Machine Learning in Imaging

BME 590L
Roarke Horstmeyer

Lecture 12: CNN implementation, tracking and results

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

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

Important components of a CNN

CNN Architecture

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

Architecture choices

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…

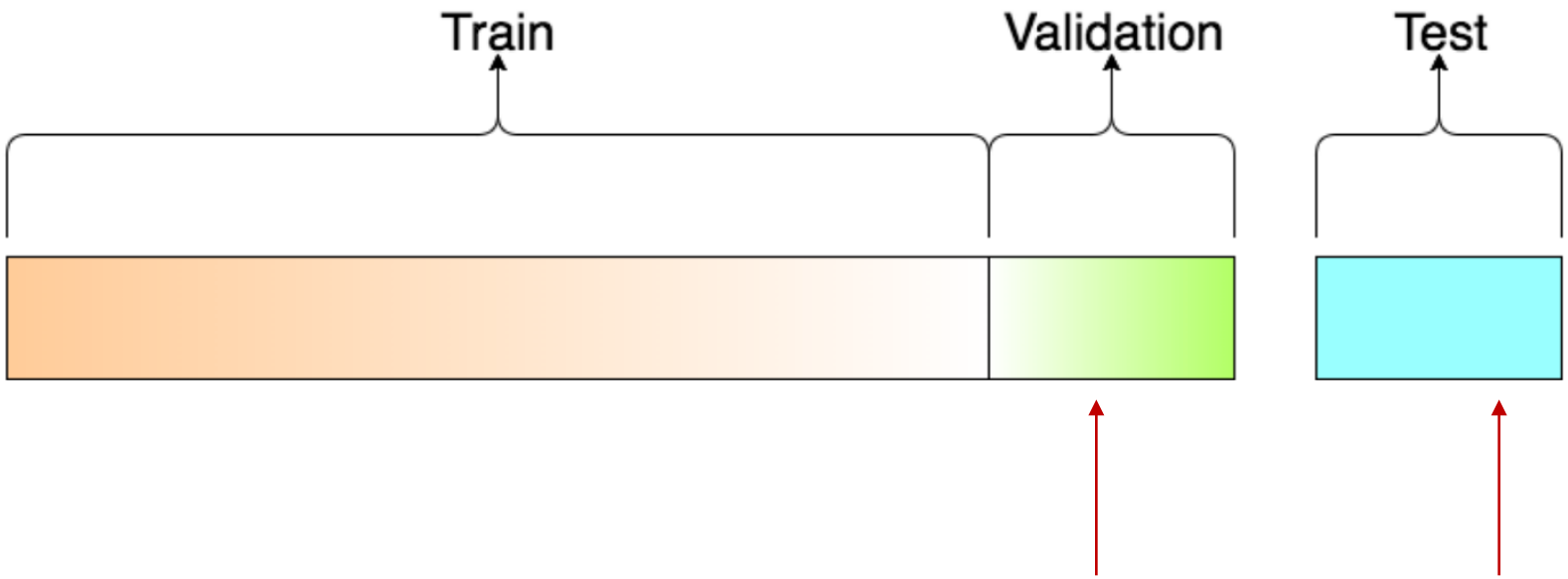
Review: What Kevin talked about

- Data augmentation
- Normalization
- Dropout
- Minibatch gradient descent
- Momentum

Today:

1) Let's look at some example code
2) Run through how you might bug-check & improve performance
3) How to view and interpret the output
4) Let's see some examples!

Training dataset, test dataset and validation dataset



Use to evaluate while tuning hyperparameters

- effect will creep into model as you continue to use it

For the “final” test

Let's identify the following in Ouwen's example code

- Structure of input/output
- Normalization
- Train/Validation/Test split
- Cost function
- Optimization method
- Batch size
- Data augmentation?
- Dropout?

Regularization: A common pattern

Training: Add some kind of randomness

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

Testing: Average out randomness (sometimes approximate)

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

Example: Batch Normalization

Training: Normalize using stats from random minibatches

Testing: Use fixed stats to normalize

Regularization: A common pattern

Training: Add some kind of randomness

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

Testing: Average out randomness (sometimes approximate)

