deep imaging

Lecture 12: CNN implementation, visualization and analysis of results

Machine Learning and Imaging

BME 548L

Roarke Horstmeyer

Class project – what are the first steps?

1. Think about it!
2. Discuss with your friends/others in the class (feel free to use Slack!) to form group
3. Look at previous projects: https://deepimaging.github.io/proj-past/
4. Schedule a short 15 meeting with myself or TA's
    - **Meetings will occur the week after spring break, will send out details soon**
5. Start to write-up a proposal
    - General aim: 1 paragraph, specifying physical layer or hardware analysis component
    - Discussion: (a) data source(s), (b) expected simulations, (c) expected CNN, (d) quantitative analysis of physical layer/physical component (comparison, plot, etc).

- Project proposal due date: **TBD (1-2 weeks)**
- Final project will be presented during final exam slot
- (note: due to large class size, this may go a bit over 3 hours, can maybe split in 2 sessions)

Example project topics:

deep imaging

Can we design a new lens/transducer/antenna shape to improve classification of X?

What is the tradeoff between image resolution and accuracy for X (classification, segmentation, etc.)? What if we had access to n low-resolution cameras – how might we position them to get the best performance?

Can we determine an optimal set of colors to improve fluorophore distinguishability?

How does classification accuracy change with sensor bit depth, down to the 1-bit level for single-photon detectors?

If we just had a few sensors, how should be arrange them e.g. a mask to be able to predict the position of X?

Is there some optimal shift-variant blur that we can to use for a particular task?

Or, given a shift-variant blurry image, can we establish a good deconvolution using locally connected layers?

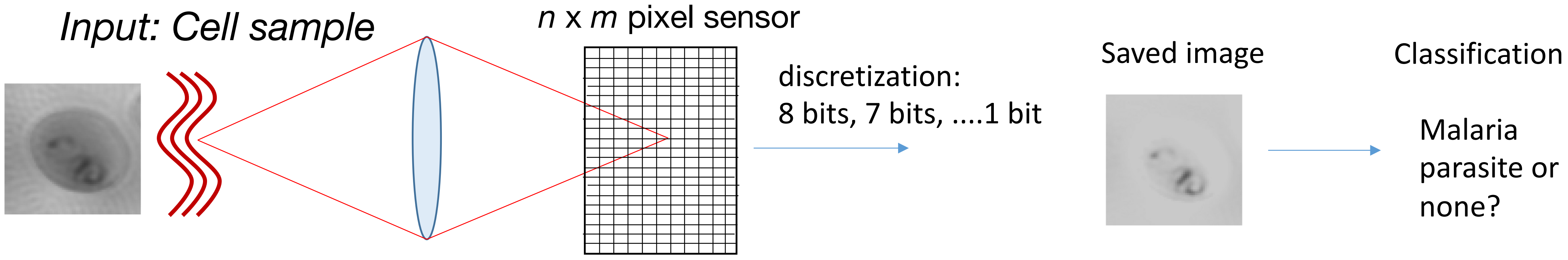
What is the optimal way to layout filters on a sensor to capture a color image for classification? Or an HDR image?

HDR image generation with filters over pixels – what is optimal design?

What if we could make a sensor with different sized pixels – how should they be laid out to achieve the best X?

How does classification accuracy change with sensor bit depth, down to the 1-bit level for single-photon detectors?

deep imaging



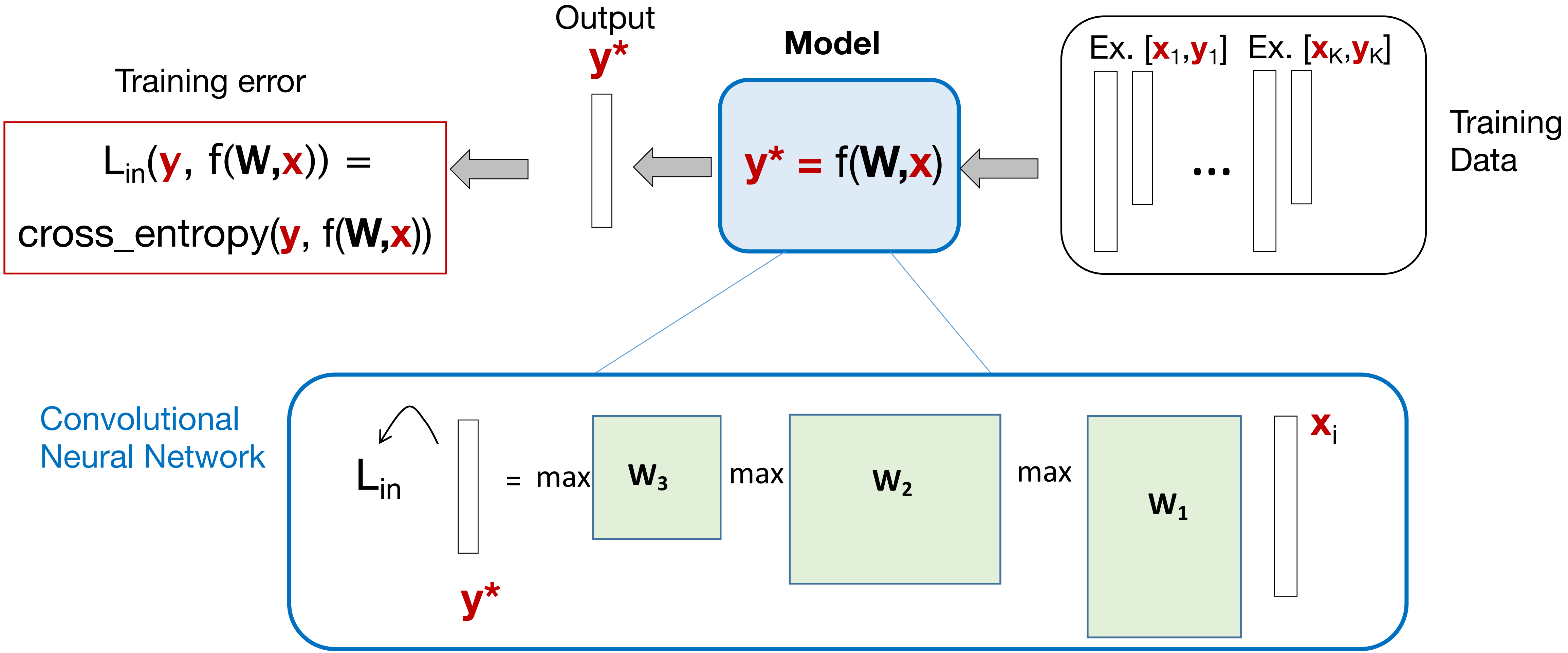
Physical layer test: per-pixel discretization (max. # bits/image)

I propose to test the classification performance of a microscope as a function of sensor bit depth (i.e., image discretization). I will plot average classification test accuracy as a function of number of sensor bits from 1 bit to 8 bits. I will additionally test whether the pixel discretization value can be optimized as a physical layer parameter. I will simulate a pixel discretization value, at each pixel, by multiplying the associated raw intensity value at each pixel by a weight, and will then using the max() operator to set a threshold. I will examine how classification accuracy varies with this additional constraint, and will attempt to draw insights into where the network prefers to have more bits/pixel.

