Building a Caffe Workflow

Mia Polansky

Rice University RECG

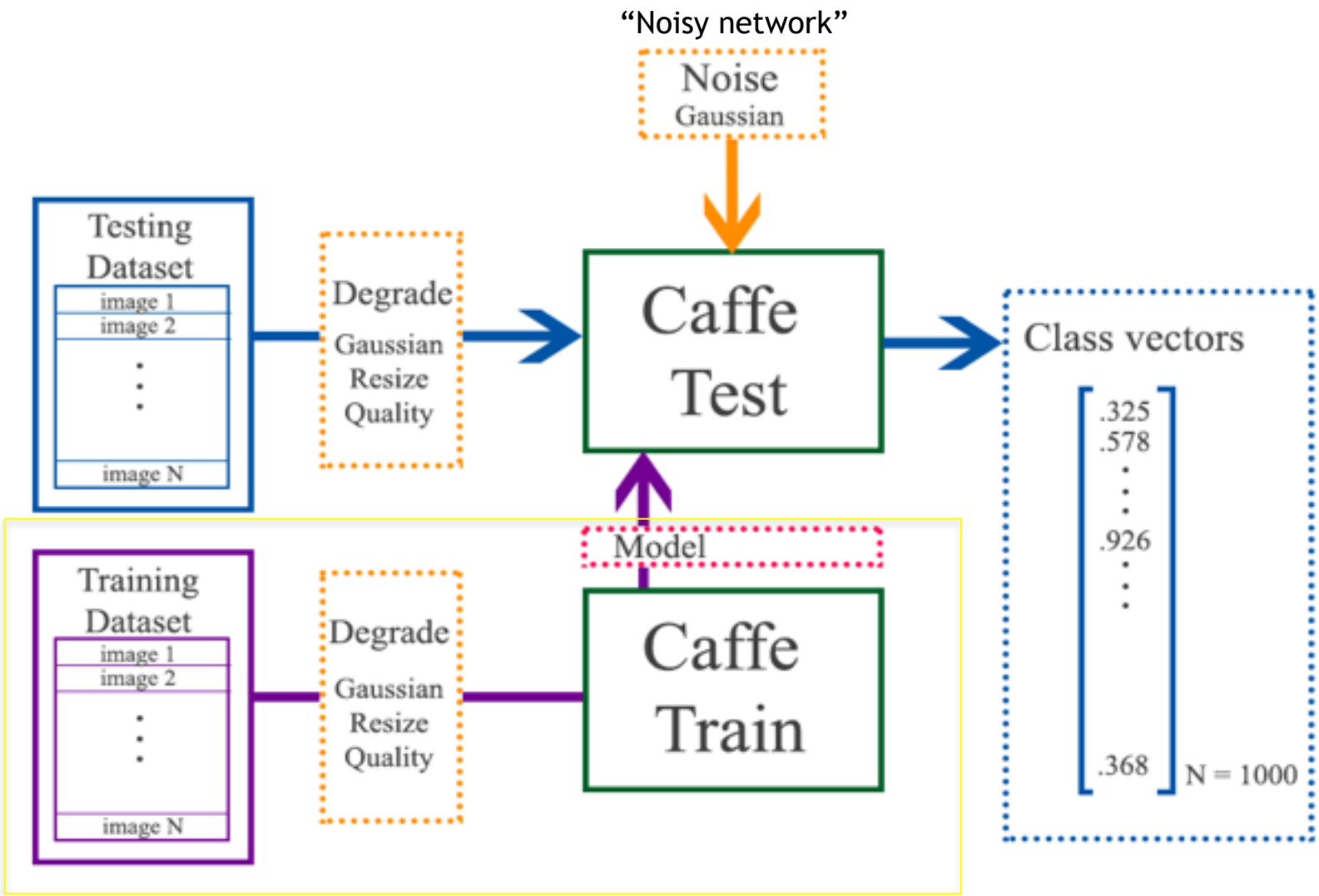
Overview

Motivation: To create a workflow that will allow for streamlined caffe testing and generation of statistics from the testing results

