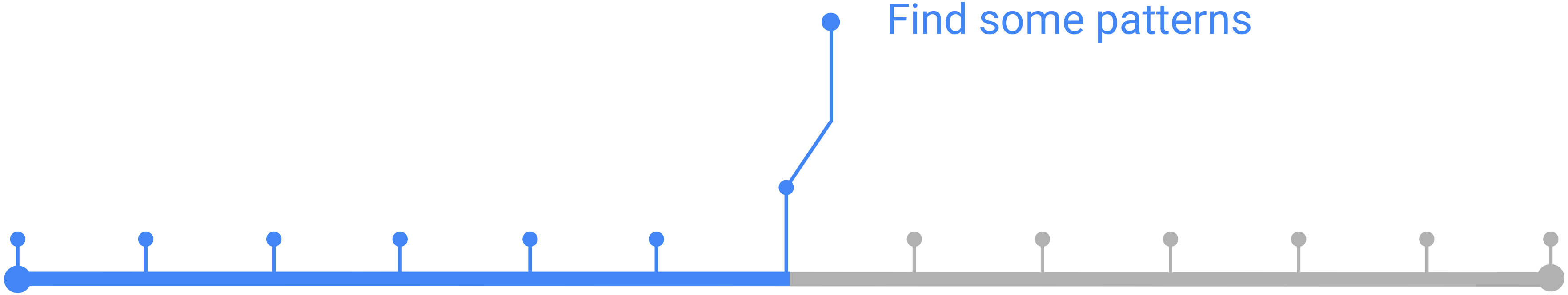
Deep Learning Walkthrough - 06

Code in github.com/google-aai/sc17

Cassie Kozyrkov
Chief Decision Scientist, Google Cloud
GitHub: kozyrkov; Twitter: @quaesita

Google Cloud

Step 6 | Train your models

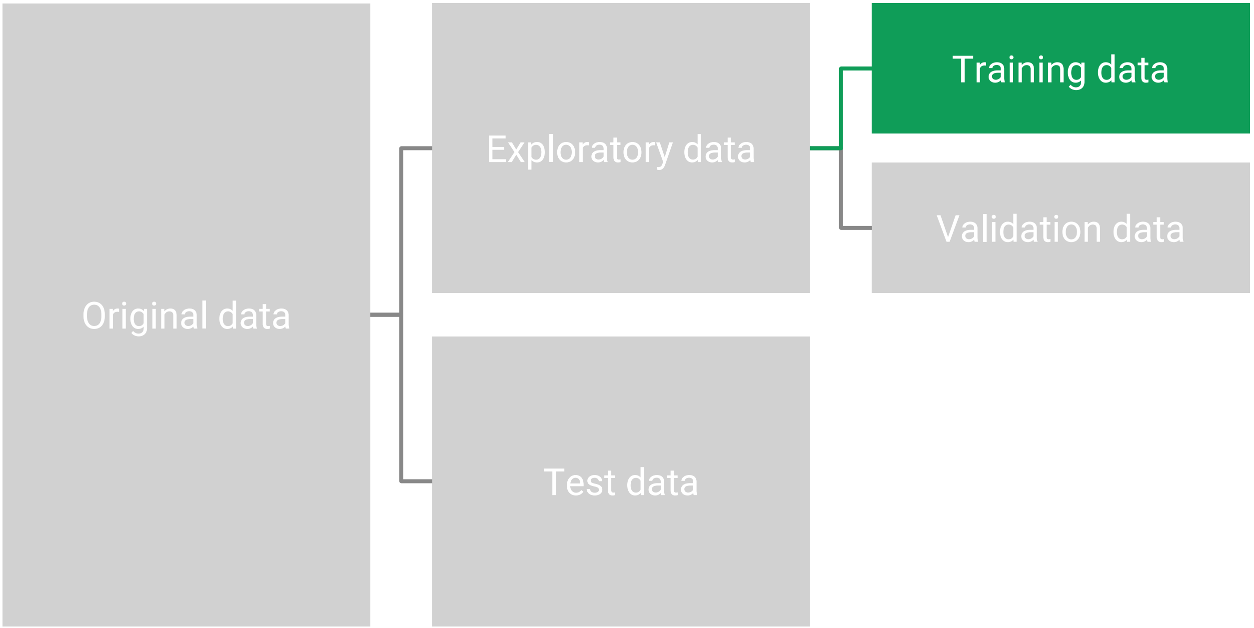


Chief actors:

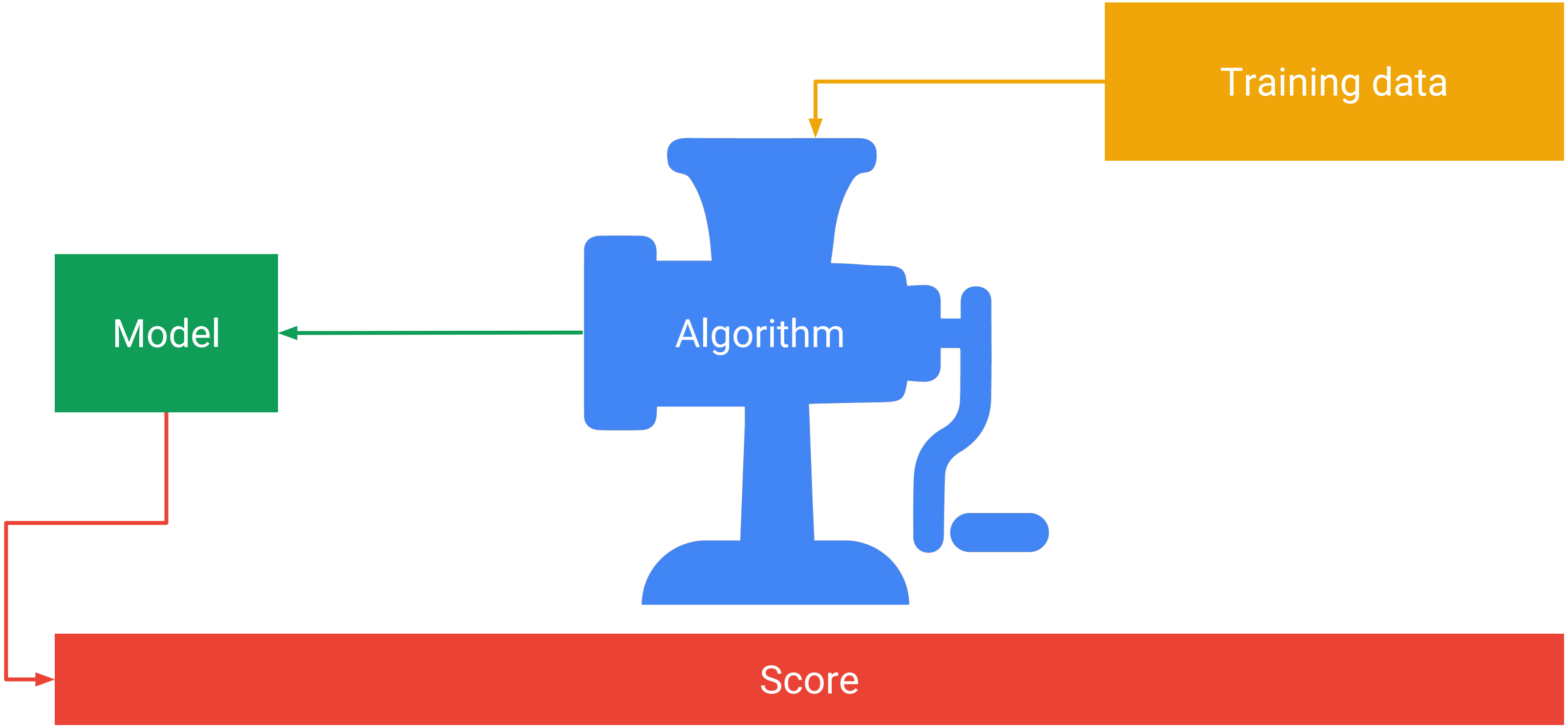
Engineer

Analyst

Use this, not that



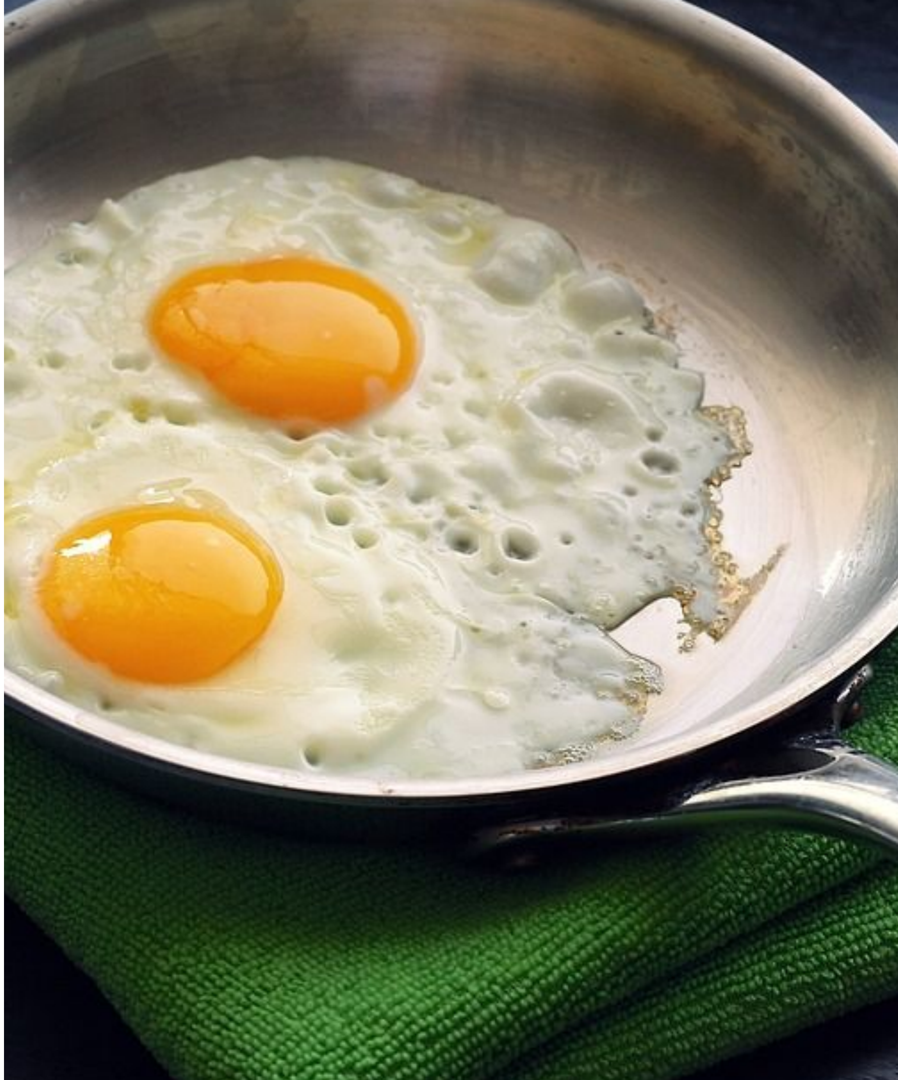
Process



In practice

!