Dataset: 12,500 images of 4 types of blood cell https://www.kaggle.com/paultimothymooney/blood-cells

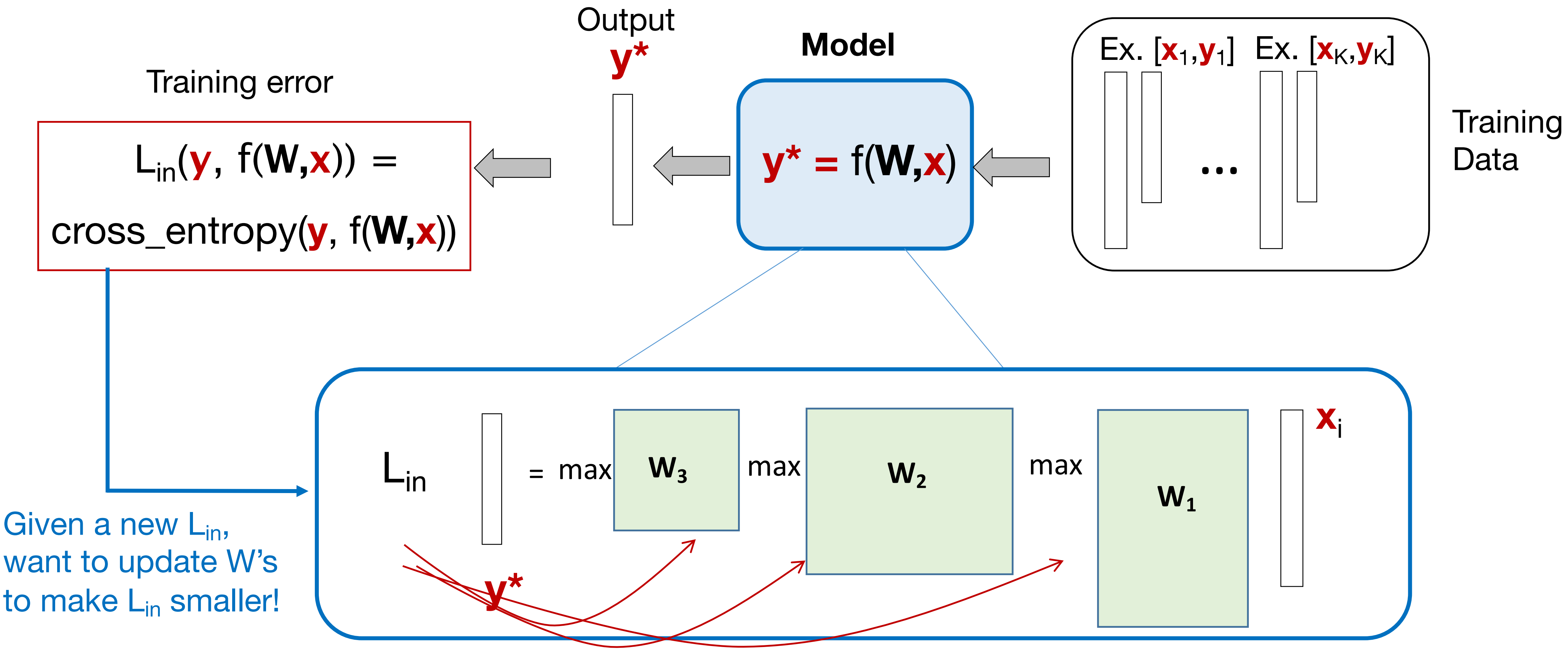
(Specify more details about simulation network, physical layer implementation and quantitative analysis)

Our very basic convolutional neural network



Forward pass: from $\mathbf{x}_i$ and current $\mathbf{W}$'s, find L_{in}

Our very basic convolutional neural network



Backwards pass: update **W** from new L_{in} via auto-differentiation

deep imaging

Important components of a CNN

CNN Architecture

Architecture choices

- CONV size, stride, pad, depth
- ReLU & other nonlinearities
- POOL methods
- # of layers, dimensions per layer
- Fully connected layers

Loss function & optimization

- Type of loss function
- Regularization
- Gradient descent method
- SGD batch and step size

Optimization choices

Other specifics: Initialization, dropout, batch normalization, data normalization & augmentation

Knobs to turn to get things to work…

Regularization: A common pattern

Training: Add some kind of randomness, *z*:

$$y = f_W(x, z)$$

Testing: Average out randomness (sometimes approximate)

$$y = f(x) = E_z\big[f(x, z)\big] = \int p(z) f(x, z) dz$$

Example: Batch Normalization

Training:
Normalize using stats from random minibatches

Testing: Use fixed stats to normalize

Slide from http://cs231n.stanford.edu/

deep imaging

Regularization: A common pattern

Training: Add some kind of randomness, *z*:

$$y = f_W(x, z)$$

Testing: Average out randomness (sometimes approximate)

$$y = f(x) = E_z\big[f(x, z)\big] = \int p(z) f(x, z) dz$$

Example: Batch Normalization

Training: Normalize using stats from random minibatches

Testing: Use fixed stats to normalize

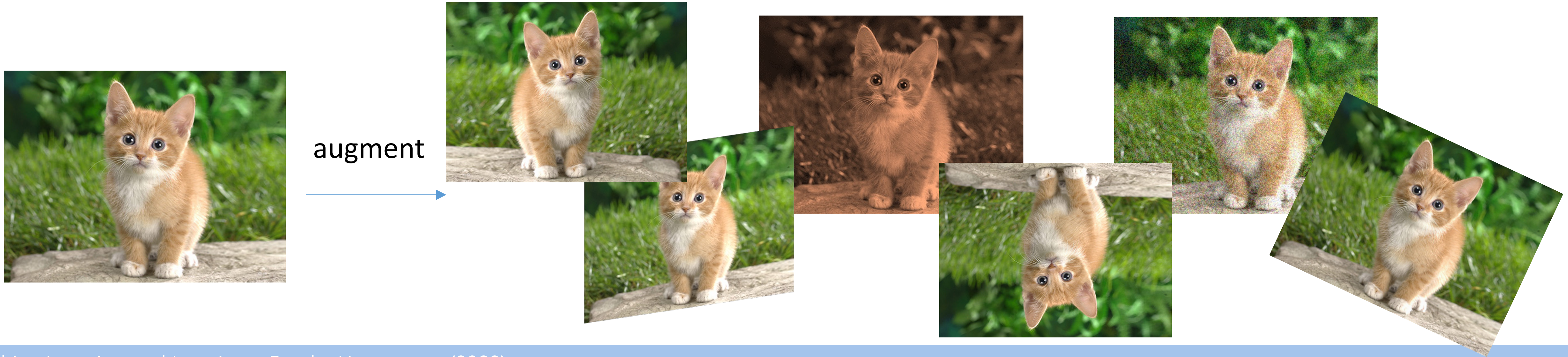
Obvious examples: Dropout, data augmentation
Advanced examples: DropConnect, Fractional Max Pooling, Stochastic Depth

Wan et al, "Regularization of Neural Networks using DropConnect", ICML 2013
Graham, "Fractional Max Pooling", arXiv 2014
Huang et al, "Deep Networks with Stochastic Depth", ECCV 2016

[Slide from http://cs231n.stanford.edu/](http://cs231n.stanford.edu/)

Regularization Ex. 1: Data augmentation

- Basic idea: to simulate variation that you might actually see in real life
- It's a form of regularization
- Not an exact science, but try it out – it's free!