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

Example: Batch Normalization

Training: Normalize using stats from random minibatches

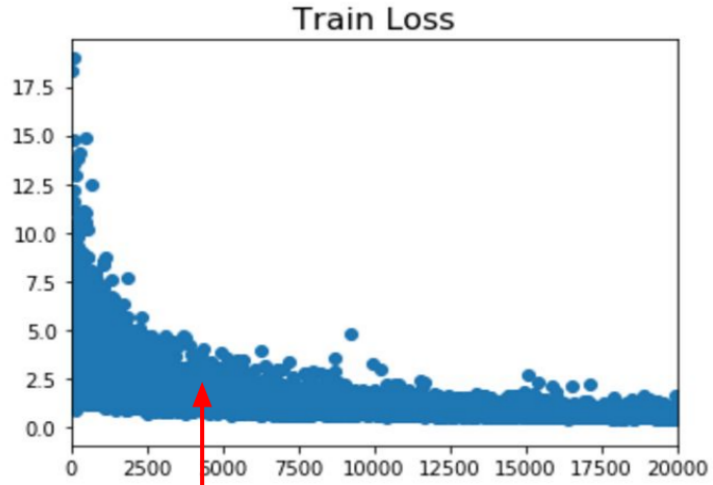
Testing: Use fixed stats to normalize

Other obvious examples: Dropout, data augmentation

Advanced examples: DropConnect, Fractional Max Pooling, Stochastic Depth

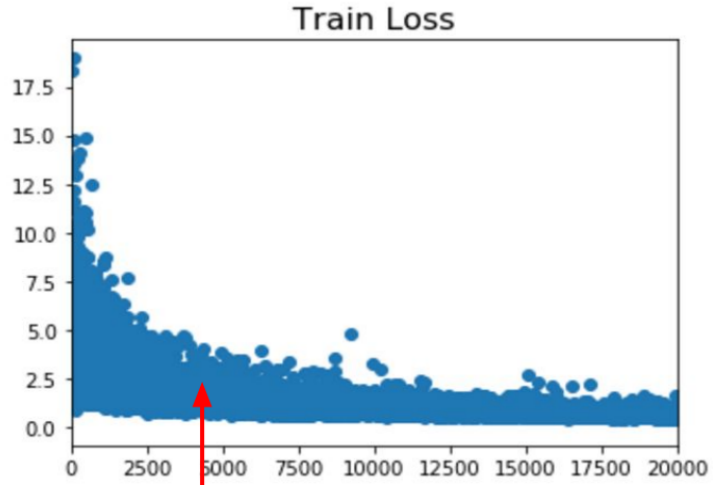
Wan et al, "Regularization of Neural Networks using DropConnect", ICML 2013
Graham, "Fractional Max Pooling", arXiv 2014
Huang et al, "Deep Networks with Stochastic Depth", ECCV 2016

What you'll typically see…



Better optimization algorithms
help reduce training loss

What you'll typically see…

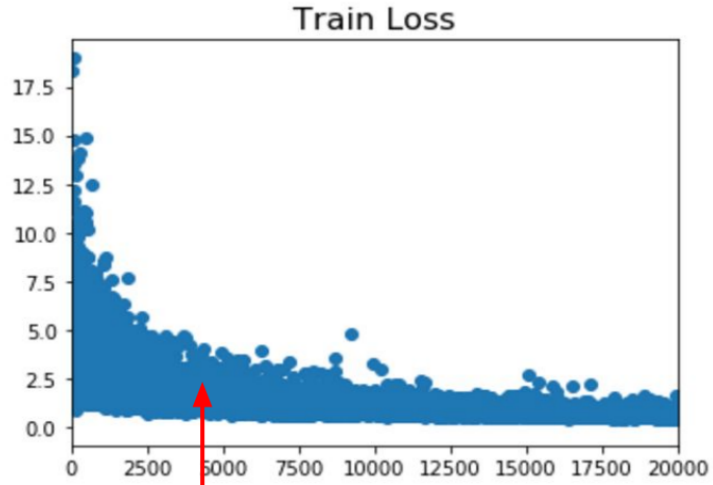


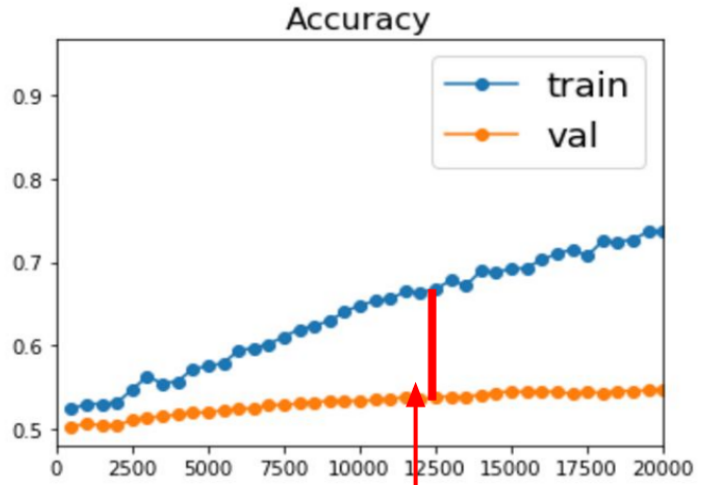
Better optimization algorithms
help reduce training loss

What you can do to help out training error:

- Optimizer choice
- Optimizer step size

What you'll typically see…





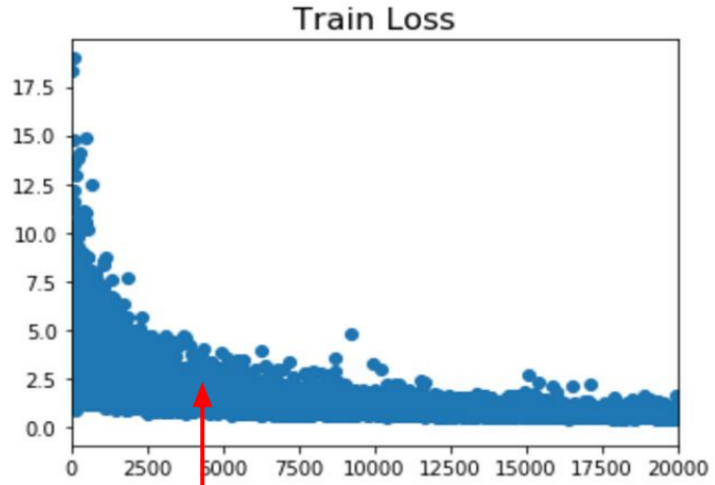
Better optimization algorithms help reduce training loss