Training is easy, but getting your data into lots of different methods at scale **takes engineering effort**



In practice

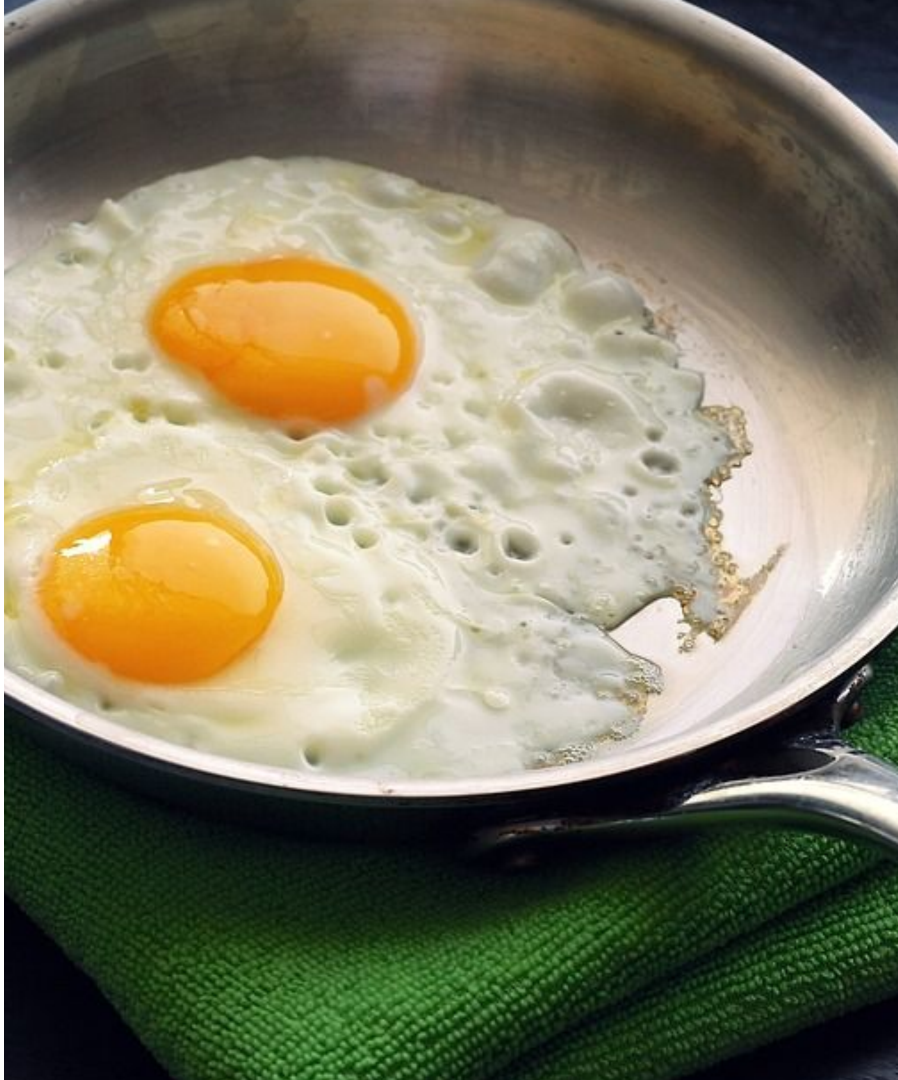
!

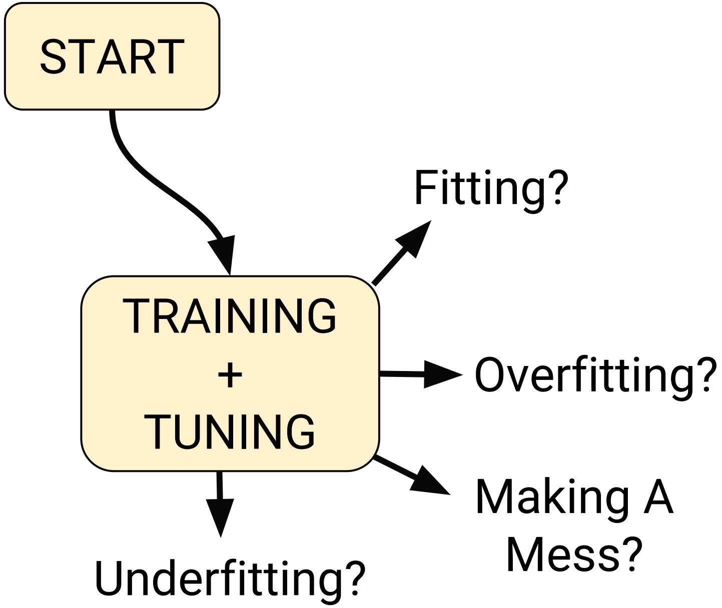
Tinkering can be slow when:

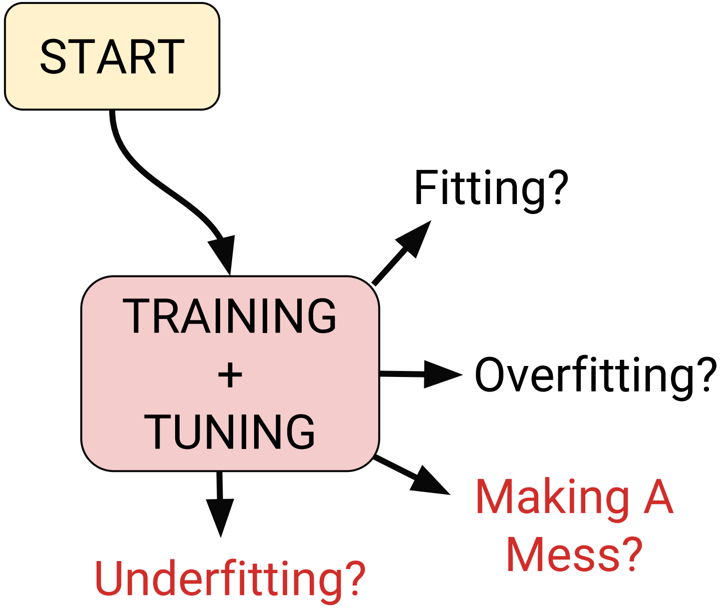
- You have a lot of data
- Software's geared at production

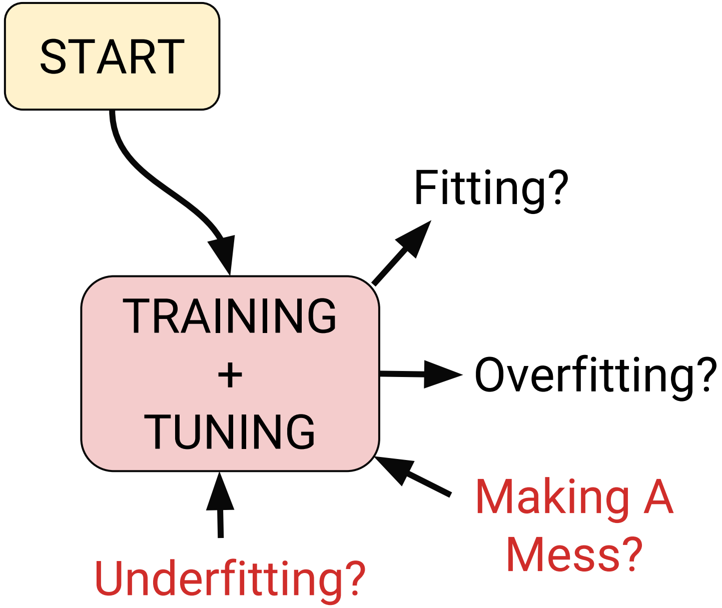
Solutions

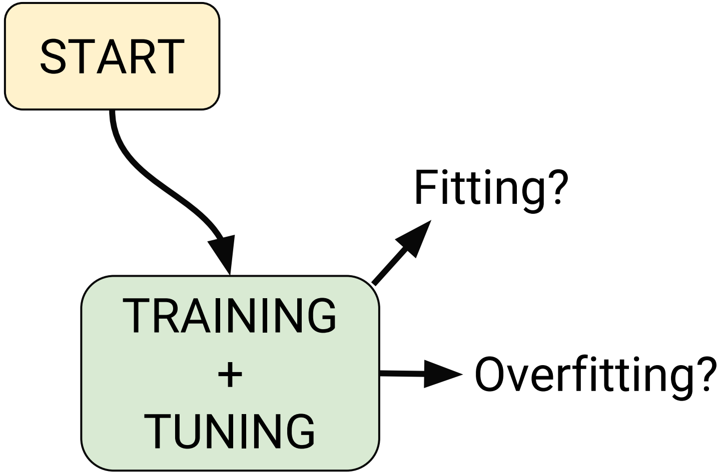
- Use prototyping tools
- Try it with a subset of your training data for inspiration