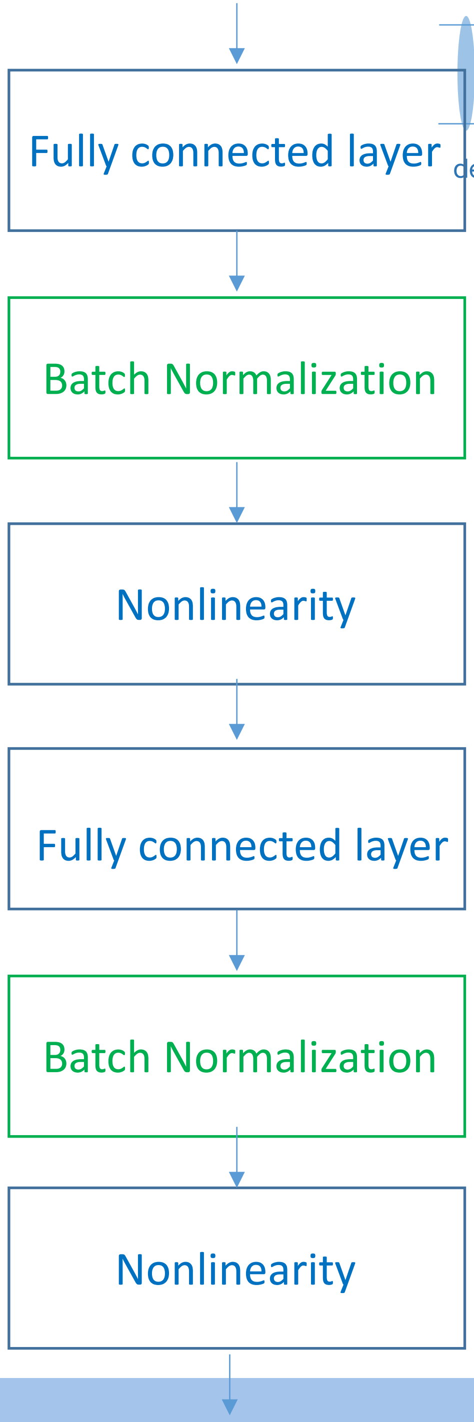
Regularization Ex. 2: Batch normalization (BN)

- Before BN, training very deep networks was hard
  - If using sigmoid activations, large weights could result in saturation
  - Updating earlier layers' weights causes the distribution of weights in later layers to shift – the *internal covariate shift*
- To address this covariate shift, BN "resets" the layer it is applied to by normalizing to 0 mean, 1 variance
  - Mean and variance are computed over the batch at the current iteration

Batch normalization update for inputs x:

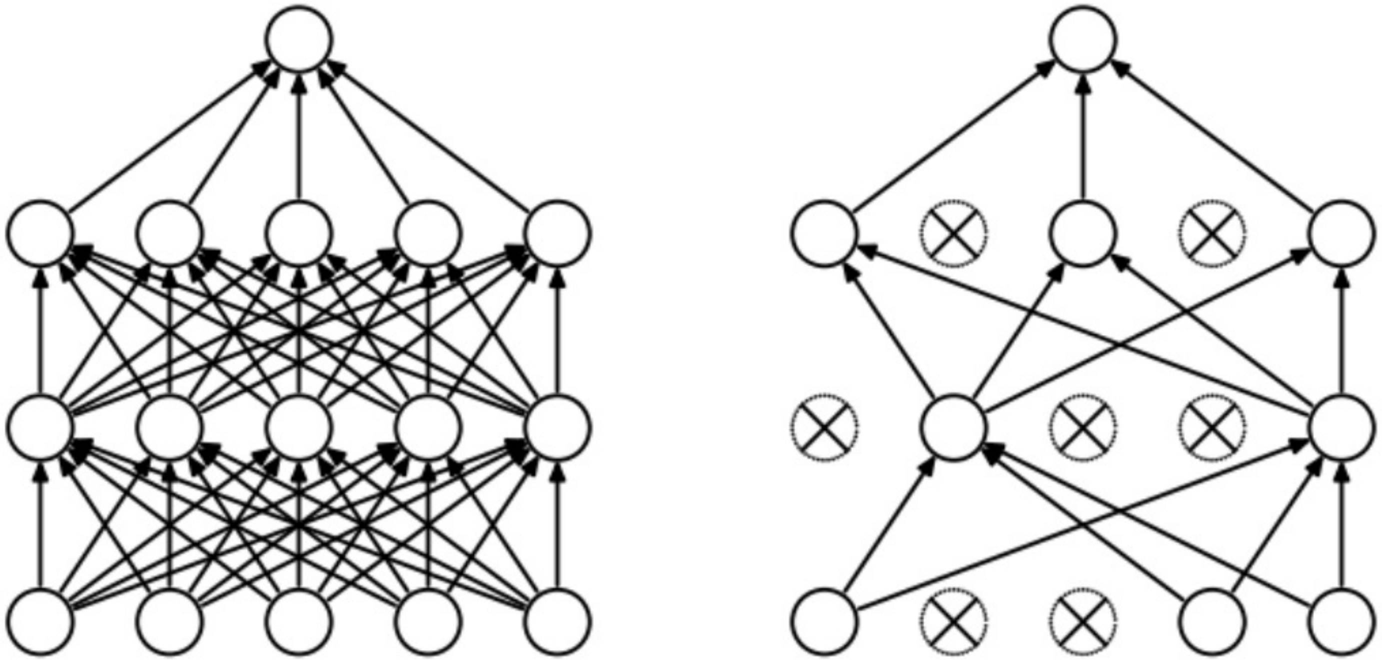
$$\boxed{x'(i) = (x(i) - E[x(i)]) / STD[x(i)]}$$

- Mean subtract
- Normalize by standard deviation



Regularization Ex. 3: Dropout

- At each train iteration, randomly delete a fraction p of the nodes
- Prevents neurons from being lazy
- A form of model averaging
- (related: DropConnect – drop the connections instead of nodes)



(a) Standard Neural Net (b) After applying dropout.

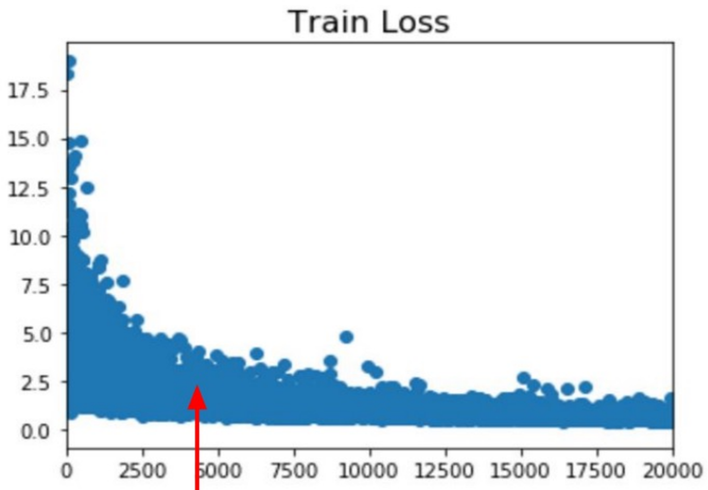
https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5

Let’s identify the following in some example code

- Structure of input/output
- Train/Validation/Test split
- Cost function
- Optimization method, steps (epochs)
- Batch size
- Data augmentation?
- Dropout?