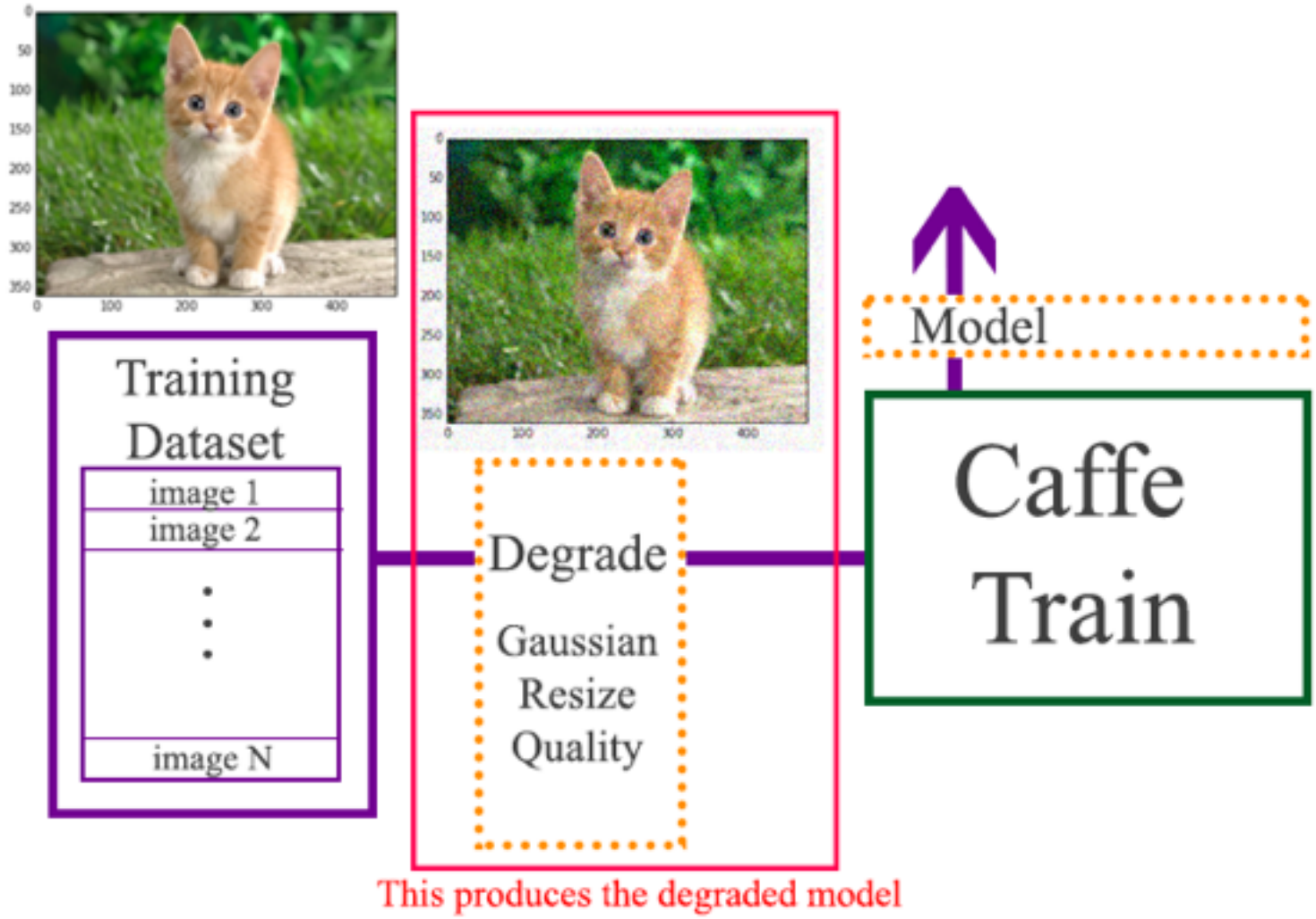
Outline:

- Caffe Refresher
- Overview of the desired workflow
- Automation of testing runs
- Creation of a testing GUI
- Generating statistics