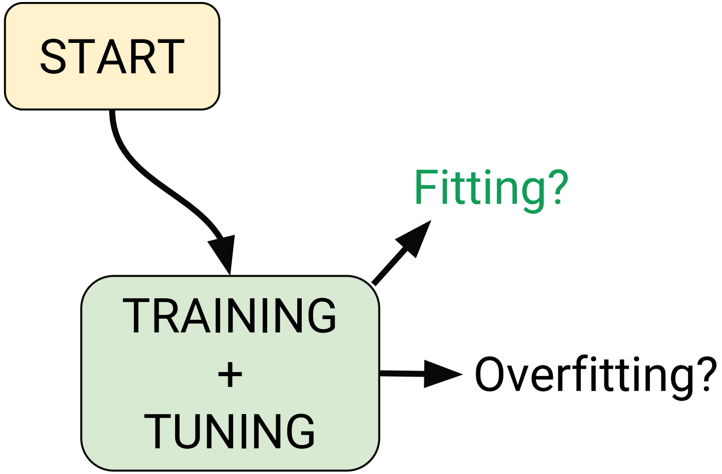


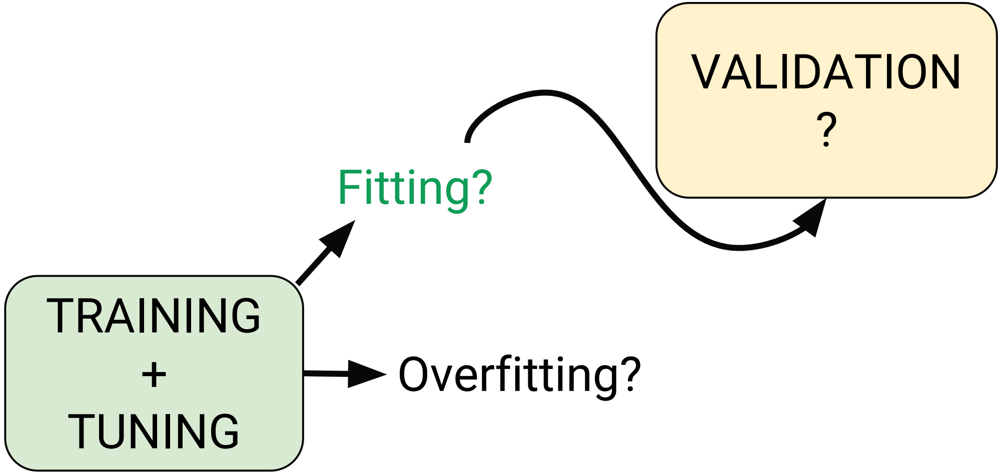

START
TRAINING
+
TUNING
Fitting?
Overfitting?
Making A
Mess?
Underfitting?

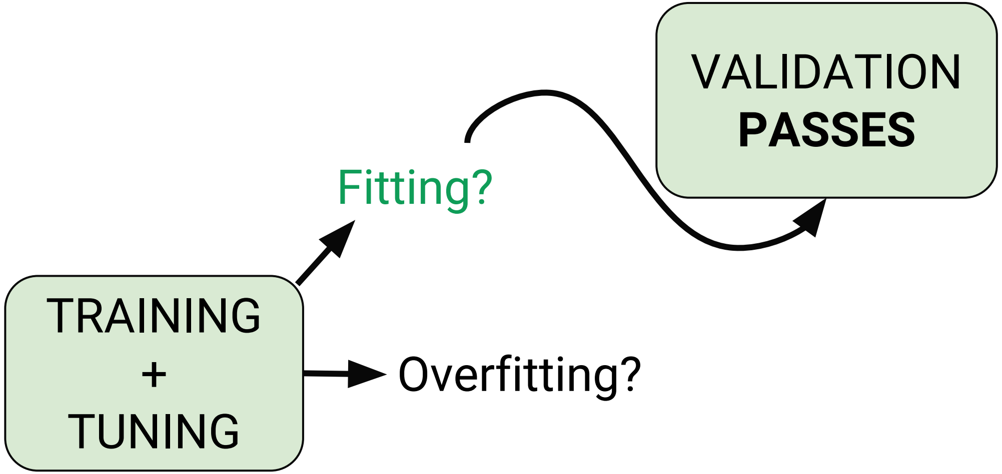

START
Fitting?
TRAINING
+
TUNING
Overfitting?
Making A
Mess?
Underfitting?

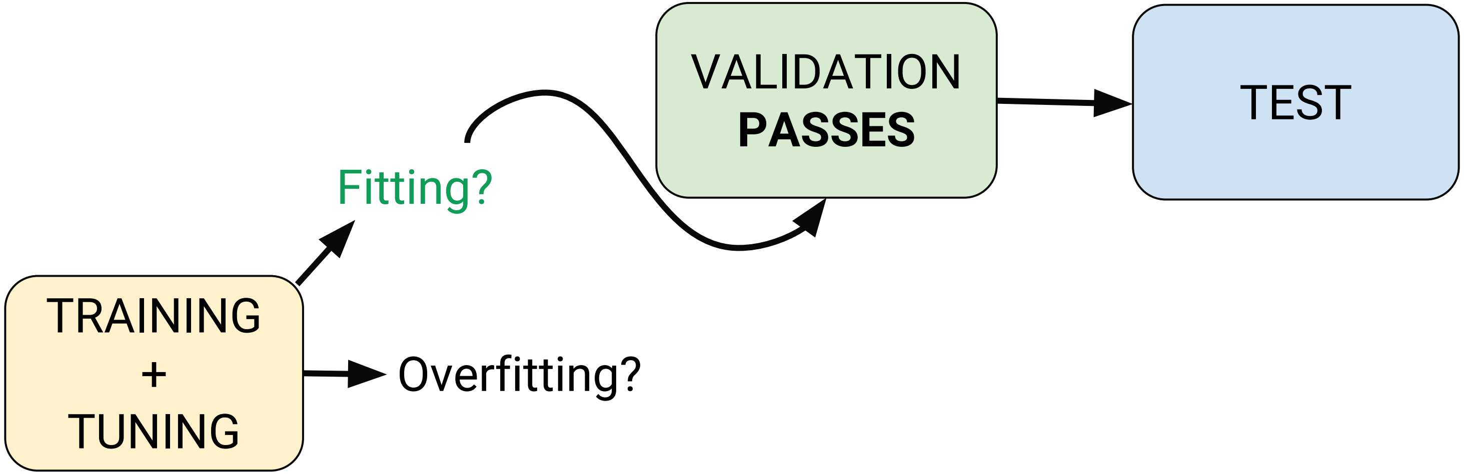

START
Fitting?
TRAINING
+
TUNING
Overfitting?
Making A
Mess?
Underfitting?

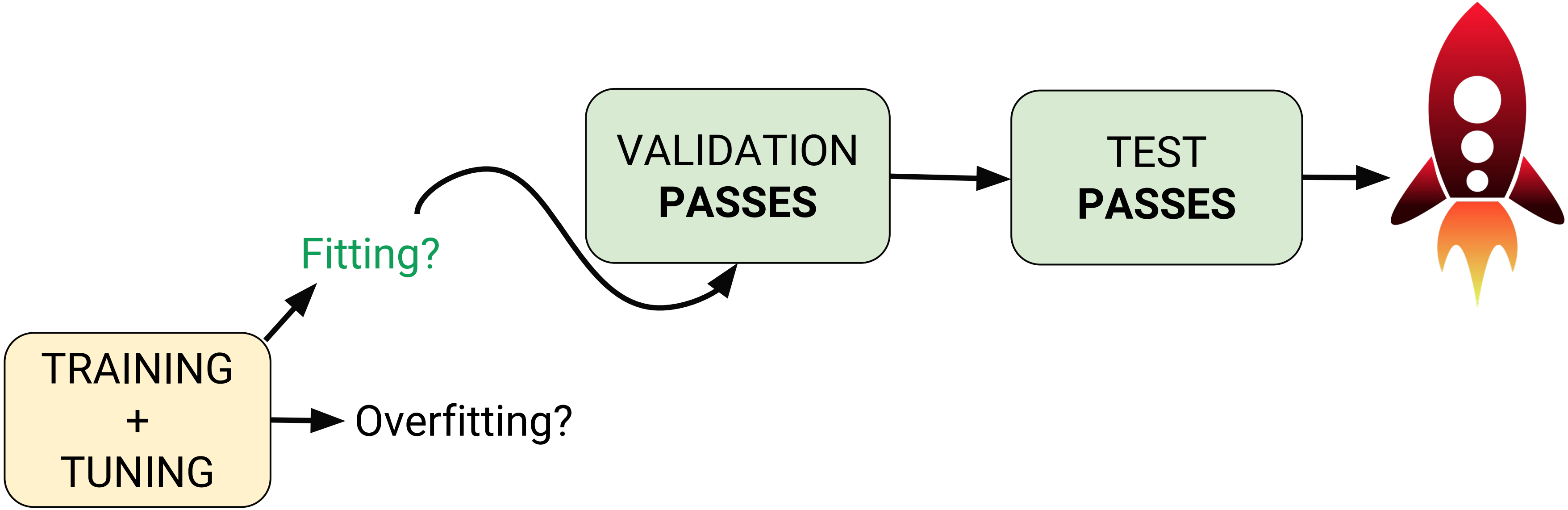

START
TRAINING
+
TUNING
Fitting?
Overfitting?