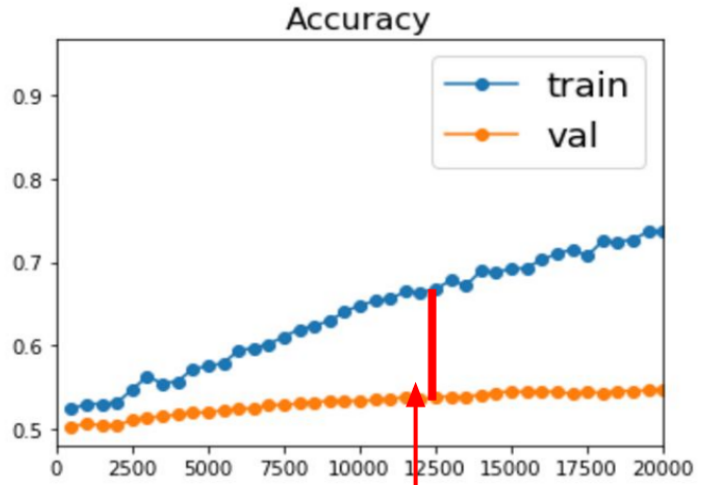
But we really care about error on new data - how to reduce the gap?

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?

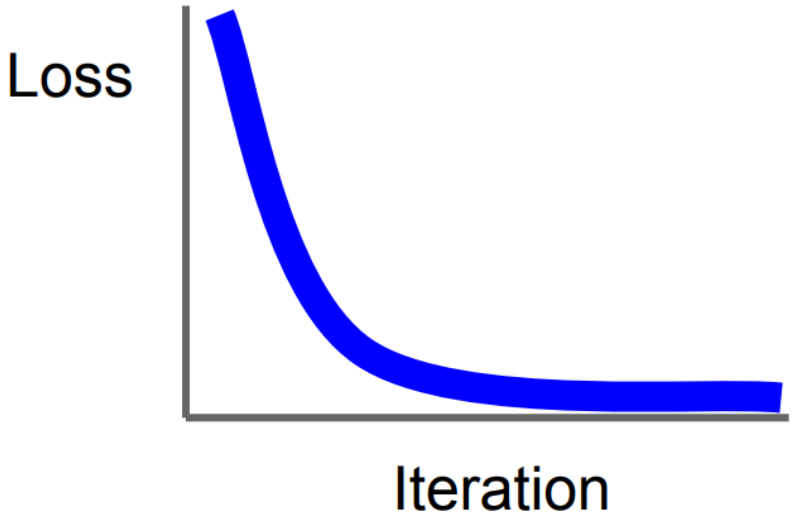
What you can do to help out training error:

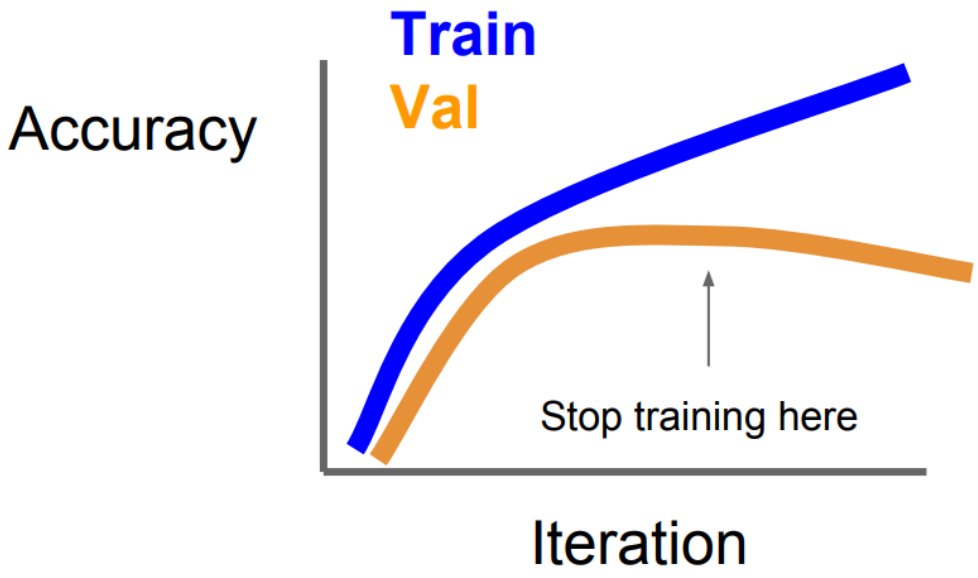
- Optimizer choice
- Optimizer step size

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

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)