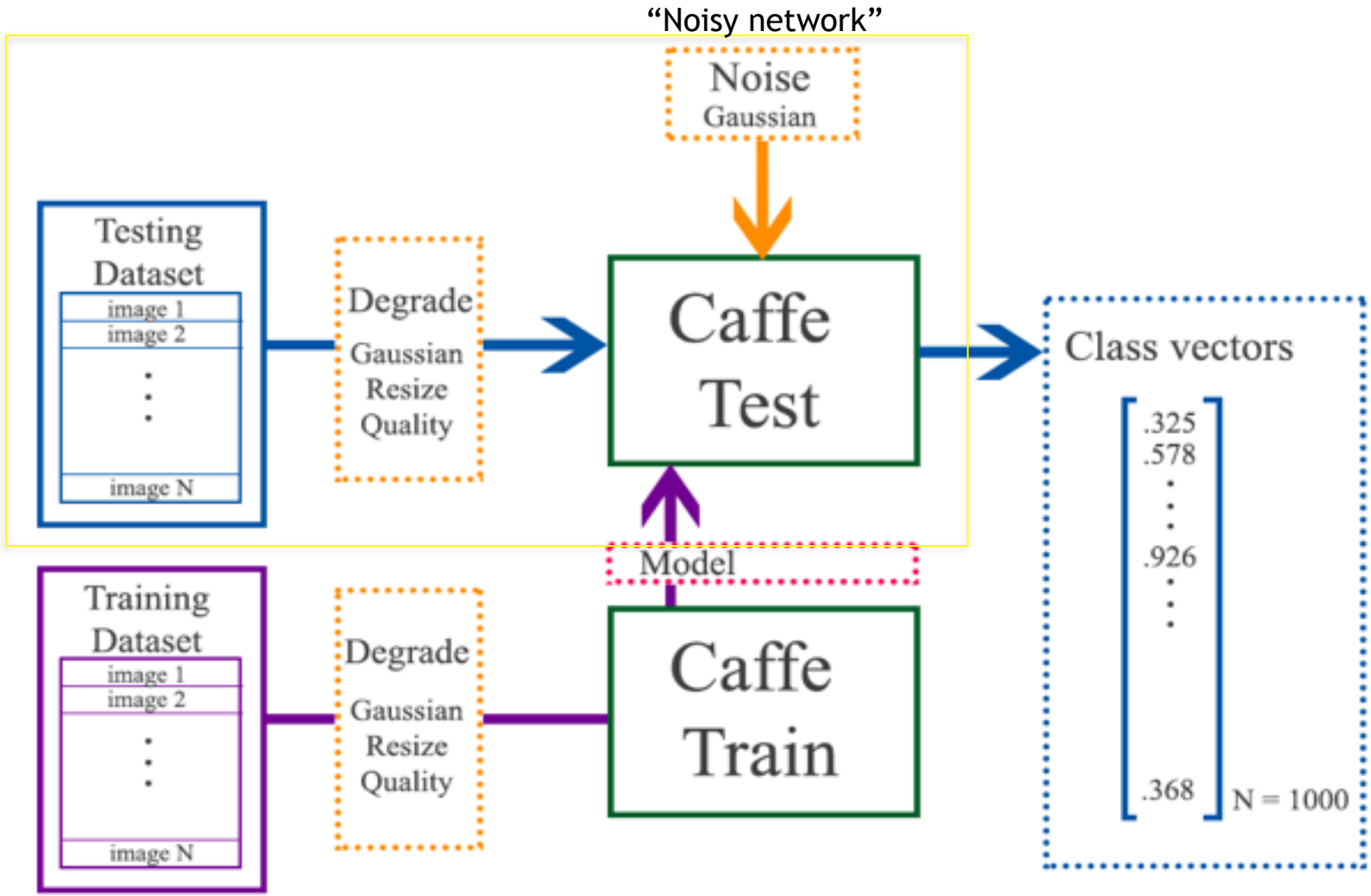
Caffe Refresher: CaffeTrain



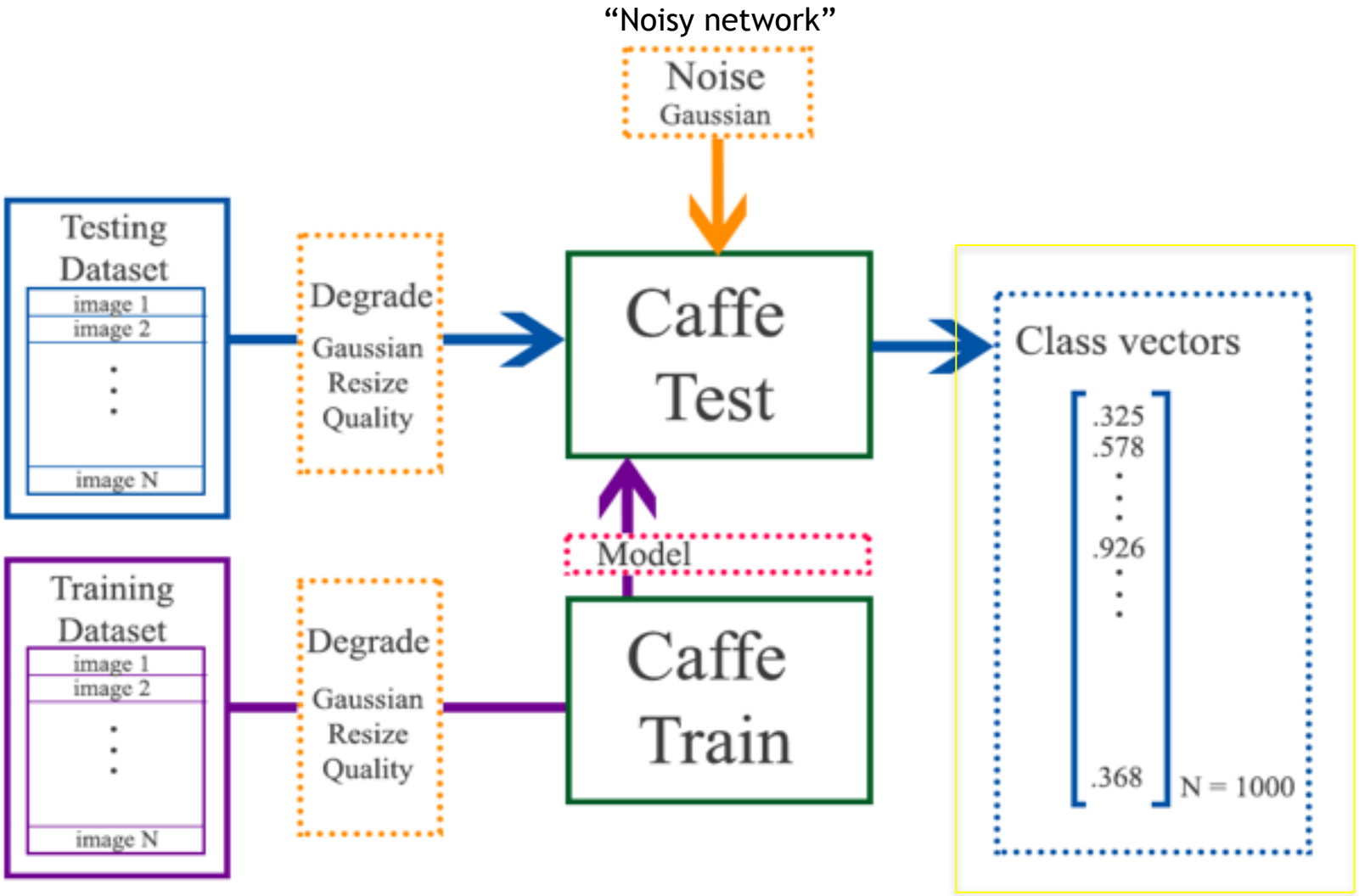
Caffe Refresher: CaffeTrain