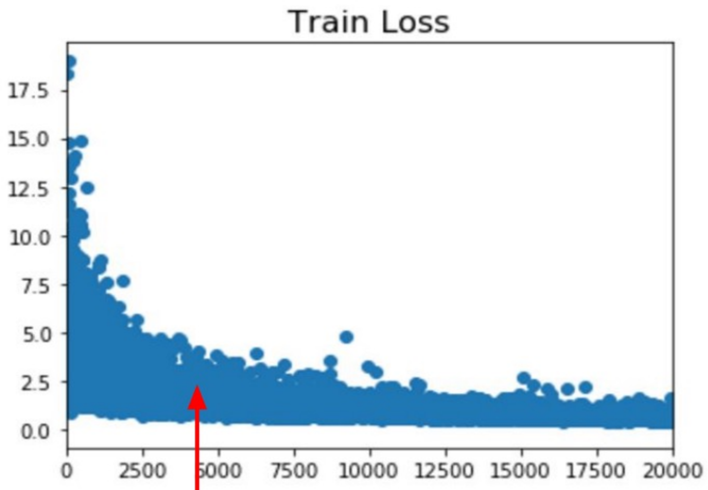
https://deepimaging.github.io/data/high_level_tf_intro.ipynb

What you'll typically see…



Better optimization algorithms
help reduce training loss

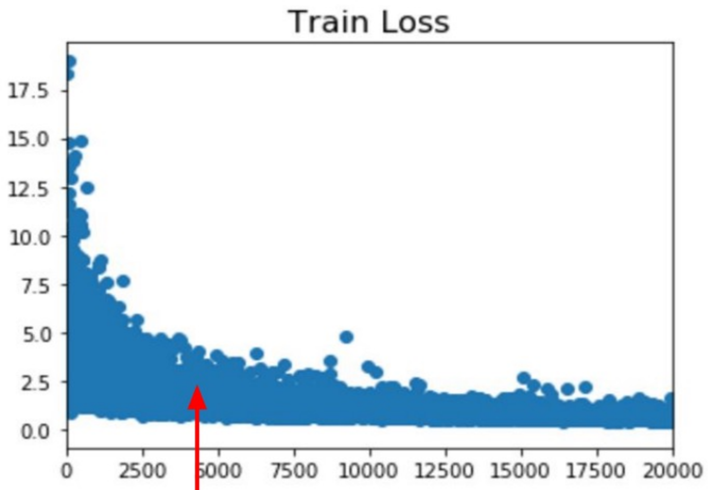
What you'll typically see…

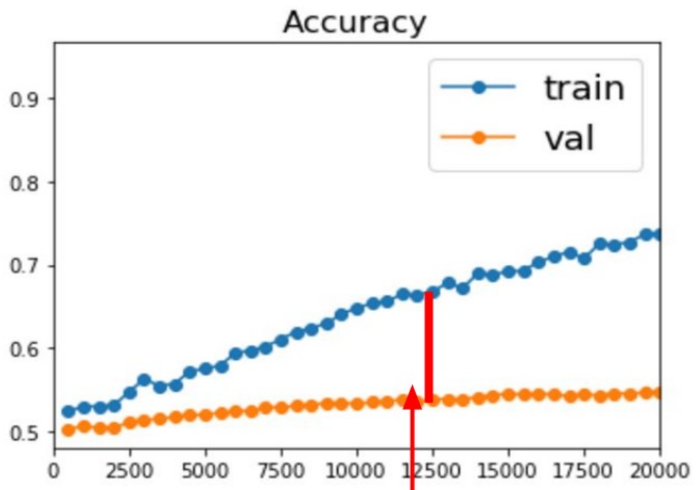


Better optimization algorithms
help reduce training loss

What you can do to help out training error:
- Optimizer choice
- Optimizer step size

What you'll typically see…





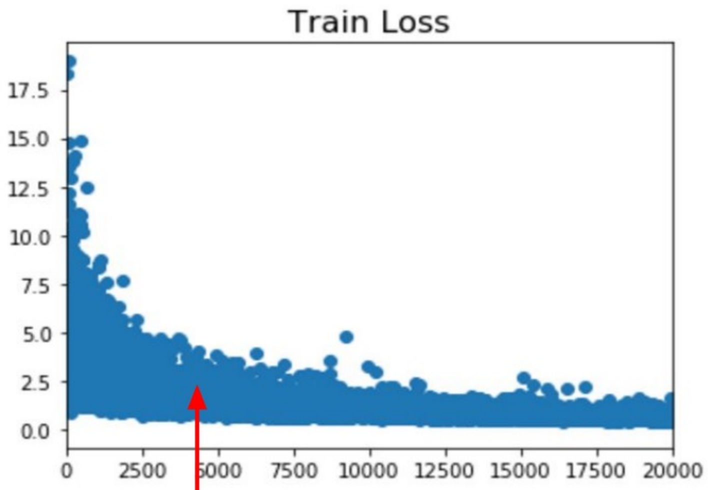
Better optimization algorithms help reduce training loss

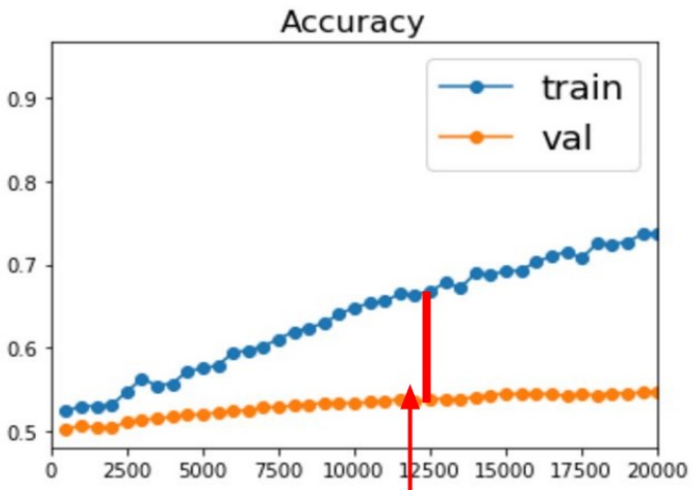
But we really care about error on new data - how to reduce the gap?