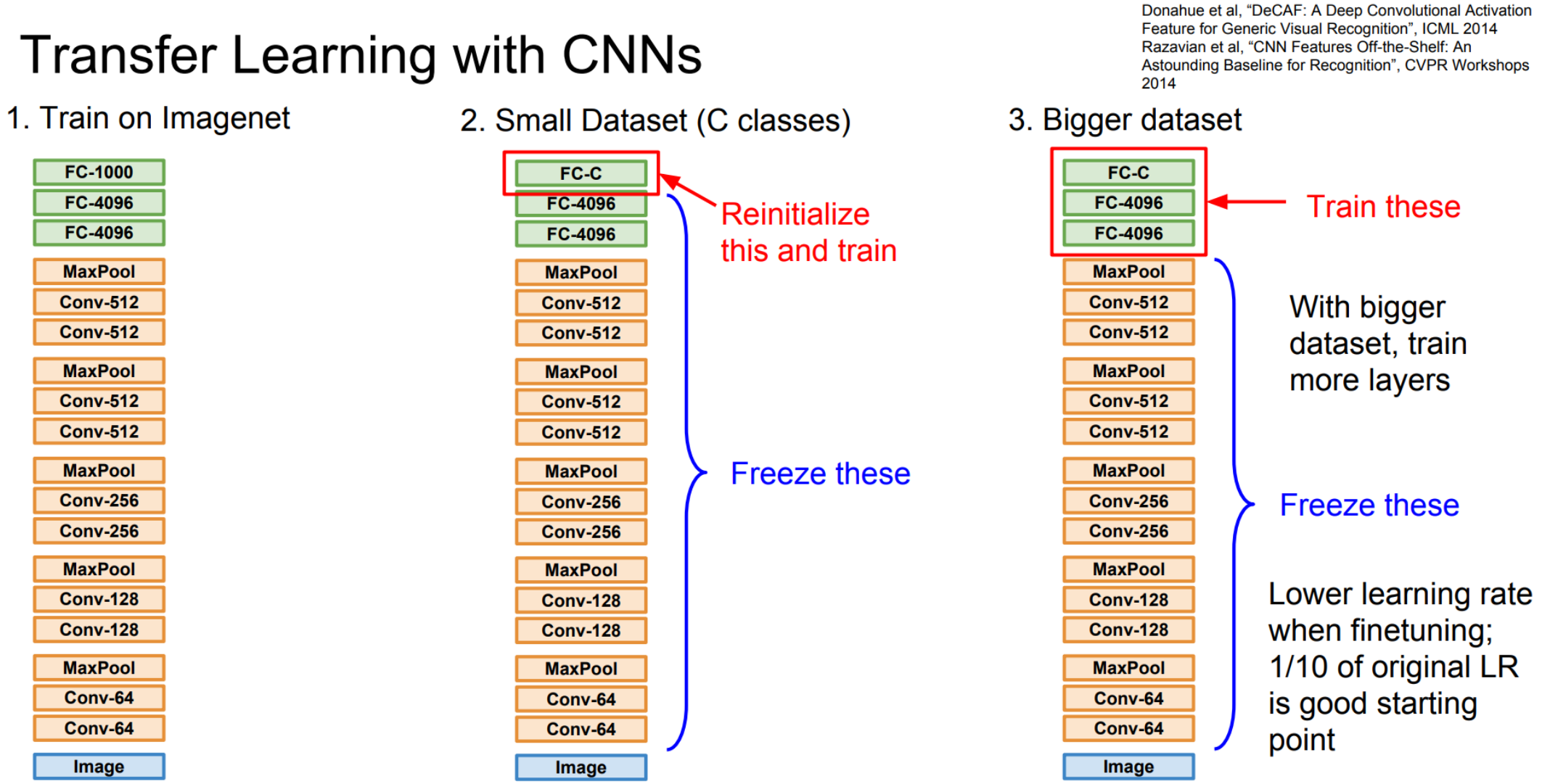
Enjoy 2% extra performance

Related concept/term: majority voting

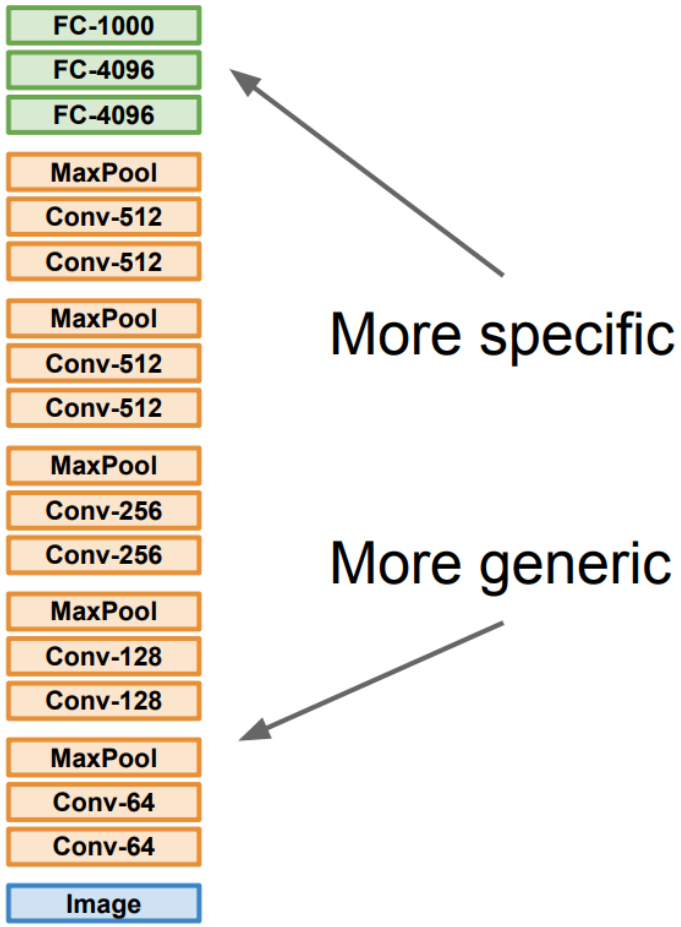
E.g., look at *same* dog image from test data 9X, each w/ uniquely trained model