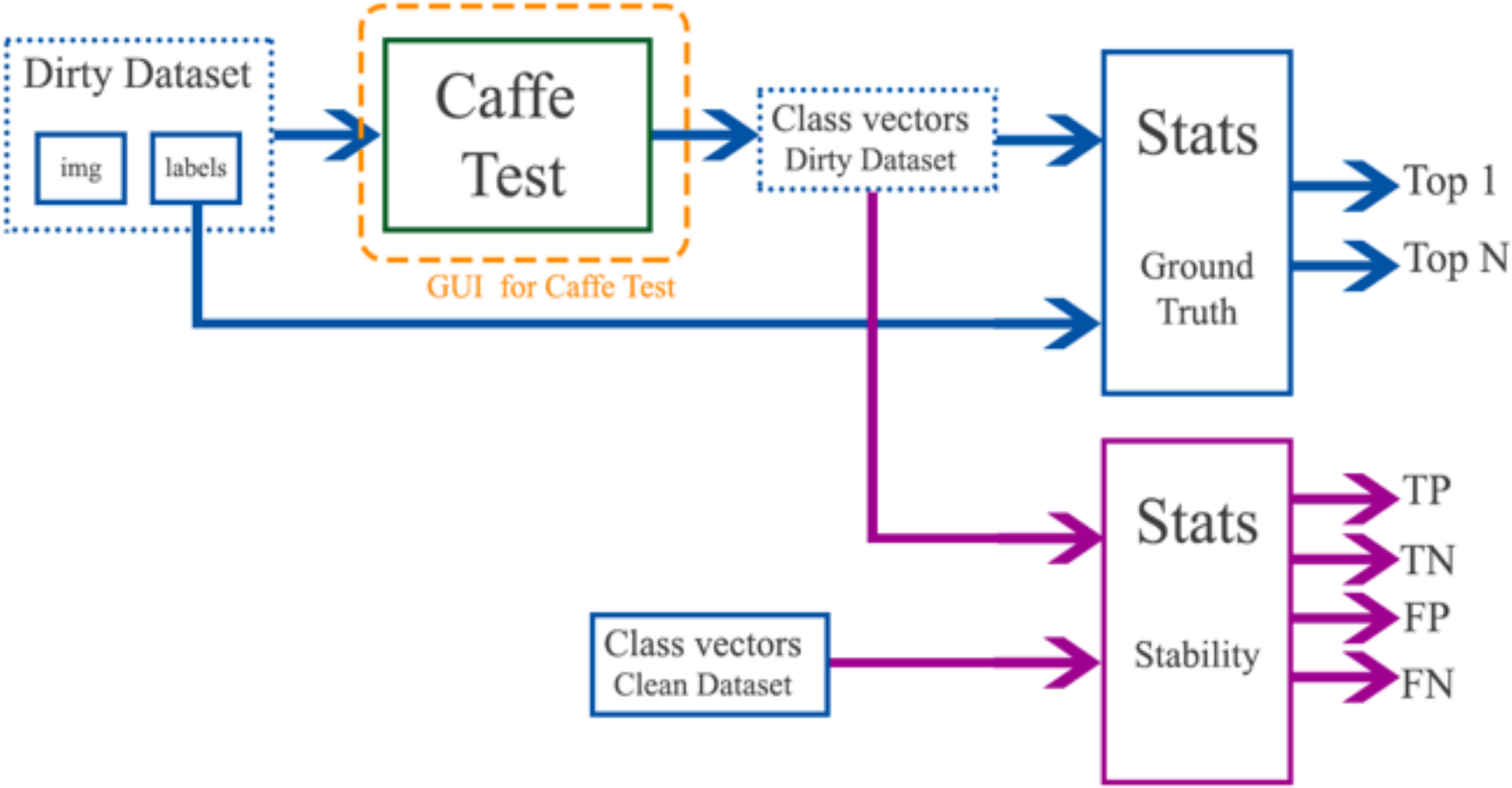
Caffe Refresher: The Output



Caffe Refresher



CaffeTab Workflow Overview



Testing Parameters

- Many parameters needed for testing:
  - General parameters
  - Run parameters
  - Degradation parameters

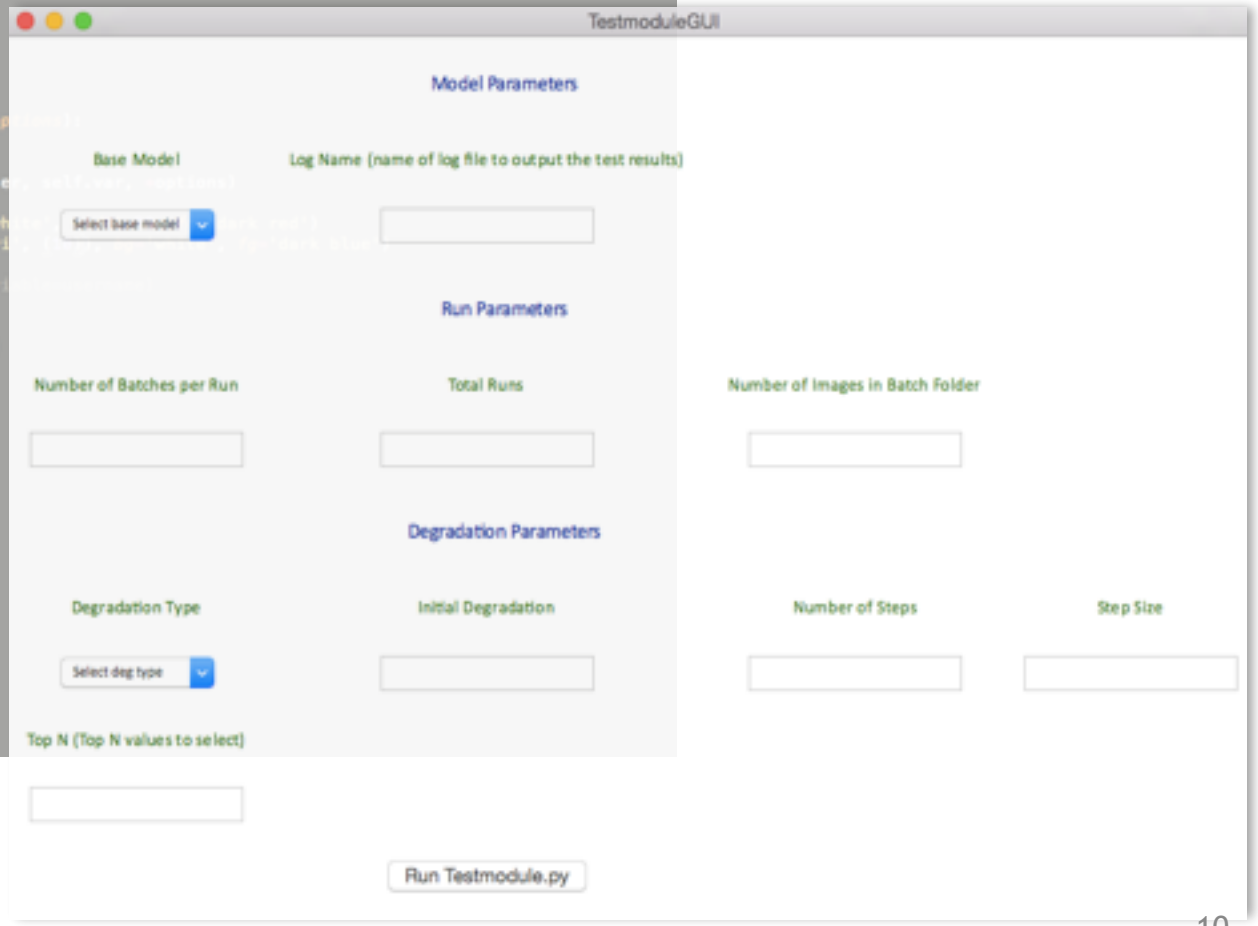
CaffeTab Workflow (1):
Multiple Runs in a Row