What you can do to help out training error:

- Optimizer choice
- Optimizer step size

What you'll typically see…





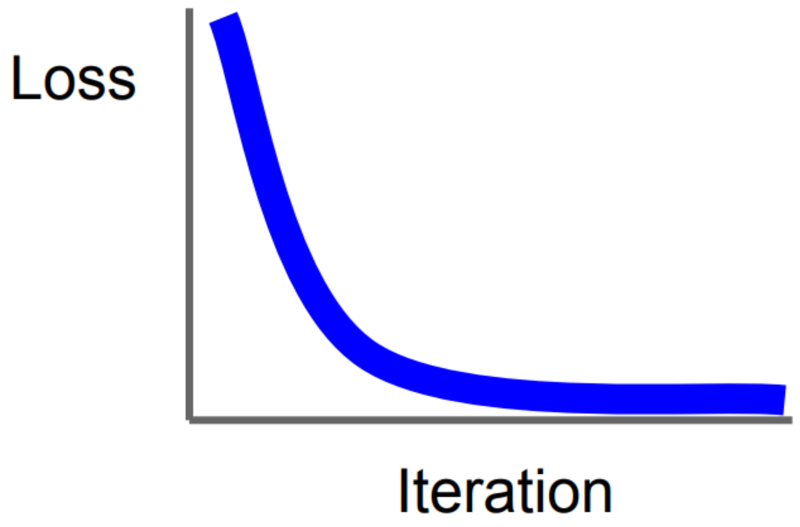
Better optimization algorithms help reduce training loss

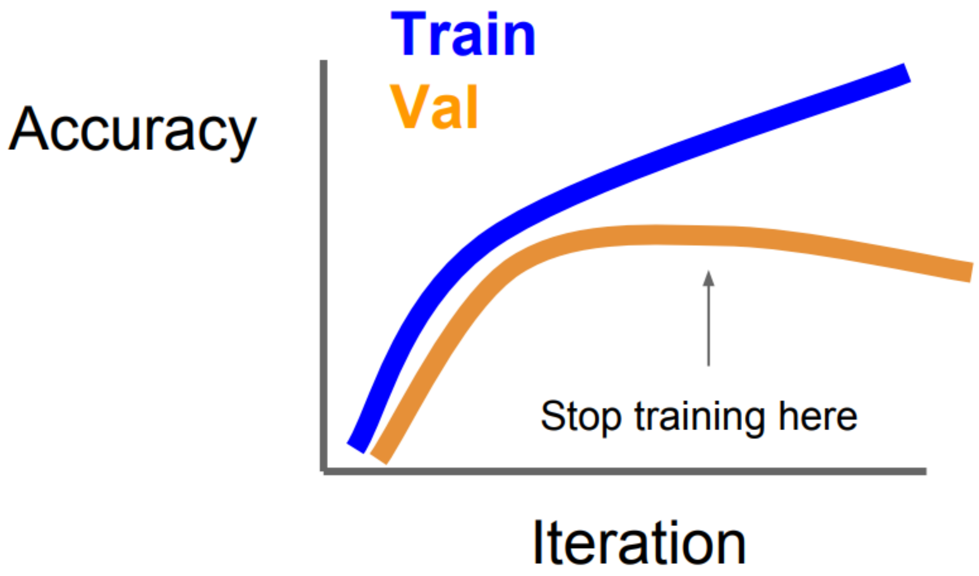
But we really care about error on new data - how to reduce the gap?

What you can do to help out training error:
- Optimizer choice
- Optimizer step size

What you can do to help out training error:
- More regularization!
  - Dropout
- Data normalization
- Data augmentation
- A few other tricks..

Trick #1: Early stopping





Stop training the model when accuracy on the validation set decreases
Or train for a long time, but always keep track of the model snapshot that worked best on val

Slide from http://cs231n.stanford.edu/

Trick #2: Use Model Ensembles

1. Train multiple independent models
2. At test time average their results
   (Take average of predicted probability distributions, then choose argmax)

Enjoy 2% extra performance

Related concept/term: majority voting

E.g., look at *same* dog image from test data 9X, each w/ uniquely trained model

- Get (let's say) [6, 3] for output classification
- so guess [1,0] = it's a dog
- Will do better than running model once!

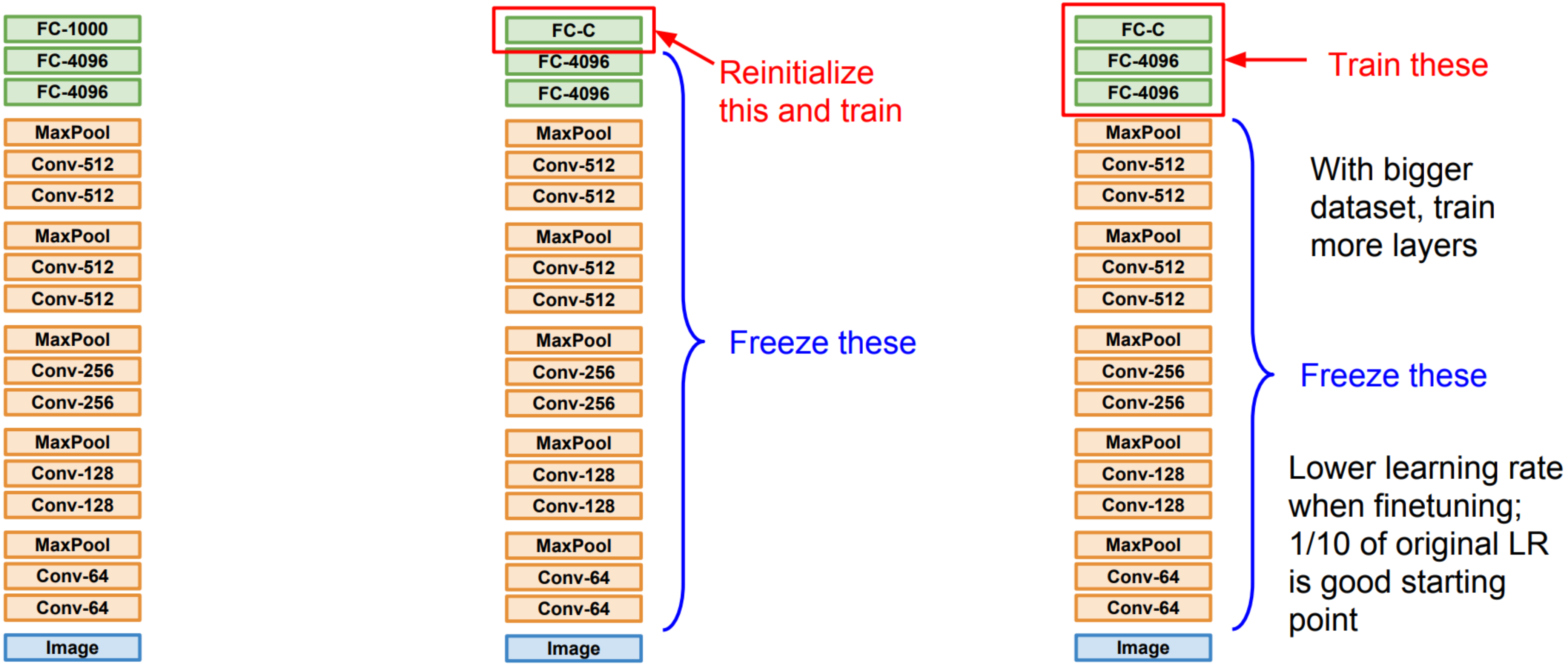
Related technique: Test Time Augmentation
Also relevant: Dropout-type methods