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

Trick #3: Transfer learning



Trick #3: Transfer learning



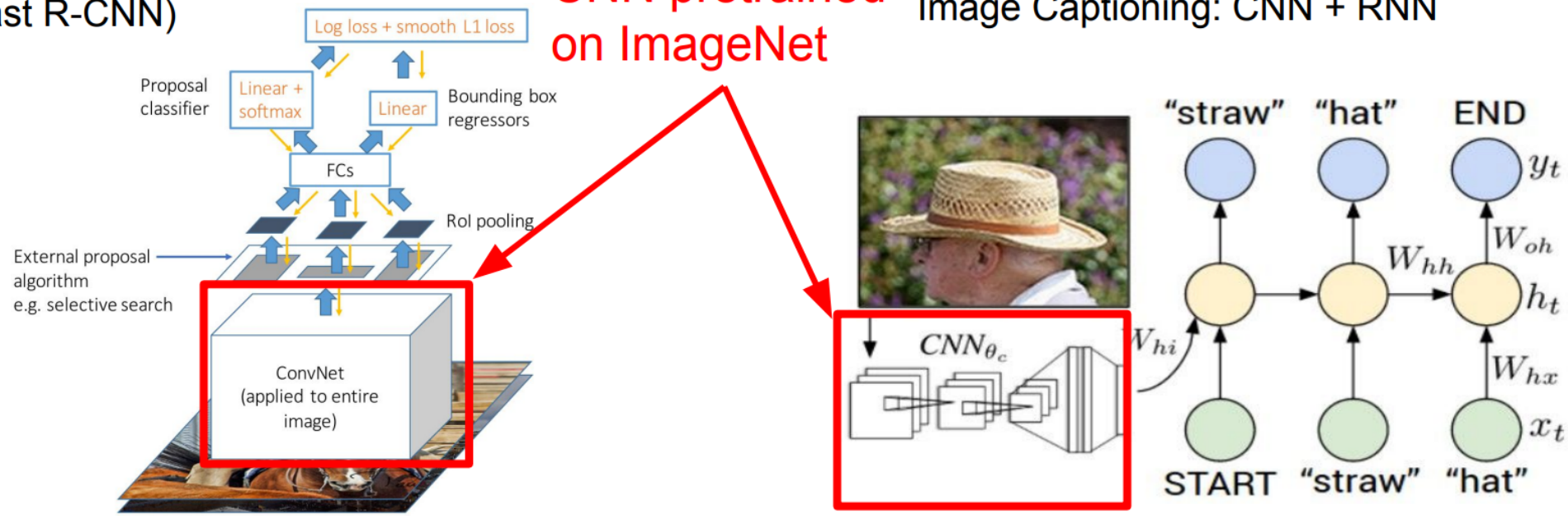
	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

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.

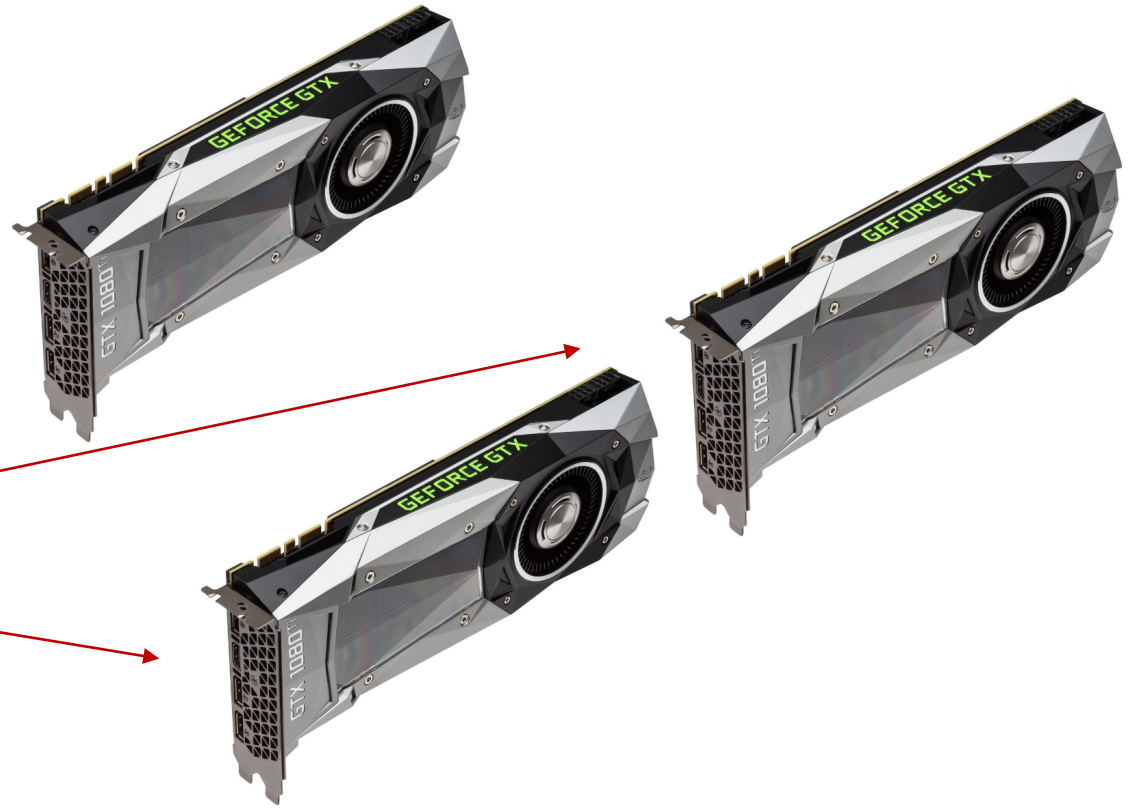
Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