- Need to run the Caffe Test for 500 or more folders

```
runs = 500
numBatchesPerRun = 1
numImagesInBatchFolder=100

for i in range(runs):
    #startingBatch = i * numPerBatch;
    new3 = CaffeTester(croot, iroot, 'googlenet', '', i, numBatchesPerRun, numImagesInBatchFolder, 'new_gaussian13')
    new3.degrade(['resize'], .1, 5, .15, 5, i, 0.5)
    #new3.degrade(['clean'], .001, 1, 1, 10, i, 0.5)
```

CaffeTab Workflow (2):
A Testing GUI

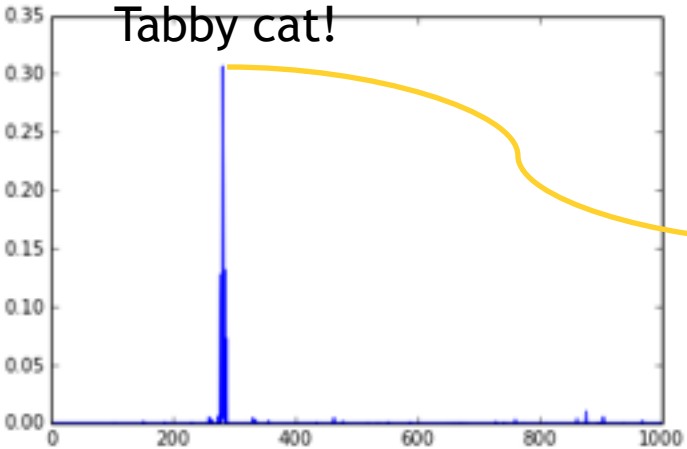
```
import sys
import Tkinter as tk
import subprocess
import testmodule
croot = '../caffe/'
#iroot = '/home/roblkw/Work/caffe/work'
iroot = '/hard2/ILSVRC2014_img_val'

class MyOptionMenu(tk.OptionMenu):
    def __init__(self, master, status, *options):
        self.var = tk.StringVar(master)
        self.var.set(status)
        tk.OptionMenu.__init__(self, master, self.var, *options)
        self.config(
            font=('calibri', (10)), bg='white', fg='dark red')
        self['menu'].config(font=('calibri', 8), bg='white', fg='dark blue')

        #tk.Entry.__init__(parent, textvar[illegible]

    def callback(self):
        val = '{}'.format(self.var.get())
        print(val)
        # subprocess.call([val])


def submit_data2():
    val = mymenu2.get()
    print val

def submit_data3():
    val = mymenu3.get()
    print val

def submit_data4():
    val = mymenu4.get()
    print val

def submit_data5():
    val = mymenu5.get()
    print val
```

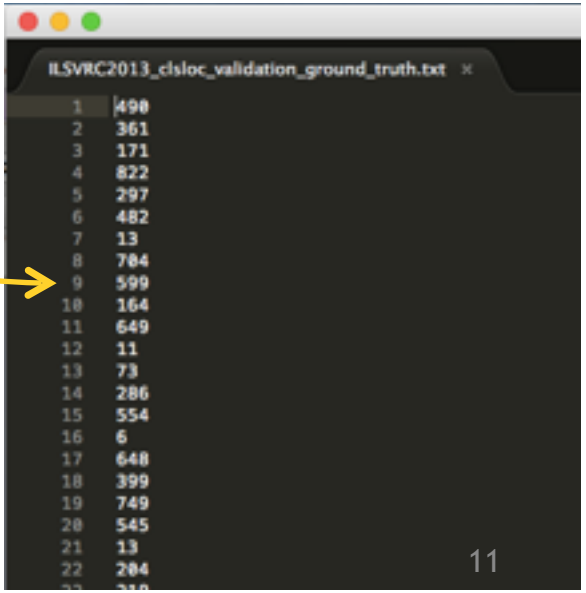


Analyzing the Results

- Caffe outputs 1000 x 1 class vectors that contain the probability of each class
- These are “guesses” → We need to compare against the ground truth