Trick #3: Transfer learning

Transfer Learning with CNNs

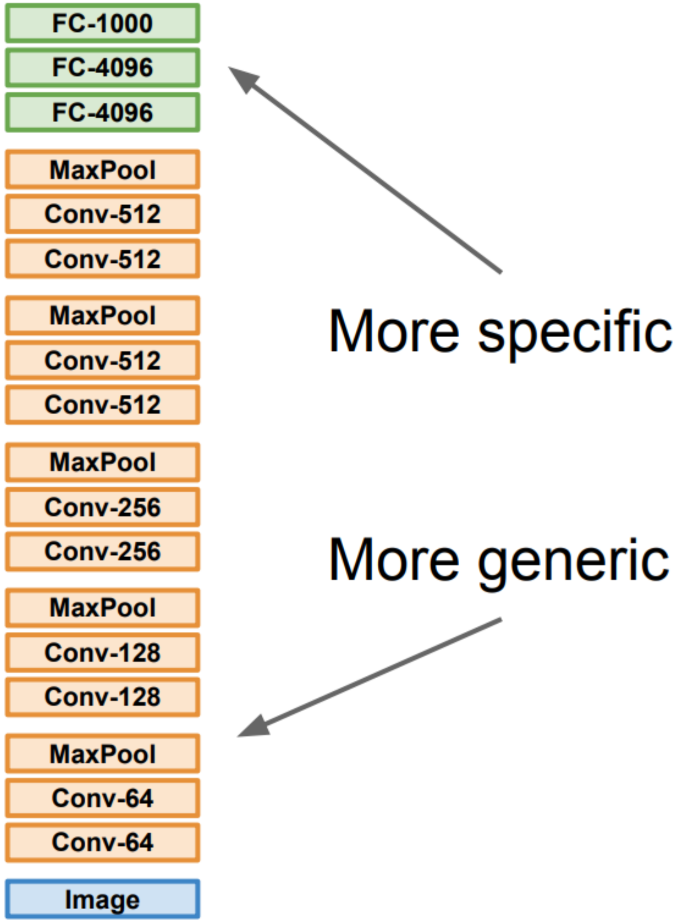
Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

1. Train on Imagenet
2. Small Dataset (C classes)
3. Bigger dataset



Slide from http://cs231n.stanford.edu/

Trick #3: Transfer learning

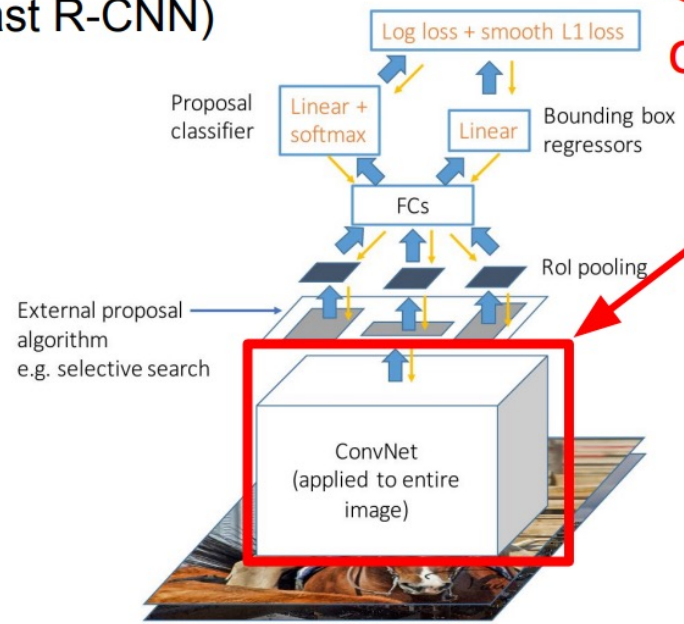


	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Slide from http://cs231n.stanford.edu/

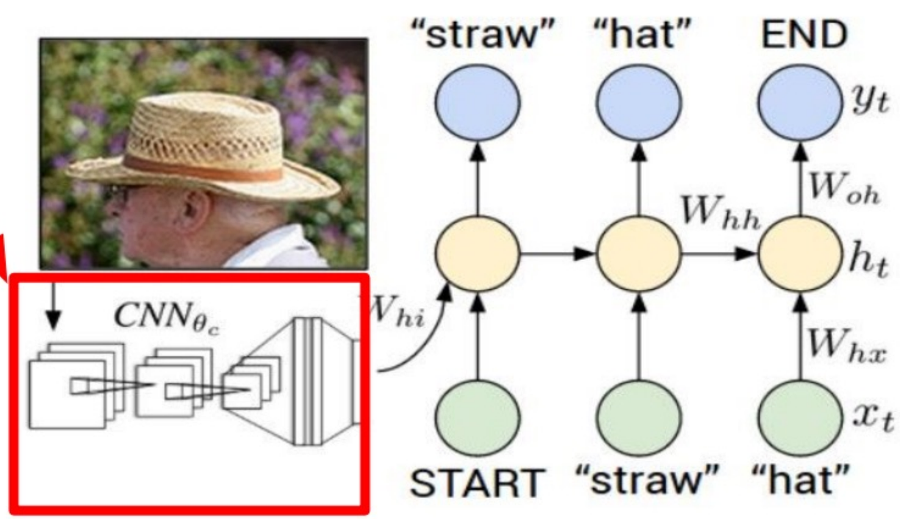
Transfer learning with CNNs is pervasive…
(it's the norm, not an exception)

Object Detection
(Fast R-CNN)



CNN pretrained on ImageNet

Image Captioning: CNN + RNN



Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

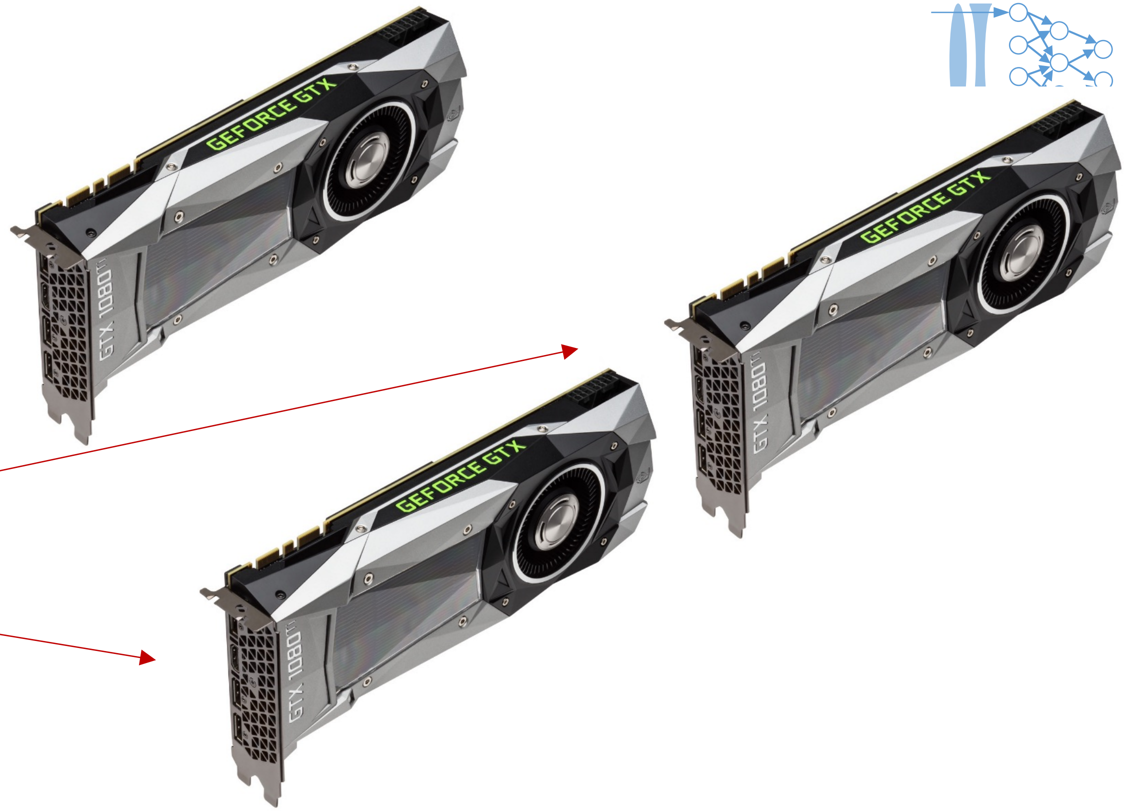
[Slide from http://cs231n.stanford.edu/](http://cs231n.stanford.edu/)

