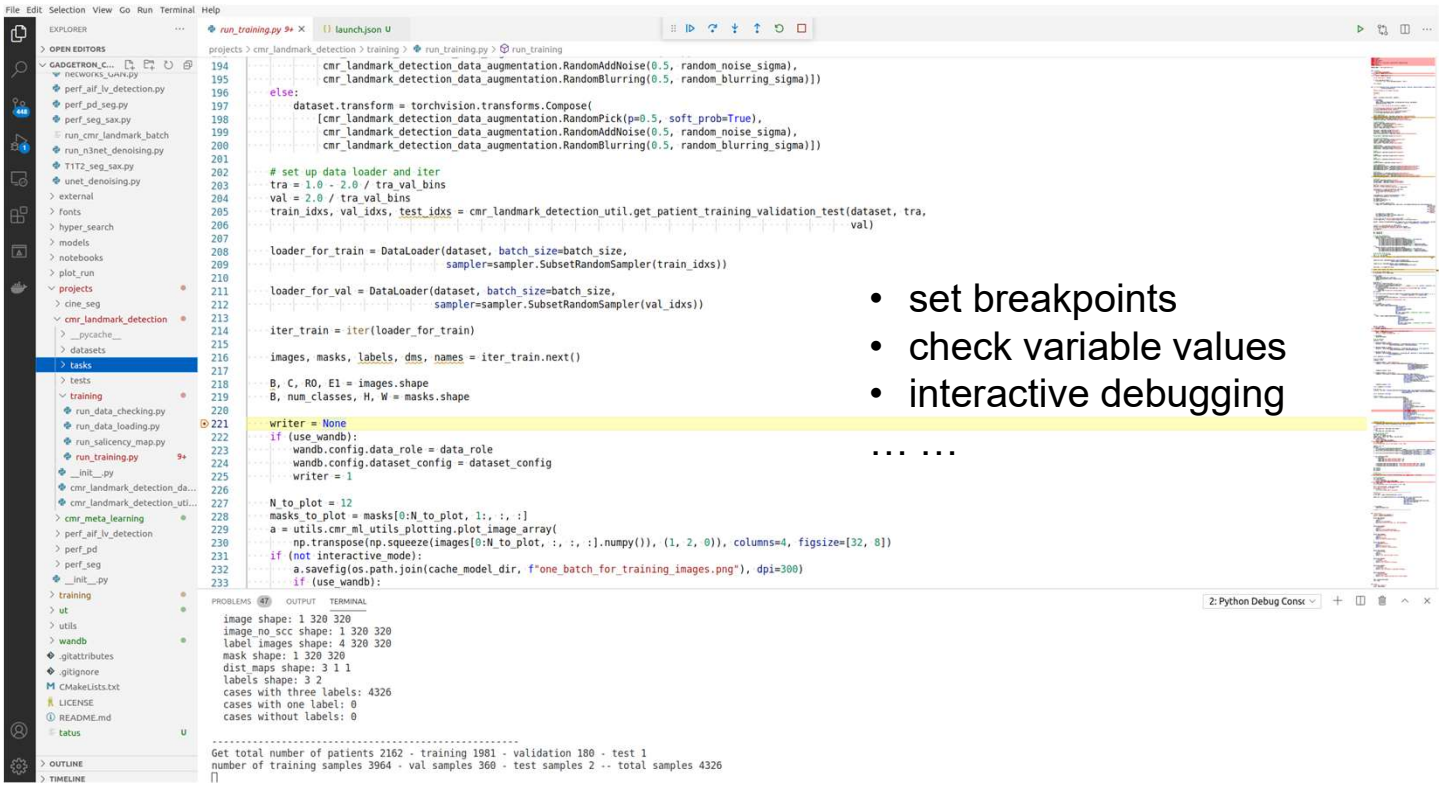
Note with one-cycle scheduler, at high LR, the accuracy can dip

Use Debugger

Tensor shape mismatch
Loss function exceptions
Nan or inf error
All other exceptions …