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

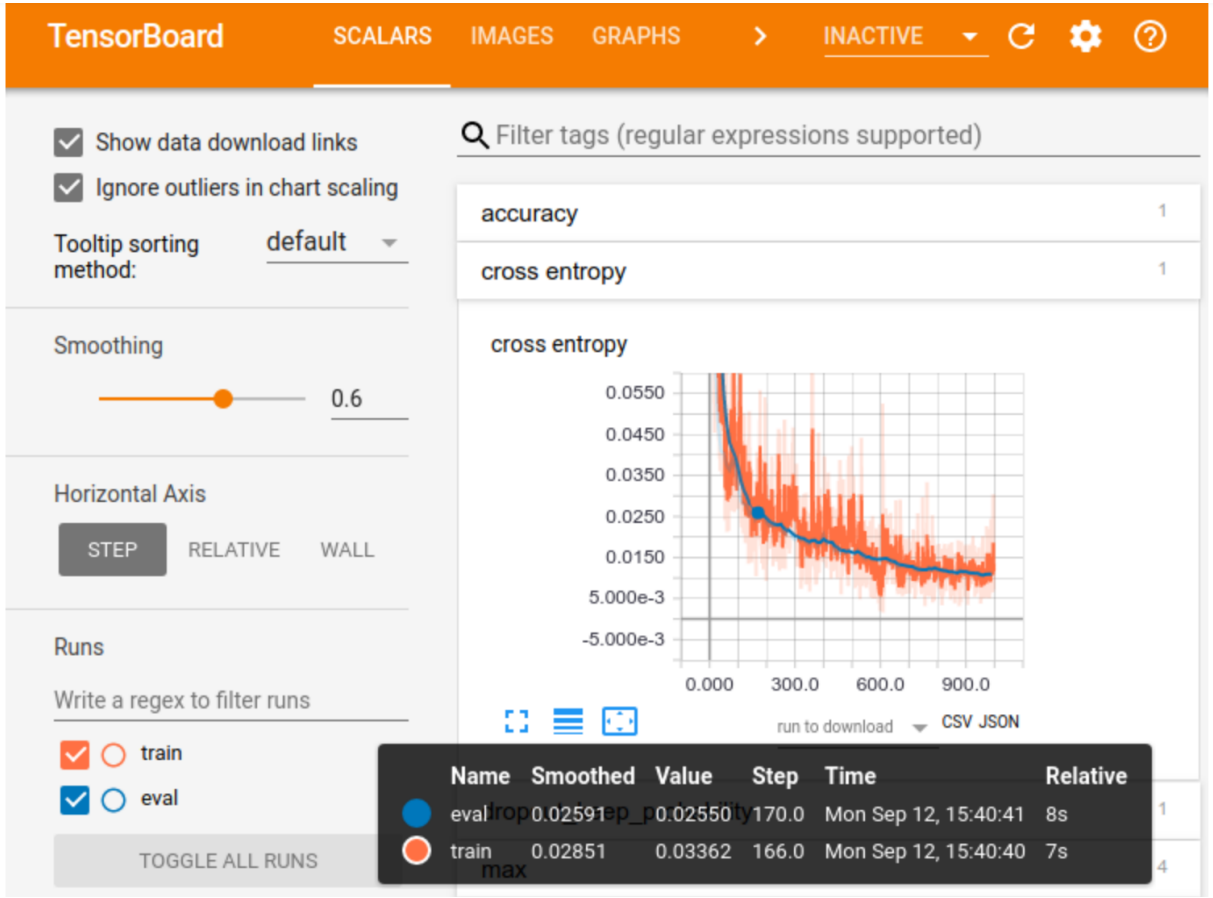
During Training:

- Tensorboard
  - Loss/accuracy versus iteration

After Testing:

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

- Beyond classification:
  - image-to-image similarity
  - segmentation overlap




```
  with tf.name_scope('total'):
    cross_entropy = tf.losses.sparse_softmax_cross_entropy(labels=y_, logits=y)
tf.summary.scalar('cross_entropy', cross_entropy)

with tf.name_scope('train'):
  train_step = tf.train.AdamOptimizer(FLAGS.learning_rate).minimize(
      cross_entropy)

with tf.name_scope('accuracy'):
  with tf.name_scope('correct_prediction'):
    correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
  with tf.name_scope('accuracy'):
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
tf.summary.scalar('accuracy', accuracy)

# Merge all the summaries and write them out to /tmp/mnist_logs (by default)
merged = tf.summary.merge_all()
train_writer = tf.summary.FileWriter(FLAGS.summaries_dir + '/train',
                                     sess.graph)
test_writer = tf.summary.FileWriter(FLAGS.summaries_dir + '/test')
tf.global_variables_initializer().run()
```

In [3]:
```
tensorboard = tf.keras.callbacks.TensorBoard(log_dir='/tmp/logs_1', histogram_freq=1, write_graph=
True, write_images=False)

model.fit(X_train, y_train,
          batch_size=64,
          epochs=5,
          verbose=1,
          validation_data=(X_val, y_val),
          callbacks=[tensorboard])
```

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

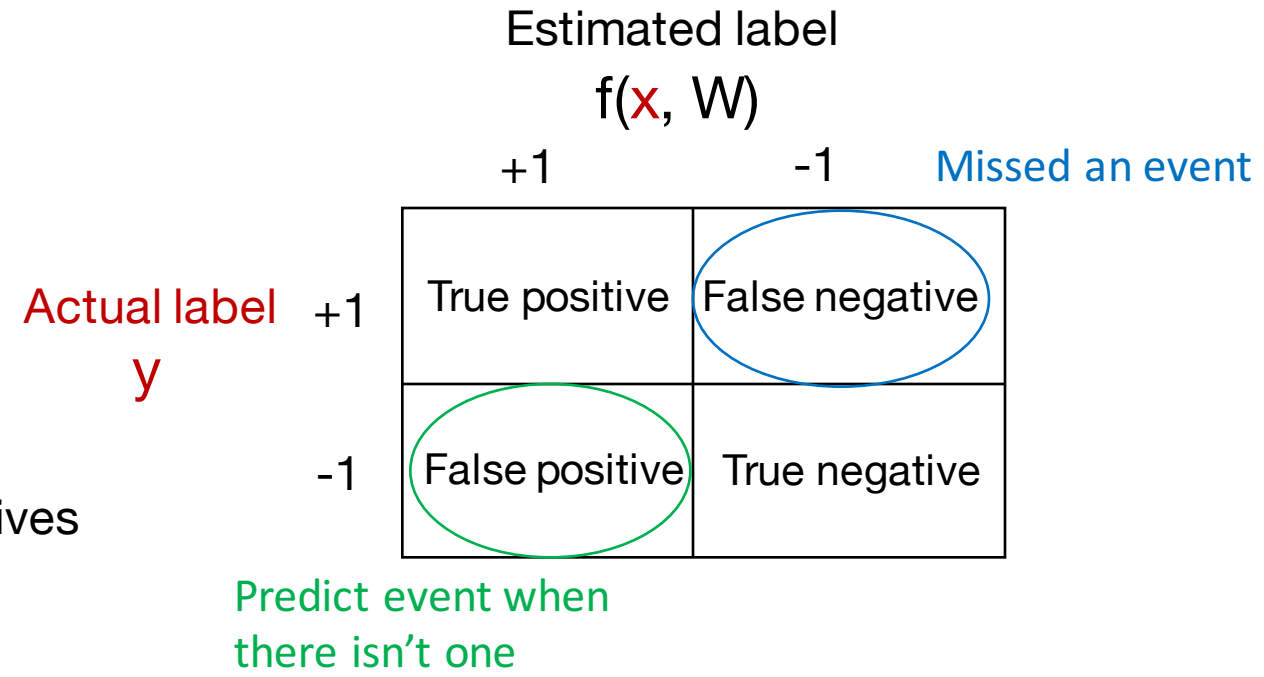
ROC curve and confusion matrix

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



ROC curve and confusion matrix

TP Rate =

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

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

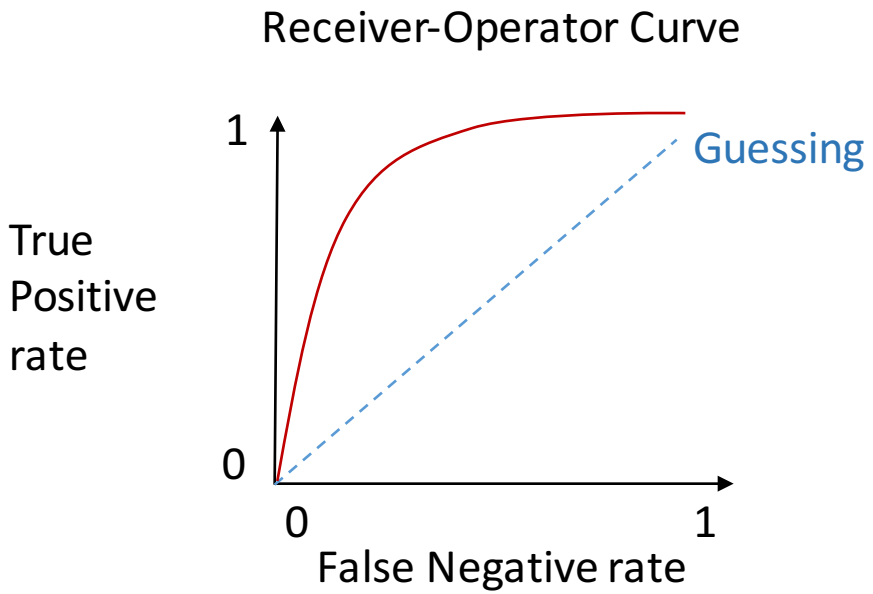
Specificity = TN / (TN + FP) = TN / Actual negatives
= 1 – False Positive Rate