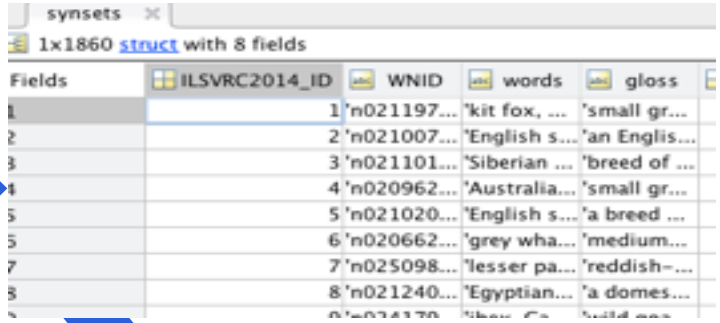
Image #8

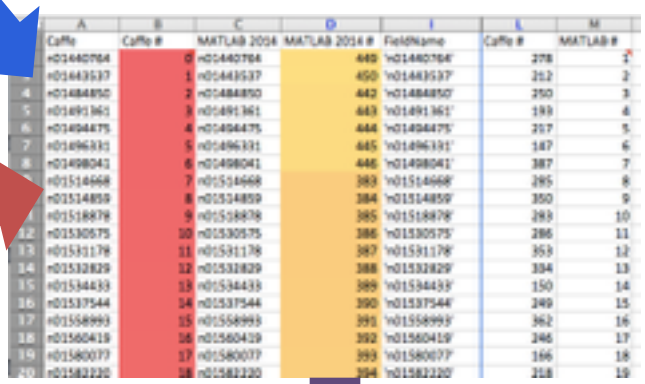




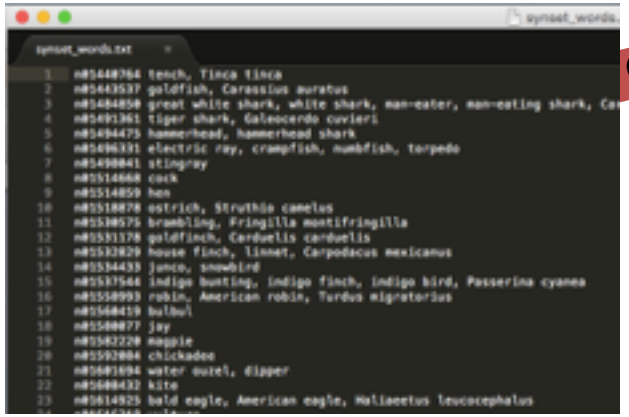
Mapping the Ground Truth Data

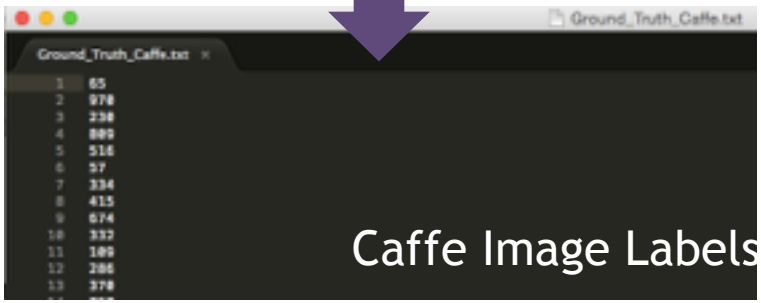
Matlab Numbering





Caffe Numbering





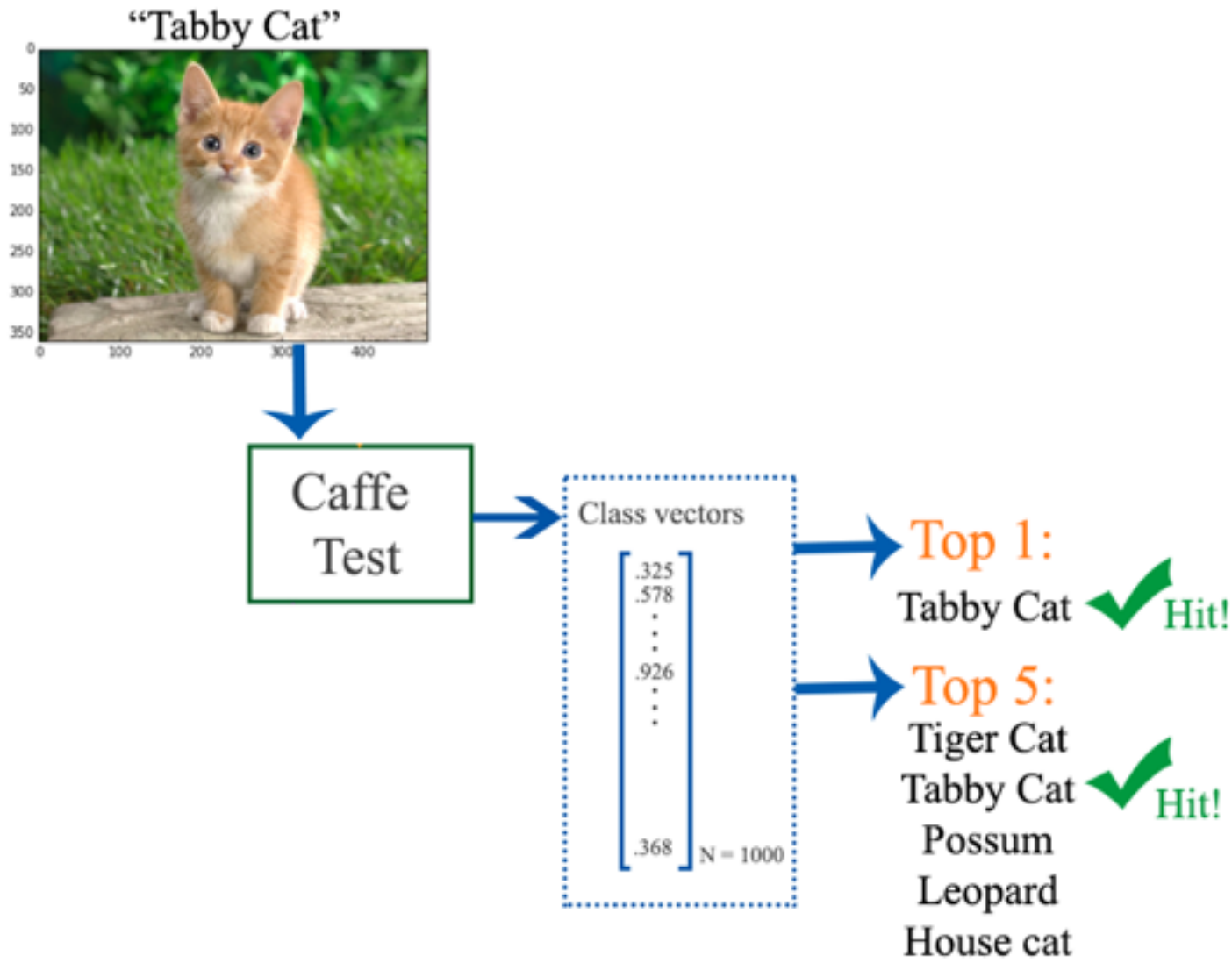
Comparing against the Ground Truth

```
import csv
import numpy as np
import glob
import os
from sys import argv
import argparse

script, num_batches, max_deg_level, img_per_batch, deg_type, toprint = argv

gt = np.loadtxt("Ground_Truth.txt")

def f(a,N):
    return np.argsort(a)[::-1][:N]

#top_1 = np.argmax(gt)

#max_deg_level = 1
#num_batches = 500
#img_per_batch = 100

match5 = np.zeros(int(max_deg_level)+1)
match1 = np.zeros(int(max_deg_level)+1)

for deg_lvl in range(0,int(max_deg_level)+1): #+1?
    for batch in range(0,int(num_batches)):
        #compare1 = np.load("/Users/miap/ensky/caffe/caffe_exp/classvec_2013/googlenet_"+str(batch)+"_"+str(deg_lvl)+"_"+str(batch)+".npy")
        compare1 = np.load("/home/roblkw/[illegible]/caffe_exp/[illegible]_"+str(deg_lvl)+"_"+str(batch)+".npy")
        #print compare1.shape
        for img in range(0,int(img_per_batch)):

            top_5 =  f(compare1[img,:],5)
            #print top_5
            top_1 = np.argmax(compare1[img,:])

            if gt[img+100+batch] == top_1:

                match1[deg_lvl] += 1

            for num in range(0,5):

                if gt[img+100+batch] == top_5[num]:
                    match5[deg_lvl] += 1

np.savetxt("top_5_hit_rate_['"+deg_type+"']_"+max_deg_level+".csv", match5/float(500), delimiter=",")
np.savetxt("top_1_hit_rate_['"+deg_type+"']_"+max_deg_level+".csv", match1/float(500), delimiter=",")

#print "top 5:", match5/float(int(img_per_batch)*int(num_batches)), '\n', "top 1:", match1/float(int(img_per_batch)*int(num_batches))

if toprint.lower() == "true":
    print "top 5:", match5/float(int(img_per_batch)*int(num_batches)), '\n', "top 1:", match1/float(int(img_per_batch)*int(num_batches))
```

Compares the class vectors and finds “hits”:

- Is the correct class same as top predicted class?
- Is the correct class within top N predicted classes?