START
TRAINING
+
TUNING
Fitting?
Overfitting?


VALIDATION
?
Fitting?
TRAINING
+
TUNING
Overfitting?


VALIDATION
PASSES
Fitting?
TRAINING
+
TUNING
Overfitting?


VALIDATION
PASSES
TEST
Fitting?
TRAINING
+
TUNING
Overfitting?

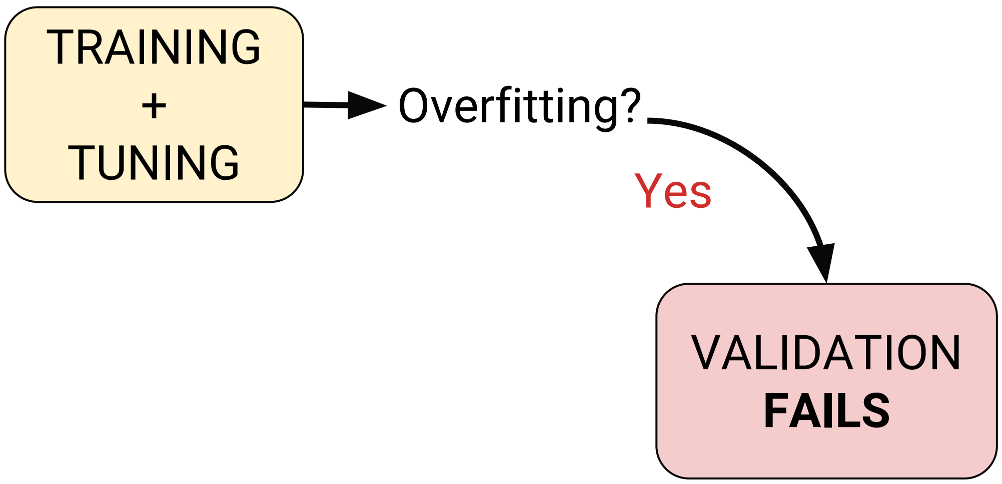

TRAINING
+
TUNING
Fitting?
Overfitting?
VALIDATION
PASSES
TEST
PASSES

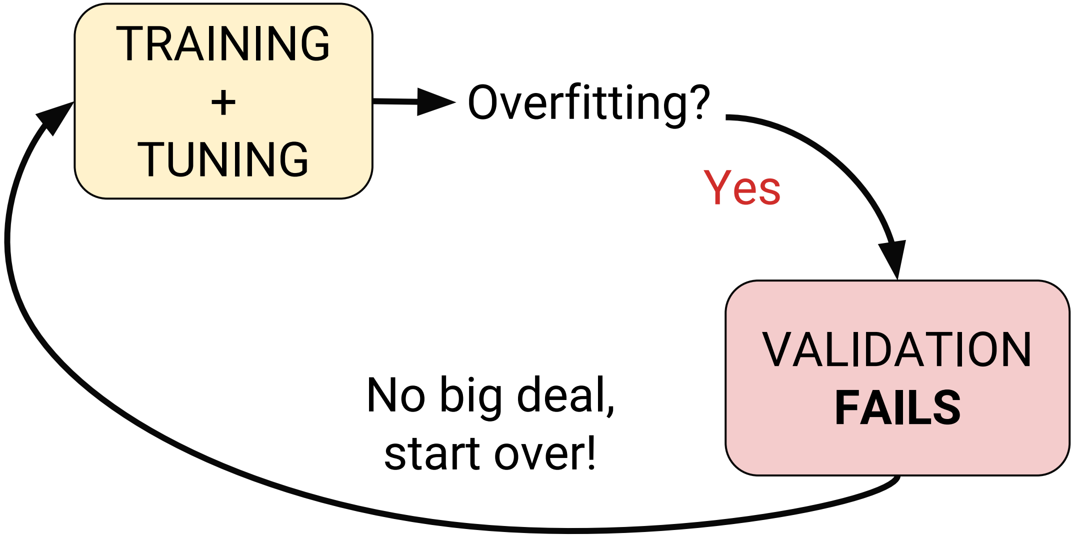
Fitting?

TRAINING
+
TUNING

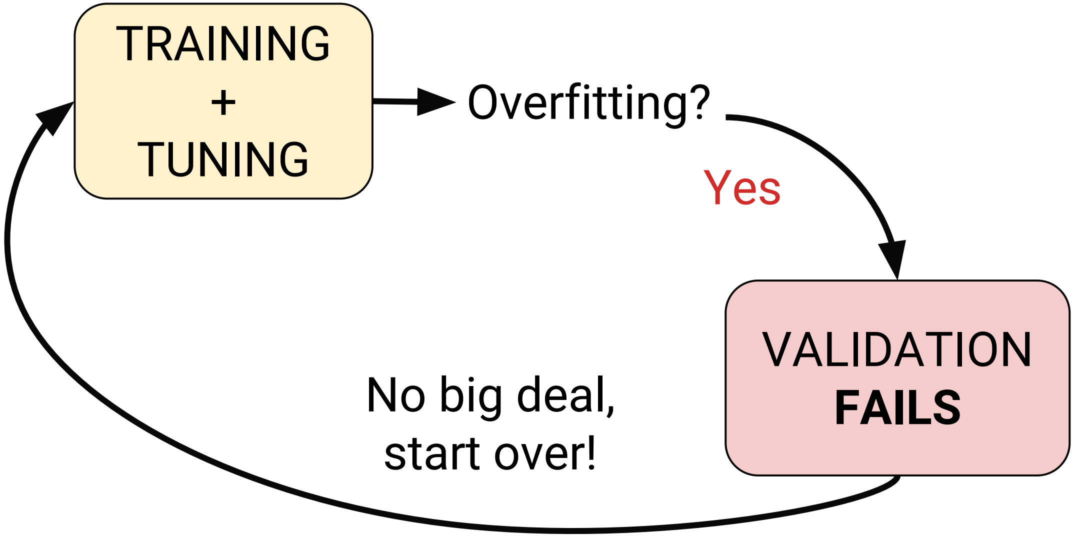
Overfitting?

Yes


TRAINING
+
TUNING
Overfitting?
Yes
VALIDATION
FAILS

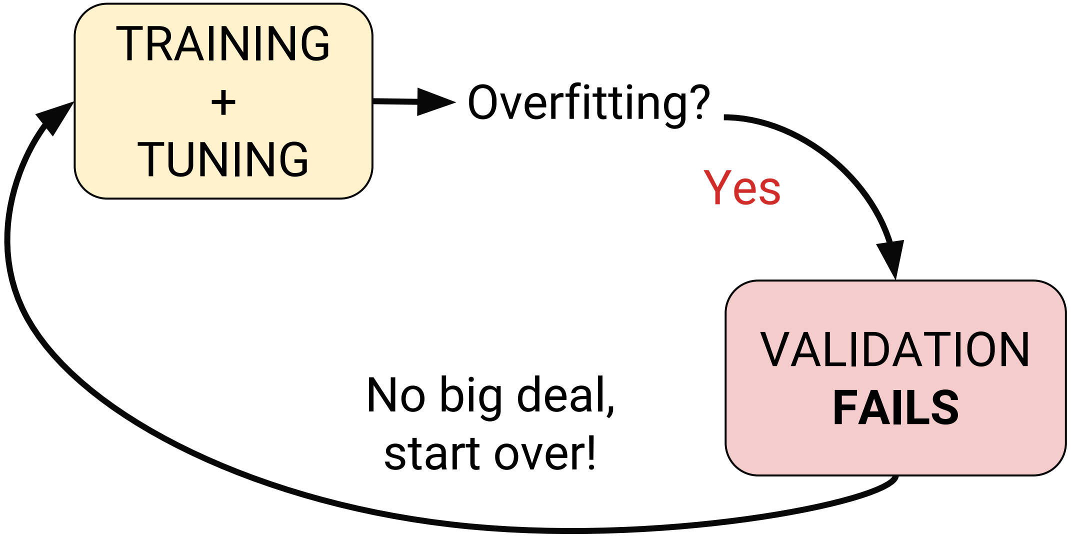

TRAINING
+
TUNING
Overfitting?
Yes
VALIDATION
FAILS
No big deal,
start over!

It’s the dreaded infinite loop!!!



Welcome to overfitting limbo

It’s the dreaded infinite loop!!!