Estimated label
f(x, W)

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



ROC curve and confusion matrix

TP Rate =

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

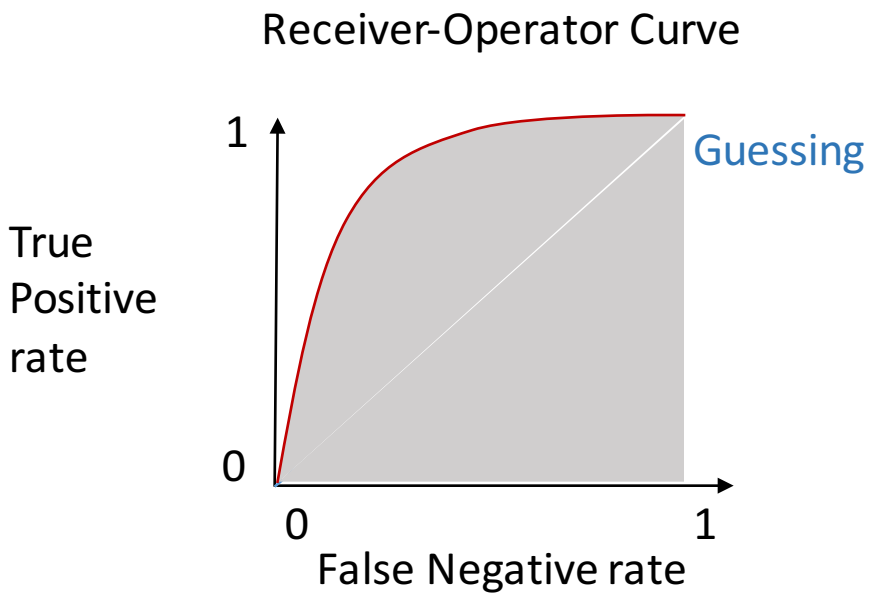
False Positive Rate = FP / (TN + FP) = FP/ Actual negatives

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

Estimated label
f(x, W)

Actual label y	+1	-1 (Missed an event)
+1	True positive	False negative
-1	False positive	True negative

Predict event when there isn't one

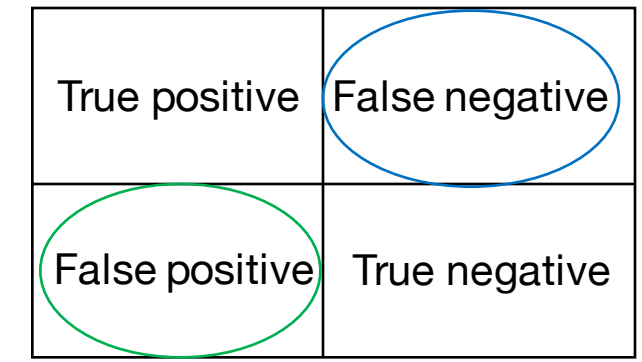


Area under the curve (AUC): Integral of ROC curve

ROC curve and confusion matrix

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

Estimated label
f(x, W)



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

Estimated label
f(x, W)

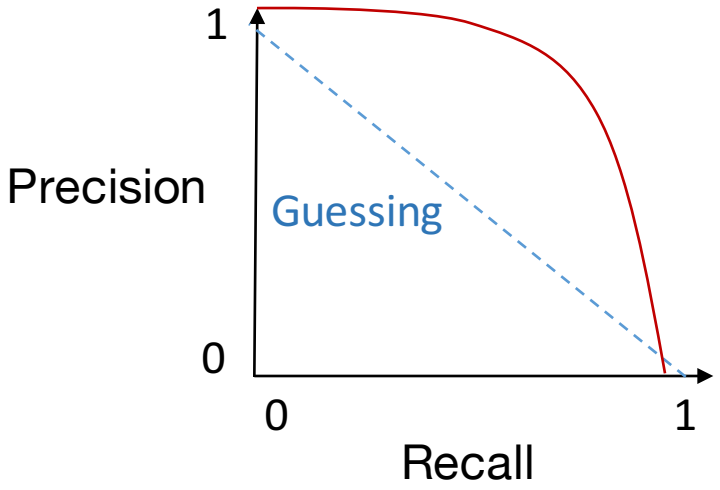
Actual label y	+1	-1
+1	True positive	False negative
-1	False positive	True negative

Missed an event

Predict event when there isn't one

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

Precision-Recall curve



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

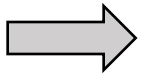
ROC curve and confusion matrix

Just 2 categories

Estimated label

f(x, W)

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

  J = |R1 ∩ R2| / |R1 U R2|

- 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.