Comparing against the Ground Truth



Simplifying the Process

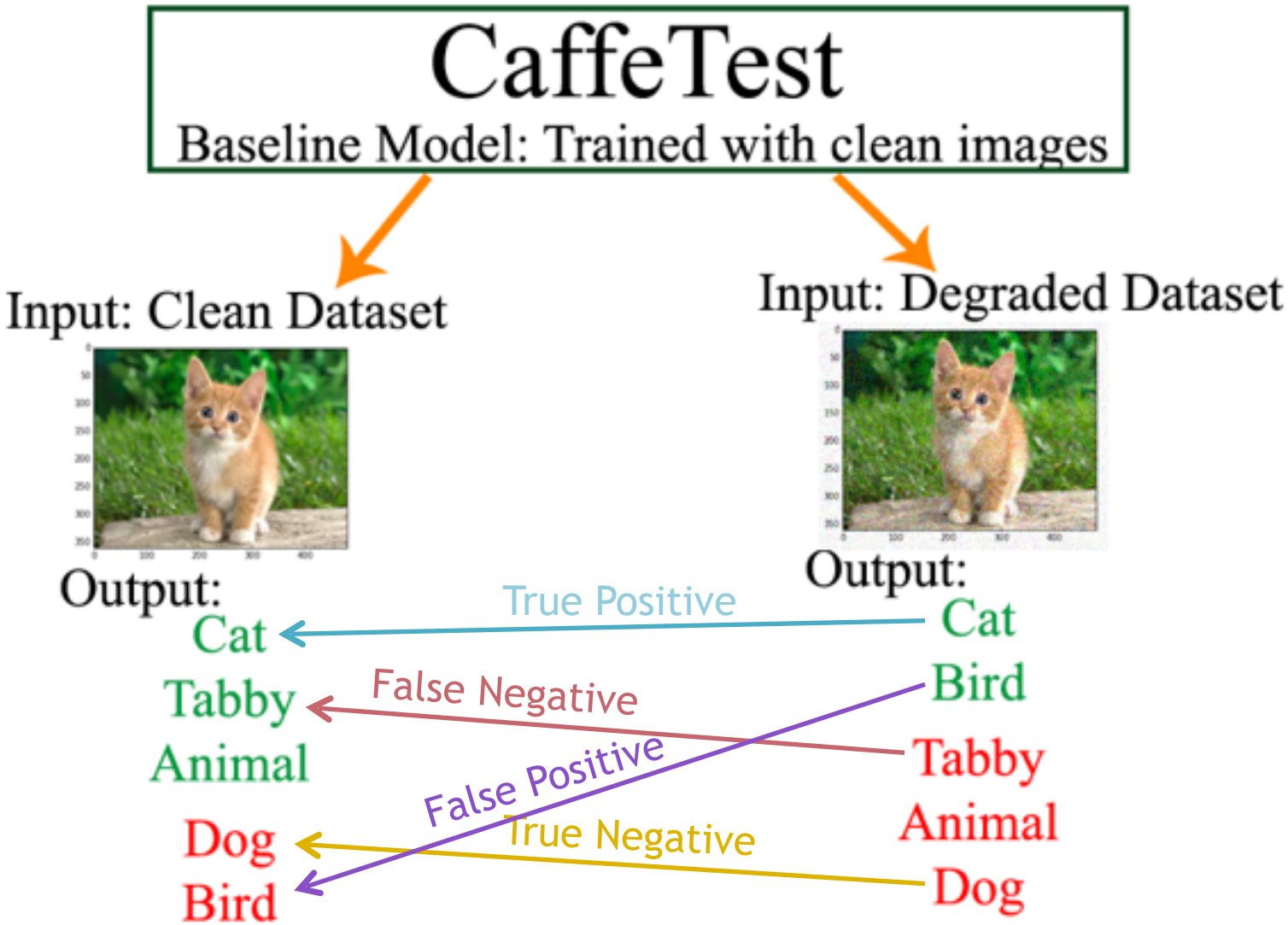
- Still tedious: need to manually input # batches, max degradation level, images per batch, degradation type, and toggle printing results
- → Use argv to define parameters from command line

Ground_Truth_Stats.py

```
#Instructions for running the script:
#This script runs on terminal and takes in 6 arguments (5 if you don't include the file name)
#The arguments are as follows:
    # 1) filename.py --> Ground_Truth_Stats.py
    # 2) num_batches --> Number of batches being analyzed (corresponds to largest first/third number of numpy array file names)
    # 3) max_deg_level --> The maximum degredation level (corresponds to largest second number of numpy array file names)
    # 4) img_per_batch --> number of images per batch
    # 5) deg_type --> degredation type
    # 5) toprint --> Whether to print results; either true or false
# Example:
# miapolansky$ Ground_Truth_Stats.py 500 0 100 Gaussian True
```

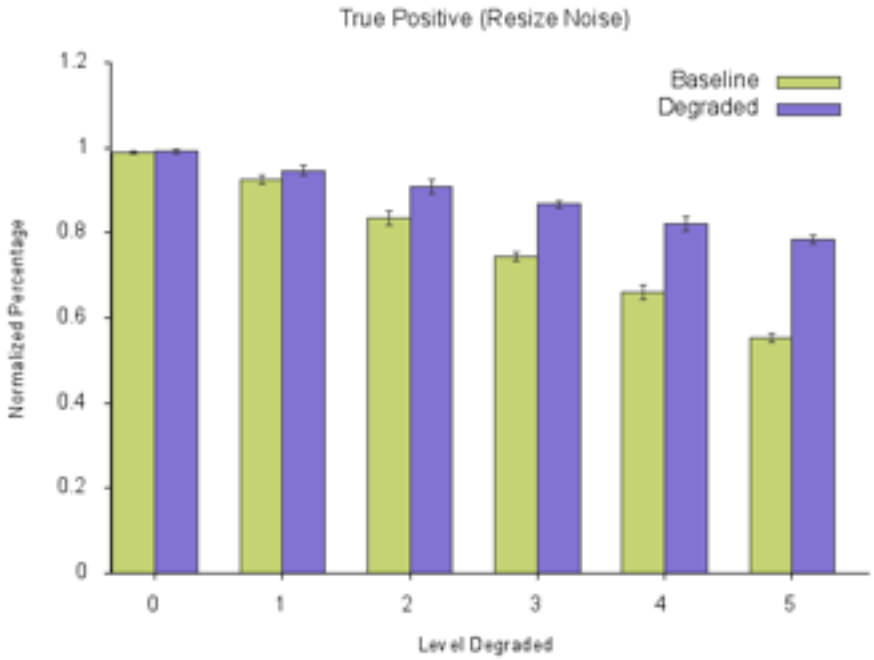
```
Mia:~ miapolansky$ cd caffe/caffe_exp/tools
Mia:tools miapolansky$ python Ground_Truth_Stats.py 500 0 100 Gaussian True
```

Calculating TP, TN, FP, FN



Visualizing the Results

- Afterwards, graph data with gnuplot



Baseline: Model trained on clean dataset, run on clean network
Degraded: Model trained on degraded dataset, run of clean network
Noisy: Model trained on noisy network and degraded dataset, run on noisy network