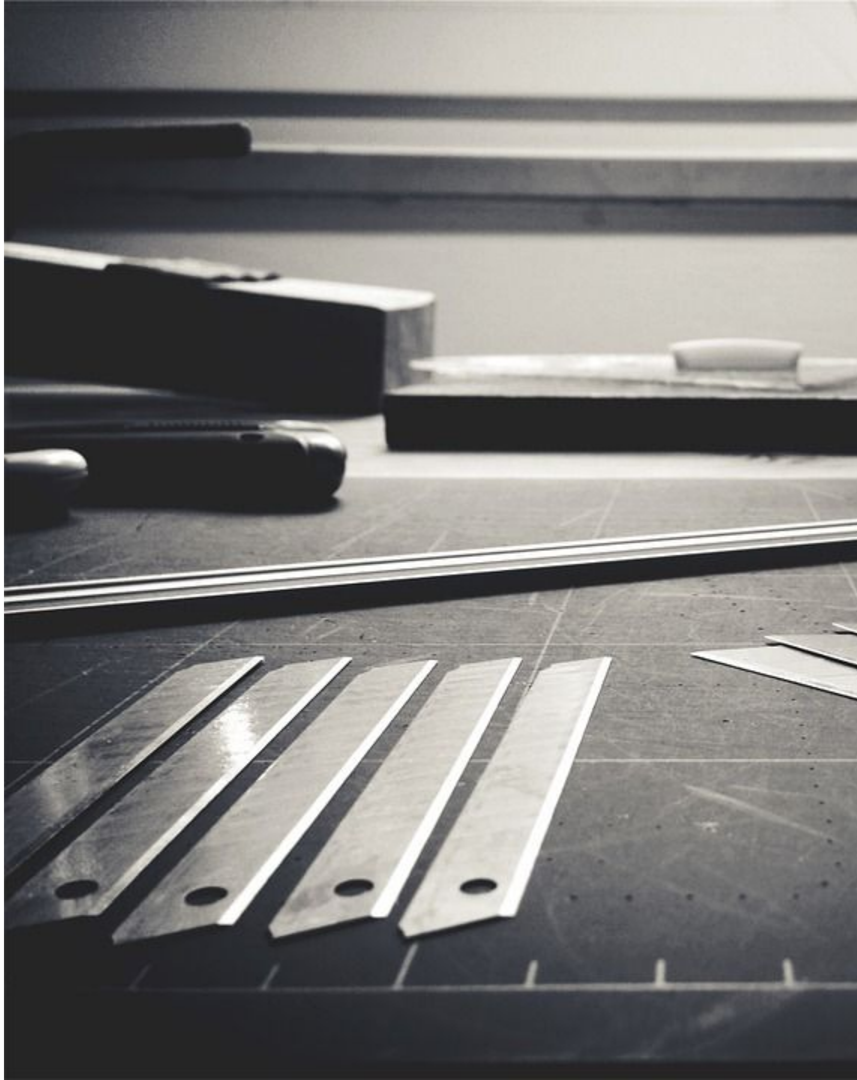
STILL WAITING...

TO PASS VALIDATION

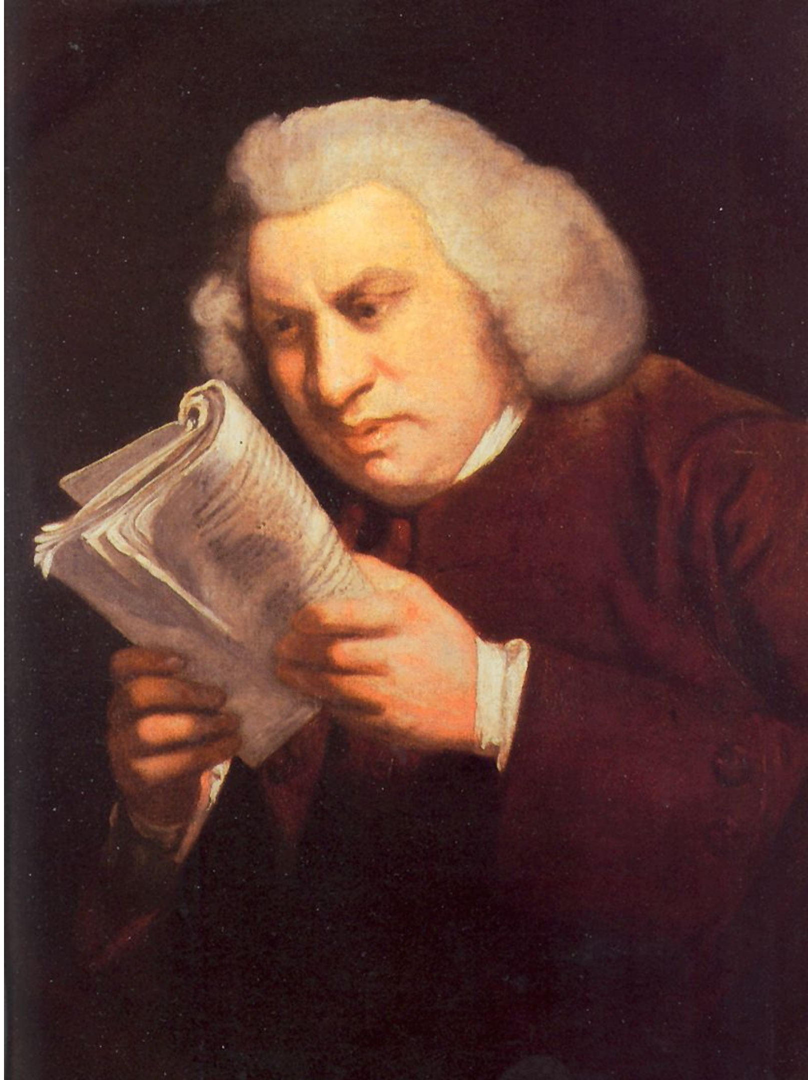
Solution? Start Simple!

Beware excessive complexity.

Model complexity increases the chances of overfitting.



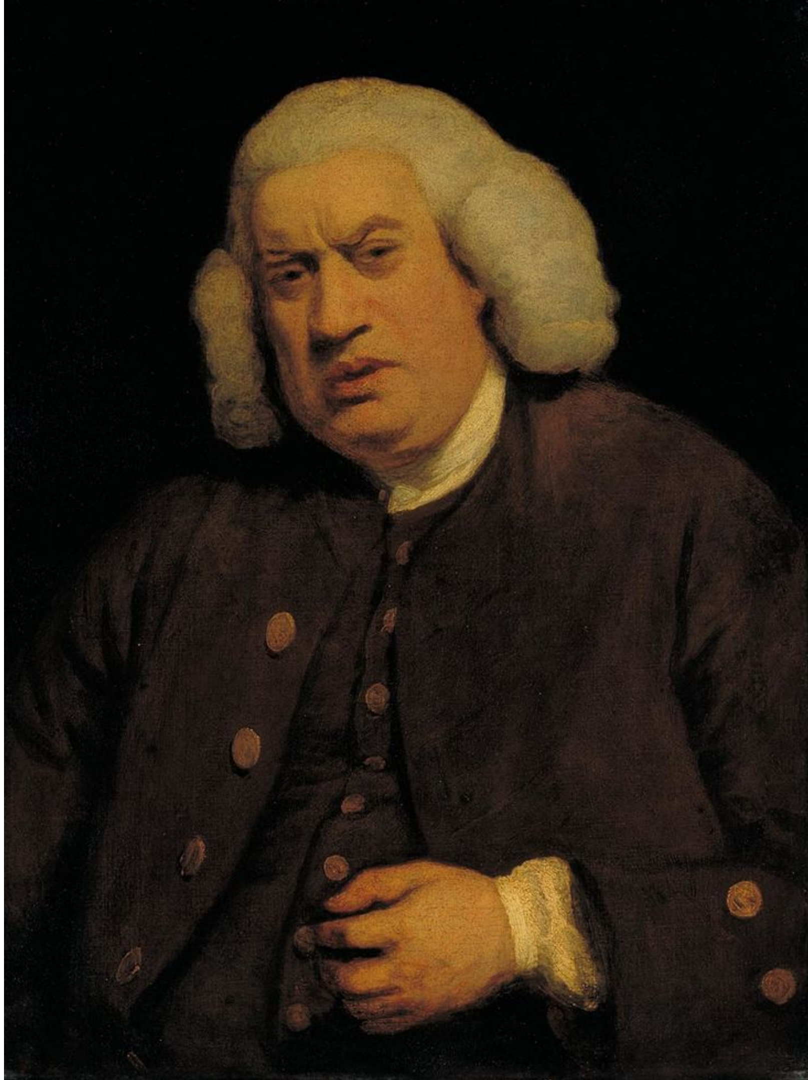
Regularization?



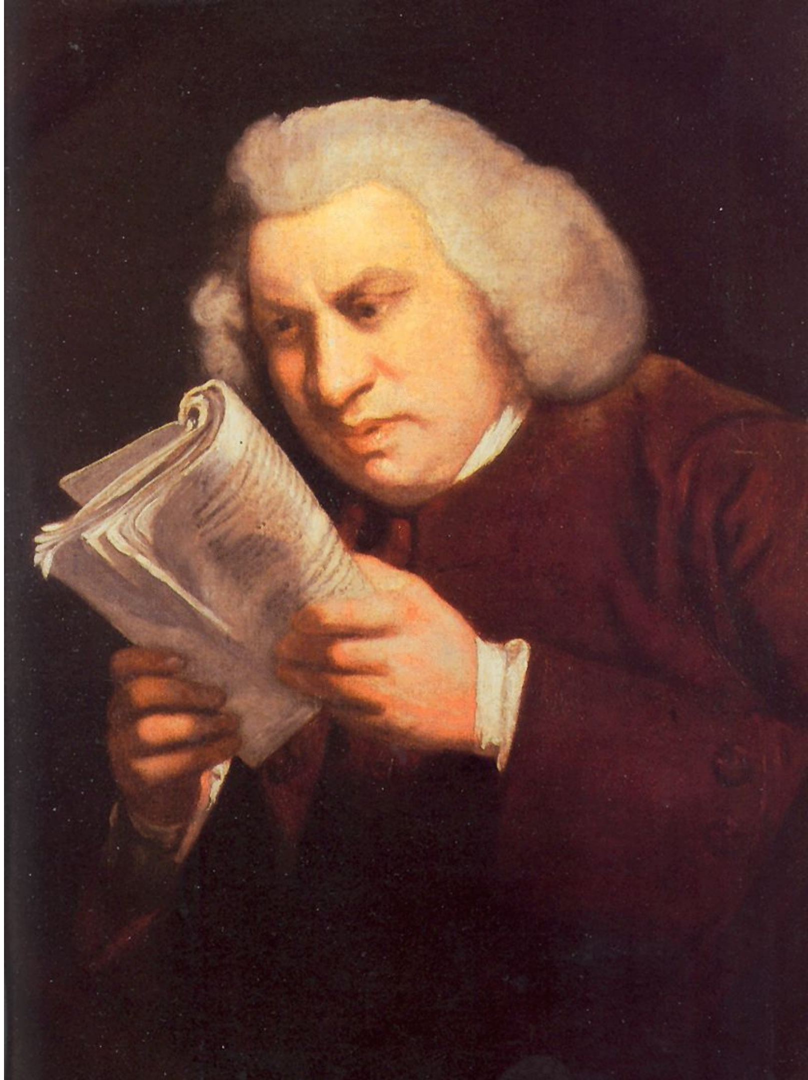
Regularization

“Learn to love simplicity.”