Trick #4: Hyperparameter optimization

For learning_rate in range(9):

For gradient_scheme in range(5):

⋮



Meta-learning

B. Baker et al., “Designing neural network architectures using reinforcement learning,” arXiv 2017

E. Real, ” Large-Scale Evolution of Image Classifiers,” ICML 2017

Visualization: a few options at different stages

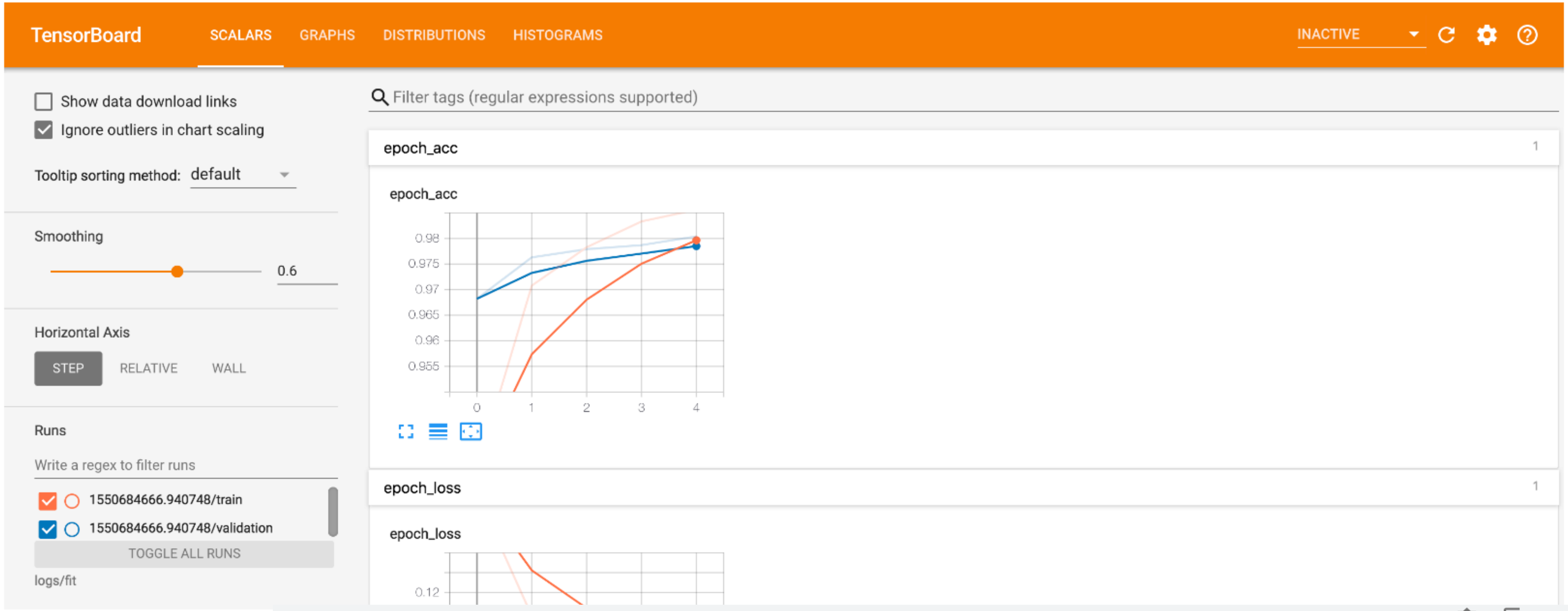
During Training:

- tf.summary()
- Tensorboard
  - Plots of loss/accuracy versus iteration, etc.

After Testing:

- Sliding window
- ROC curve, Precision-Recall
- Confusion matrix
- tSNE visualization

- Beyond classification:
  - image-to-image similarity metrics
  - segmentation accuracy and overlap metrics



https://colab.research.google.com/github/tensorflow/tensorboard/blob/master/docs/graphs.ipynb

```
model = create_model()
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

model.fit(x=x_train,
          y=y_train,
          epochs=5,
          validation_data=(x_test, y_test),
          callbacks=[tensorboard_callback])
```

```
%tensorboard --logdir logs/fit
```

deep imaging

How to examine and present your results: a few options at different stages

Options to examine your test data after processing:

- ROC curve, Precision-Recall
- Confusion matrix
- Sliding window visualization
- Layer visualizations
- Saliency maps etc.
- tSNE visualization

ROC curve and confusion matrix

- Can set threshold for f(x,W) wherever
- Leads to sliding window between FN and FP rate
- Need to summarize both statistics as a function of sliding window

Estimated label

f(x, W)

Actual label y		+1	-1
	+1	True positive	False negative
	-1	False positive	True negative

Missed an event

Predict event when there isn't one

deep imaging

ROC curve and confusion matrix

deep imaging

TP Rate =

Sensitivity = TP / (TP + FN) = TP / Actual positives

False Positive Rate = FP / (TN + FP) = FP/ Actual negatives

Specificity = TN / (TN + FP) = TN / Actual negatives
= 1 – False Positive Rate

Estimated label

$f(x, W)$

Missed an event

Actual label y		+1	-1
	+1	True positive	False negative
	-1	False positive	True negative