Visual Studio Code

PyCharm
The Python IDE
for Professional Developers



- set breakpoints
- check variable values
- interactive debugging

… …

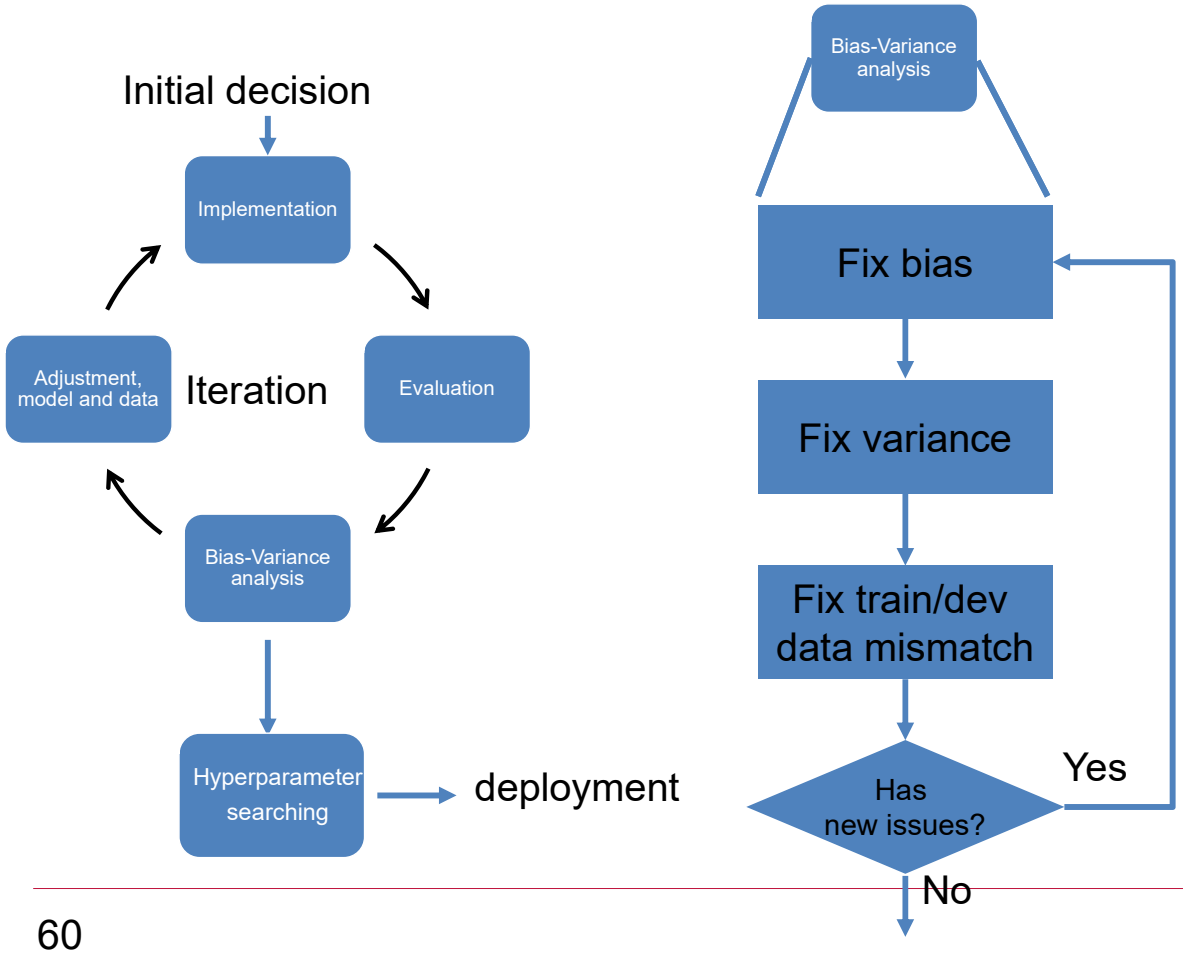
or, pdb or ipdb for CLI experiences of debugging

Use Experimental management tools

- Record your training and parameters
- Integrated with python training code
- Support hyperparameter searching
- Easy to compare different experiments

Many iterations may be needed to reach required performance

Do not use hyperparameter searching as the first remedy



- Start hyperparameter searching after fixing obvious bias/variance problems
- Often large performance gain is from error analysis and collecting specific new data for problems found
- Fix bias/variance problems can change model and which hyperparameters to tune