Complexity increases loss.



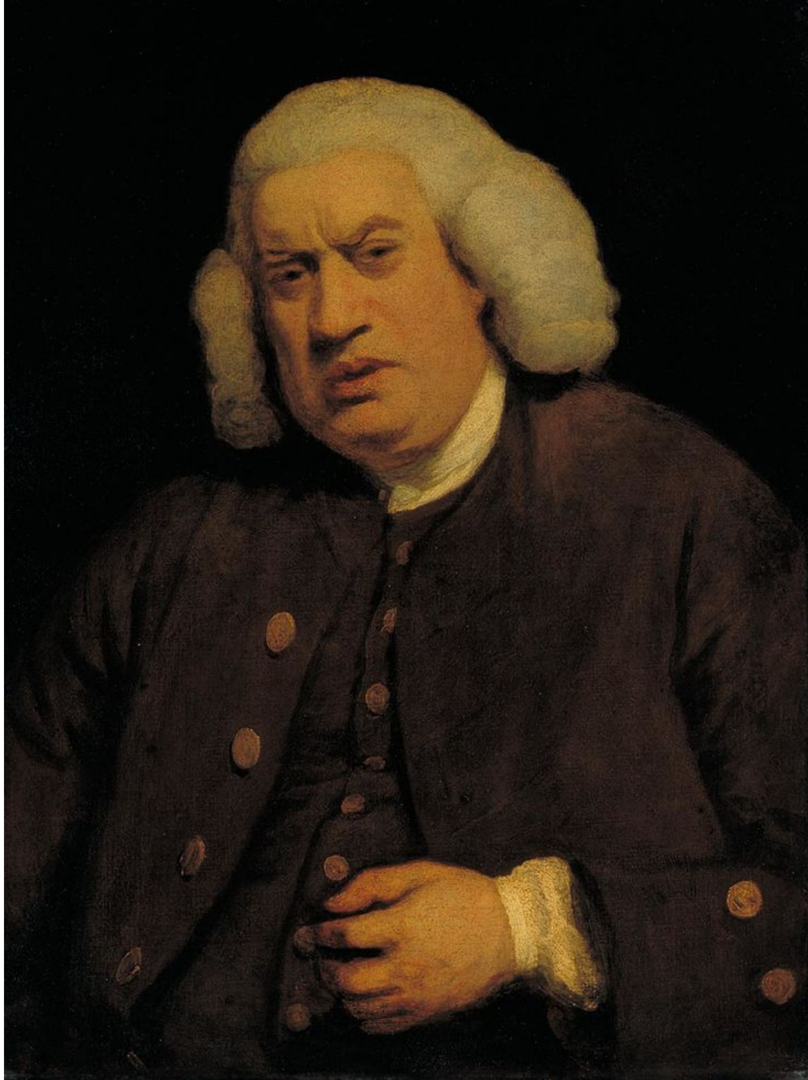
Dropout?



Dropout

"Make a neural network love simplicity by shooting some neurons."

Don't train with all of the units.



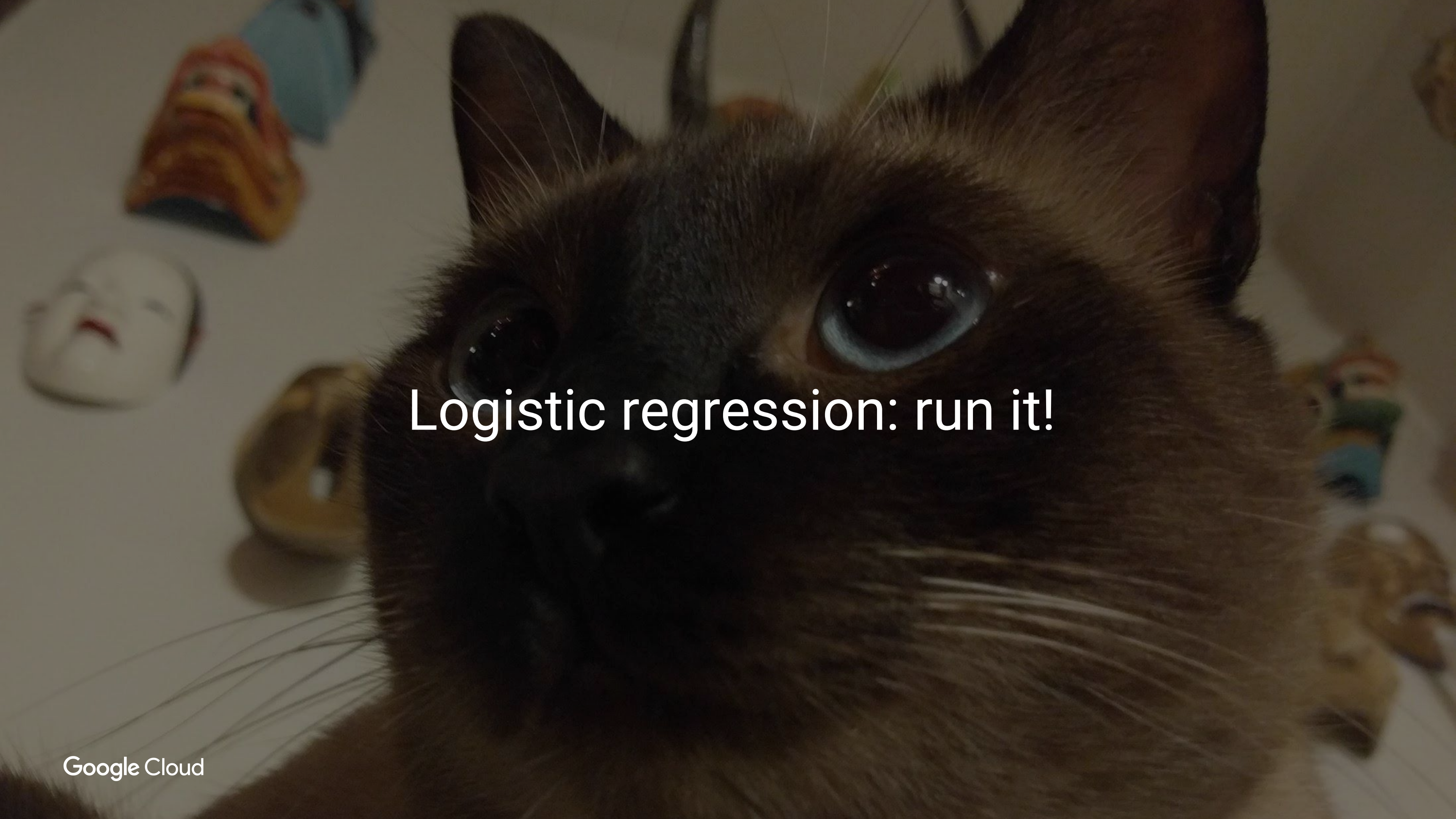
Key message

!

Start simple and build up the complexity only when simple doesn't work well. Use regularization.

You final solution might be sophisticated, but your first attempt shouldn't be.

Logistic regression: run it!



Step 6: Train Logistic Regression with scikit-learn

Let's train!

In [8]:

```
# Get the output predictions of the training and debugging inputs
training_predictions = model.predict_proba(training_std)[:, 1]
debugging_predictions = model.predict_proba(debugging_std)[:, 1]
```

That was easy! But how well did it do? Let's check the accuracy of the model we just trained.

That was easy! But how well did it do? Let's check the accuracy of the model we just trained.

In [9]:
```
# Accuracy metric:
def get_accuracy(truth, predictions, threshold=0.5, roundoff=2):
  """
  Args:
    truth: can be Boolean (False, True), int (0, 1), or float (0, 1)
    predictions: number between 0 and 1, inclusive
    threshold: we convert predictions to 1s if they're above this value
    roundoff: report accuracy to how many decimal places?

  Returns:
    accuracy: number correct divided by total predictions
  """

  truth = np.array(truth) == (1|True)
  predicted = np.array(predictions) >= threshold
  matches = sum(predicted == truth)
  accuracy = float(matches) / len(truth)
  return round(accuracy, roundoff)

# Compute our accuracy metric for training and debugging
print 'Training accuracy is ' + str(get_accuracy(training_labels, training_predictions))
print 'Debugging accuracy is ' + str(get_accuracy(debugging_labels, debugging_predictions))
```

```
Training accuracy is 0.76
Debugging accuracy is 0.74
```