Predict event when there isn't one

ROC curve and confusion matrix

TP Rate =

Sensitivity = TP / (TP + FN) = TP / Actual positives

False Positive Rate = FP / (TN + FP) = FP/ Actual negatives

Specificity = TN / (TN + FP) = TN / Actual negatives
= 1 – False Positive Rate

Estimated label

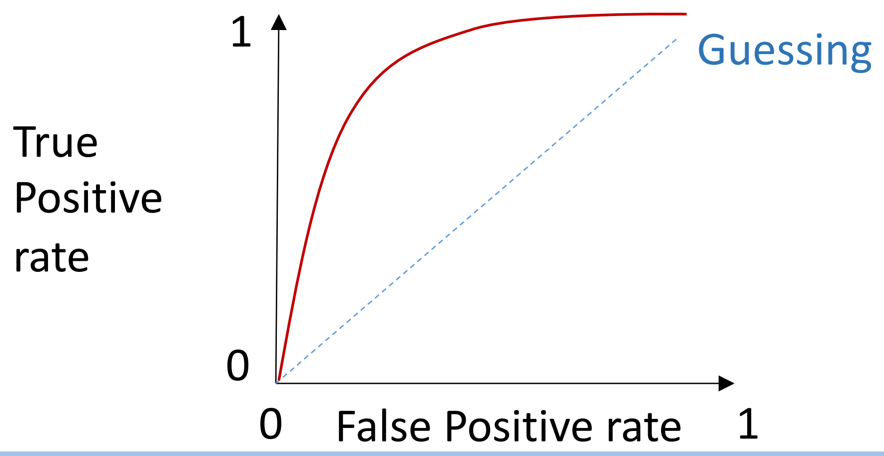
$f(x, W)$

		+1	-1
Actual label y	+1	True positive	False negative
	-1	False positive	True negative

Missed an event

Predict event when there isn't one

Receiver-Operator Curve



ROC curve and confusion matrix

TP Rate =

Sensitivity = TP / (TP + FN) = TP / Actual positives

False Positive Rate = FP / (TN + FP) = FP/ Actual negatives

Specificity = TN / (TN + FP) = TN / Actual negatives
= 1 – False Positive Rate

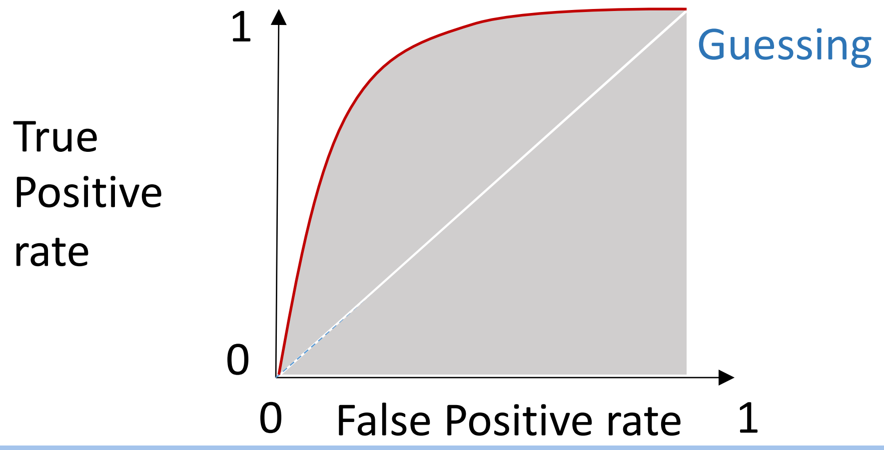
Estimated label $f(x, W)$

		+1	-1
Actual label y	+1	True positive	False negative
	-1	False positive	True negative

Missed an event

Predict event when there isn't one

Receiver-Operator Curve



Area under the curve (AUC): Integral of ROC curve

ROC curve and confusion matrix

deep imaging

Recall =
Sensitivity = TP / (TP + FN) = TP / Actual positives

Estimated label
$f(x, W)$

Actual label y	+1	-1
+1	True positive	False negative
-1	False positive	True negative

Missed an event

Predict event when there isn't one

Precision = TP / (TP + FP) = TP / Estimated positives

- Sometimes, you don't care about true negatives (just want to find events)
- In this case, use Precision and Recall

ROC curve and confusion matrix

Recall =
Sensitivity = TP / (TP + FN) = TP / Actual positives

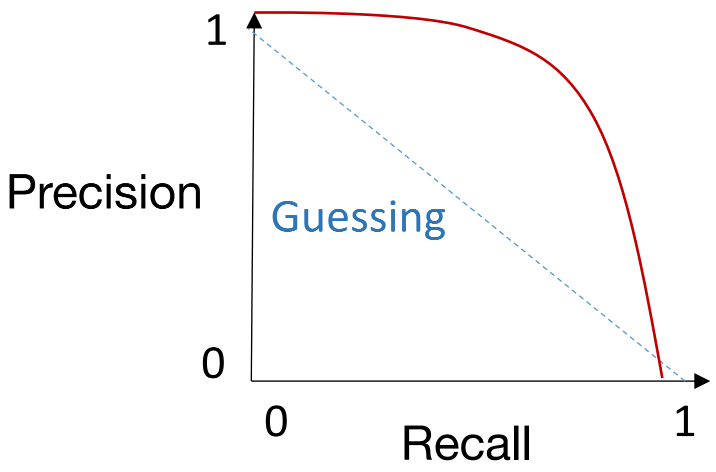
Missed an event

		Estimated label $f(x, W)$	
		+1	-1
Actual label y	+1	True positive	False negative
	-1	False positive	True negative

Predict event when there isn't one

Precision = TP / (TP + FP) = TP / Estimated positives

Precision-Recall curve



F1 Metric: $(1/\text{precision} + 1/\text{recall})^{-1}$

ROC curve and confusion matrix

deep imaging

Just 2 categories

Estimated label

f(x, W)

y		+1	-1
y	+1	True positive	False negative
y	-1	False positive	True negative

Confusion Matrix: 2+ categories

Estimated label

f(x, W)

y

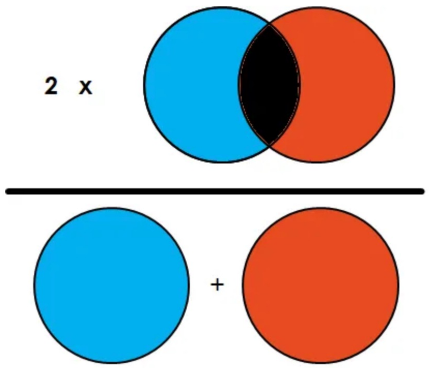
Actual ↓	State1 (Predicted)	State2 (Predicted)	State3 (Predicted)	State4 (Predicted)	State5 (Predicted)	State6 (Predicted)	State7 (Predicted)	State8 (Predicted)
State1 (Actual)	90.12 %	0.00 %	9.88 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
State2 (Actual)	0.00 %	100.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
State3 (Actual)	0.00 %	0.00 %	92.66 %	0.00 %	0.00 %	7.34 %	0.00 %	0.00 %
State4 (Actual)	0.00 %	0.00 %	0.00 %	100.00 %	0.00 %	0.00 %	0.00 %	0.00 %

Other performance metrics

- Overlap between segmented areas: Jaccard similarity coefficient
  - (also called Intersection over Union, IoU)

J = |R1 ∩ R2| / |R1 U R2|

- Dice Coefficient (F1 score): 2 x (total area of overlap) / total number of pixels in both images



- MSE, PSNR

- Structural Similarity (SSIM)

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

with:

- μ_x the average of x;
- μ_y the average of y;
- σ_x^2 the variance of x;
- σ_y^2 the variance of y;
- σ_{xy} the covariance of x and y;
- $c_1 = (k_1 L)^2$, $c_2 = (k_2 L)^2$ two variables to stabilize the division with weak denominator;
- L the dynamic range of the pixel-values (typically this is $2^{\#bits\ per\ pixel} - 1$);
- $k_1 = 0.01$ and $k_2 = 0.03$ by default.