Step 6: Train Logistic Regression with TensorFlow

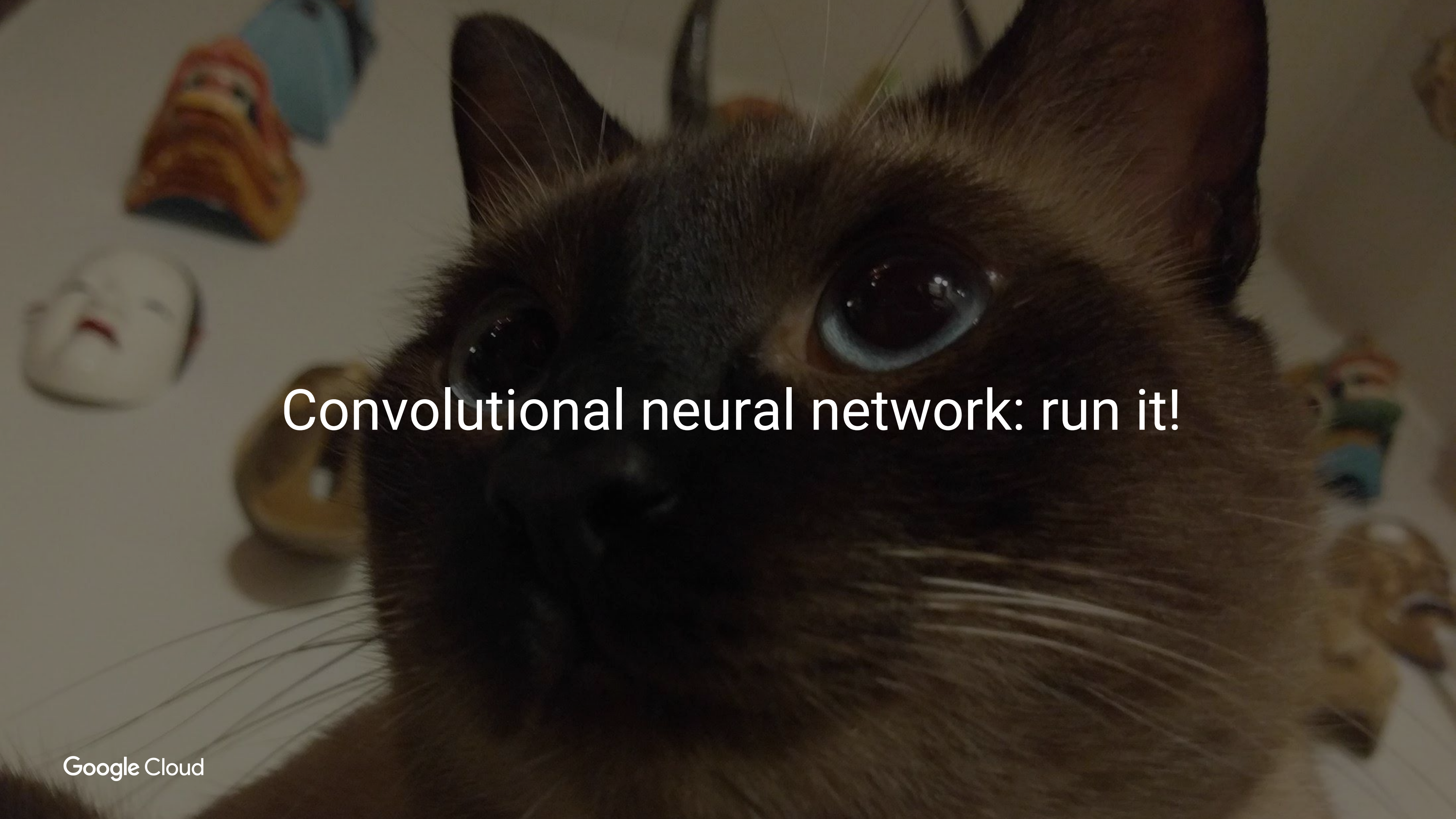
Unless you change TensorFlow's verbosity, there is a lot of text that is outputted. Such text can be useful when debugging a distributed training pipeline, but is pretty noisy when running from a notebook locally. The line to look for is the chunk at the end where "accuracy" is reported. This is the final result of the model.

In [13]:
```
learn_runner.run(generate_experiment_fn(), OUTPUT_DIR + '-model-reg-' + str(REG_L1))
```

```
INFO:tensorflow:Restoring parameters from ../../kozyrkov/data/logreg/-model-reg-5.0/model.ckpt-20000
INFO:tensorflow:Evaluation [1/1]
INFO:tensorflow:Finished evaluation at 2018-01-03-21:14:58
INFO:tensorflow:Saving dict for global step 20000: accuracy = 0.734903, accuracy/baseline_label_mean = 0.474923, accu
racy/threshold_0.500000_mean = 0.734903, auc = 0.812086, auc_precision_recall = 0.785532, global_step = 20000, label
s/actual_label_mean = 0.474923, labels/prediction_mean = 0.480623, loss = 0.541839, precision/positive_threshold_0.50
0000_mean = 0.705411, recall/positive_threshold_0.500000_mean = 0.758621
```

Out[13]:
```
({'accuracy': 0.73490274,
  'accuracy/baseline_label_mean': 0.47492322,
  'accuracy/threshold_0.500000_mean': 0.73490274,
  'auc': 0.81208611,
  'auc_precision_recall': 0.78553176,
  'global_step': 20000,
  'labels/actual_label_mean': 0.47492322,
  'labels/prediction_mean': 0.48062292,
  'loss': 0.54183859,
  'precision/positive_threshold_0.500000_mean': 0.70541084,
  'recall/positive_threshold_0.500000_mean': 0.75862068},
 [])
```

Convolutional neural network: run it!



Step 6 - Train a model!

Let's run our lovely creation on our training data. In order to train, we need learn_runner(), which we imported from TensorFlow above. For prediction, we will only need estimator.predict().

In [12]:

```python
# Enable TF verbose output:
tf.logging.set_verbosity(tf.logging.INFO)
start_time = datetime.datetime.now()
print('It\'s {:%H:%M} in London'.format(start_time) + ' --- Let\'s get started!')
# Let the learning commence!  Run the TF Experiment here.
learn_runner.run(generate_experiment_fn(), OUTPUT_DIR)
# Output lines using the word "Validation" are giving our metric on the non-training dataset (from DEBUG_DIR).
end_time = datetime.datetime.now()
print('\nIt was {:%H:%M} in London when we started.'.format(start_time))
print('\nWe\'re finished and it\'s {:%H:%M} in London'.format(end_time))
print('\nCongratulations!  Training is complete!')
```

```
It's 18:54 in London --- Let's get started!
WARNING:tensorflow:From <ipython-input-11-10bb144e4fb2>:18: calling __init__ (from tensorflow.contrib.learn.python.le
arn.experiment) with local_eval_frequency is deprecated and will be removed after 2016-10-23.
Instructions for updating:
local_eval_frequency is deprecated as local_run will be renamed to train_and_evaluate. Use min_eval_frequency and cal
l train_and_evaluate instead. Note, however, that the default for min_eval_frequency is 1, meaning models will be eva
luated every time a new checkpoint is available. In contrast, the default for local_eval_frequency is None, resulting
in evaluation occurring only after training has completed. min_eval_frequency is ignored when calling the deprecated
 local_run.
WARNING:tensorflow:From /home/user/env/local/lib/python2.7/site-packages/tensorflow/contrib/learn/python/learn/mo
nitors.py:269: __init__ (from tensorflow.contrib.learn.python.learn.monitors) is deprecated and will be removed after
2016-12-05.
Instructions for updating:
Monitors are deprecated. Please use tf.train.SessionRunHook.
INFO:tensorflow:Skipping training since max_steps has already saved.
INFO:tensorflow:Starting evaluation at 2017-11-02-18:54:23
```

```
print('\nIt was {:%H:%M} in London when we started.'.format(start_time))
print('\nWe\'re finished and it\'s {:%H:%M} in London'.format(end_time))
print('\nCongratulations!  Training is complete!')
```

```
It's 18:54 in London --- Let's get started!
WARNING:tensorflow:From <ipython-input-11-10bb144e4fb2>:18: calling __init__ (from tensorflow.contrib.learn.python.le
arn.experiment) with local_eval_frequency is deprecated and will be removed after 2016-10-23.
Instructions for updating:
local_eval_frequency is deprecated as local_run will be renamed to train_and_evaluate. Use min_eval_frequency and cal
l train_and_evaluate instead. Note, however, that the default for min_eval_frequency is 1, meaning models will be eva
luated every time a new checkpoint is available. In contrast, the default for local_eval_frequency is None, resulting
in evaluation occurring only after training has completed. min_eval_frequency is ignored when calling the deprecated
 local_run.
WARNING:tensorflow:From /home/user/env/local/lib/python2.7/site-packages/tensorflow/contrib/learn/python/learn/mo
nitors.py:269: __init__ (from tensorflow.contrib.learn.python.learn.monitors) is deprecated and will be removed after
2016-12-05.
Instructions for updating:
Monitors are deprecated. Please use tf.train.SessionRunHook.
INFO:tensorflow:Skipping training since max_steps has already saved.
INFO:tensorflow:Starting evaluation at 2017-11-02-18:54:23
INFO:tensorflow:Restoring parameters from ../../data/output/model.ckpt-100
INFO:tensorflow:Evaluation [1/2]
INFO:tensorflow:Evaluation [2/2]
INFO:tensorflow:Finished evaluation at 2017-11-02-18:54:24
INFO:tensorflow:Saving dict for global step 100: accuracy = 0.7, global_step = 100, loss = 0.466835

It was 18:54 in London when we started.

We're finished and it's 18:54 in London

Congratulations!  Training is complete!
```

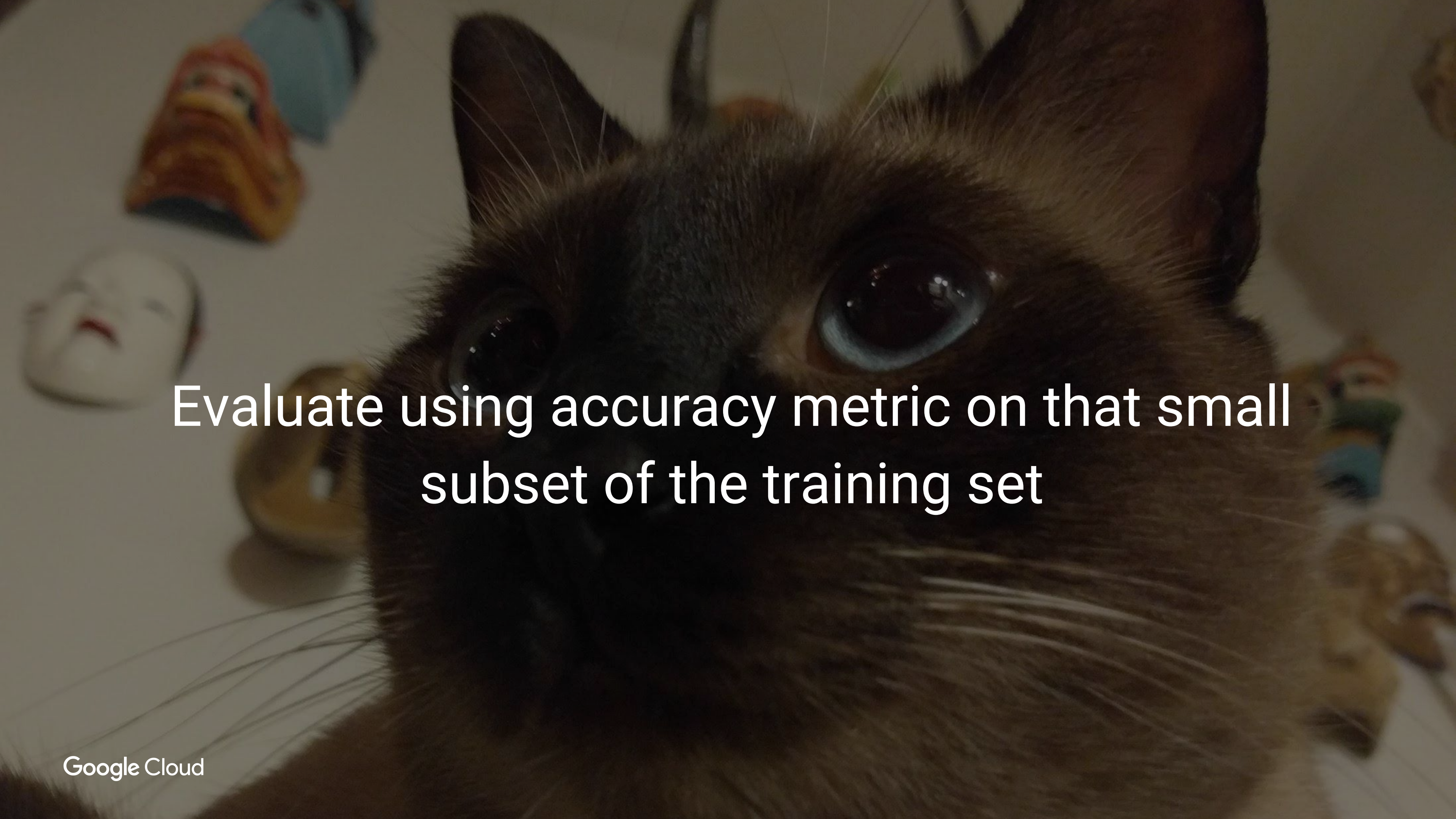
Get predictions and performance metrics

Create functions for outputting observed labels, predicted labels, and accuracy. Filenames must be in the following format: number_number_label.extension

In [13]:
```
# Observed labels from filenames:
def get_labels(dir):
  """Get labels from filenames.

  Filenames must be in the following format: number_number_label.png

  Args:
    dir: directory containing image files

  Returns:
    labels: 1-d np.array of binary labels
  """
  filelist = os.listdir(dir)  # Use all the files in the directory
  labels = np.array([])
  for f in filelist:
    split_filename = f.split('_')
    label = int(split_filename[-1].split('.')[0])
    labels = np.append(labels, label)
  return labels
```

In [14]:
```
# Cat_finder function for getting predictions:
def cat_finder(dir, model_version):
  """Get labels from model.

  Args:
    dir: directory containing image files
```

```
In [14]: # Cat_finder function for getting predictions:
def cat_finder(dir, model_version):
  """Get labels from model.

  Args:
    dir: directory containing image files

  Returns:
    predictions: 1-d np array of binary labels
  """

  num_predictions = len(os.listdir(dir))
  predictions = []  # Initialize array.

  # Estimator.predict() returns a generator g. Call next(g) to retrieve the next value.
  prediction_gen = estimator.predict(
    input_fn=generate_input_fn(dir=dir,
                               batch_size=TRAIN_STEPS,
                               queue_capacity=QUEUE_CAP
                              ),
    checkpoint_path=model_version
  )

  # Use generator to ensure ordering is preserved and predictions match order of validation_labels:
  i = 1
  for pred in range(0, num_predictions):
    predictions.append(next(prediction_gen)) #Append the next value of the generator to the prediction array
    i += 1
    if i % 1000 == 0:
      print('{:d} predictions completed (out of {:d})...'.format(i, len(os.listdir(dir))))
  print('{:d} predictions completed (out of {:d})...'.format(len(os.listdir(dir)), len(os.listdir(dir))))
  return np.array(predictions)
```



Evaluate using accuracy metric on that small subset of the training set

Google Cloud

Get training accuracy

In [15]:
```
def get_accuracy(truth, predictions, threshold=0.5, roundoff = 2):
    """Compares labels with model predictions and returns accuracy.

    Args:
      truth: can be bool (False, True), int (0, 1), or float (0, 1)
      predictions: number between 0 and 1, inclusive
      threshold: we convert the predictions to 1s if they're above this value
      roundoff: report accuracy to how many decimal places?

    Returns:
      accuracy: number correct divided by total predictions
    """
    truth = np.array(truth) == (1|True)
    predicted = np.array(predictions) >= threshold
    matches = sum(predicted == truth)
    accuracy = float(matches) / len(truth)
    return round(accuracy, roundoff)
```

In [17]:
```
files = os.listdir(TRAIN_DIR)
model_version = OUTPUT_DIR + 'model.ckpt-' + str(TRAIN_STEPS)
predicted = cat_finder(TRAIN_DIR, model_version)
observed = get_labels(TRAIN_DIR)
```

```
INFO:tensorflow:Restoring parameters from ../../data/output/model.ckpt-100
```

In [18]:
```
print('Accuracy is ' + str(get_accuracy(observed, predicted)))
```

```
Accuracy is 0.89
```

Key message

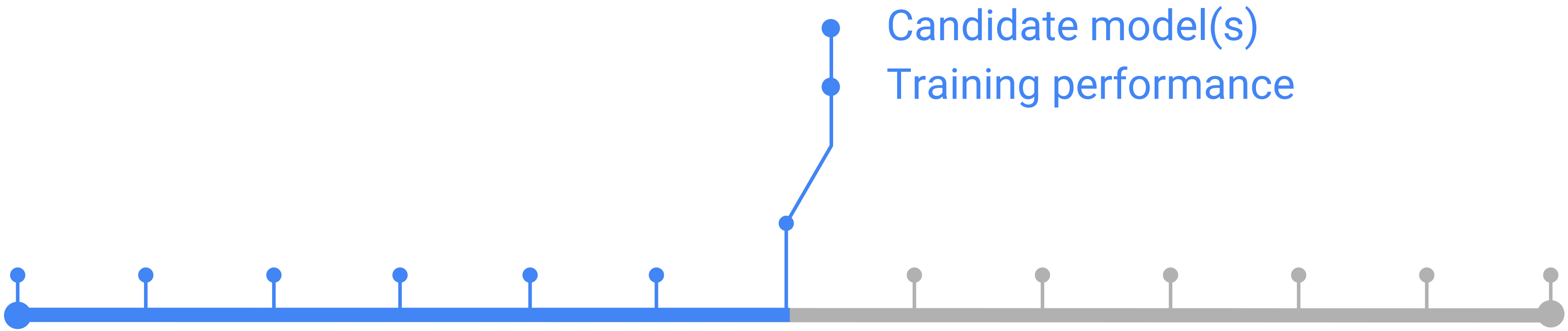
!

In training, your goal is to **find useful patterns** in your data

You're making a **shortlist of models** that seem to work

Don't worry about getting it right immediately **- it'll take a few tries**

Step 6 is finished | You've run some algorithms and got:

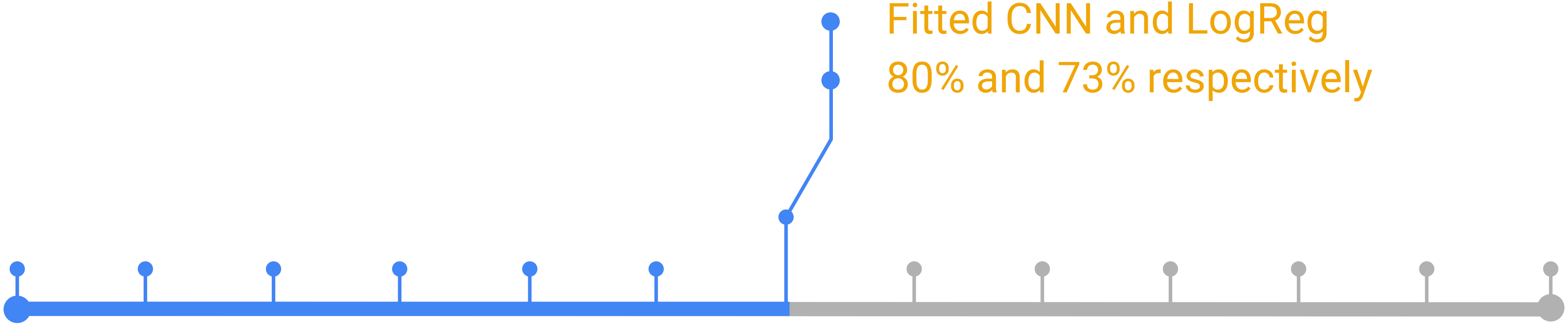


Chief actors:

Engineer

Analyst

Step 6 is finished | You've run some algorithms and got:



Chief actors:

Engineer

Analyst