Versioning and Experiment Tracking

Why do we need experiment tracking tools?

At first glance, building and deploying machine learning models looks a lot like writing code.

But there are some key differences that make machine learning harder:

- Have far more branching and experimentation than a typical software project.
- Rather than throwing errors, it just underperforms, making debugging extra difficult and time consuming.
- A single small change in training data, training code or hyperparameters can wildly change a model's performance, so reproducing earlier work often requires exactly matching the prior setup.
- Running machine learning experiments can be time consuming and compute costs may be expensive.

https://towardsdatascience.com/machine-learning-experiment-tracking-93b796e501b0

PROBLEM

Massive developer tools gap for ML

SOFTWARE 1.0 - WRITE CODE

Design	Version Code	Code & Collaborate	Deploy to Infra	CI/CD	Production Monitoring
Figma	GitHub	Jira	HashiCorp	circleci	PagerDuty
Sketch	GitLab	VS Code	puppet	Jenkins	DATADOG

SOFTWARE 2.0 - TRAIN MODELS

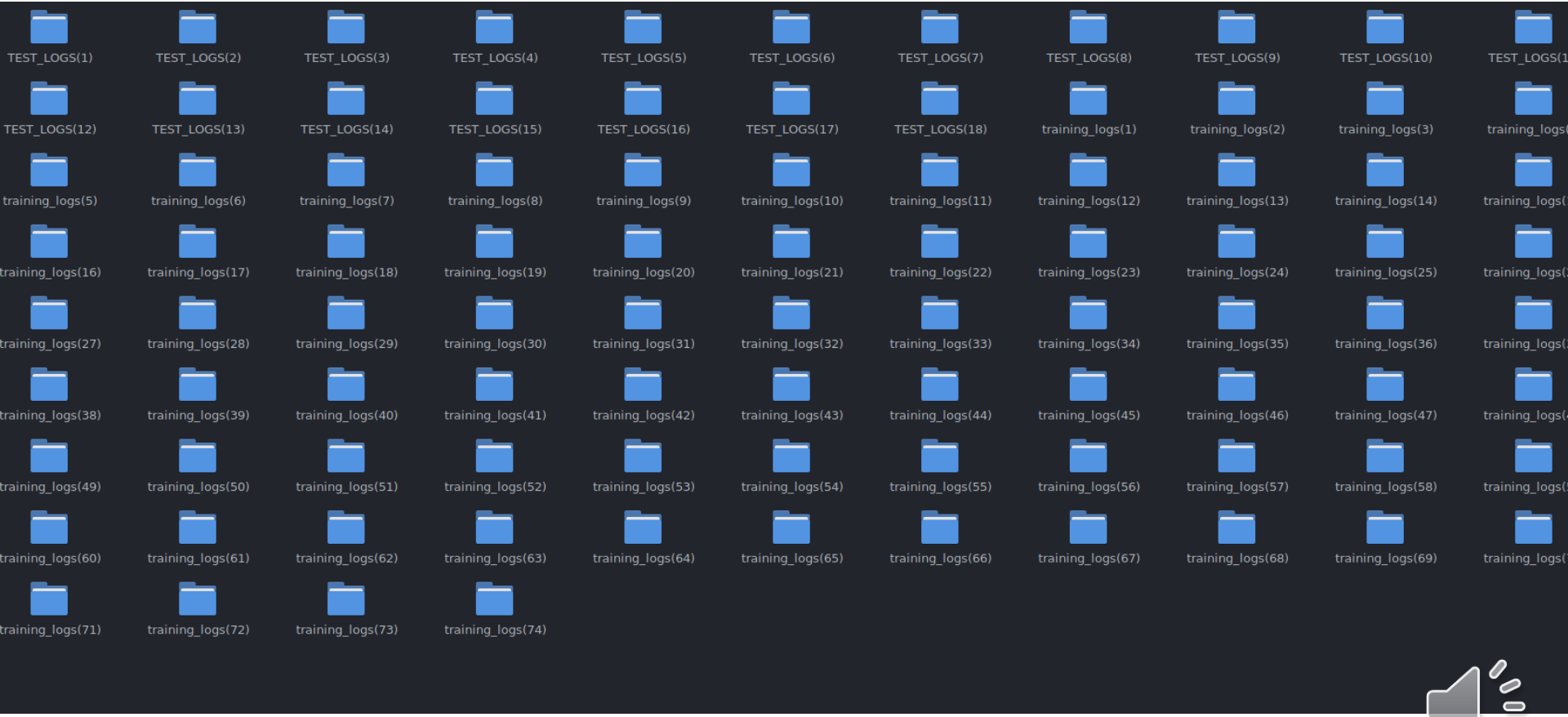
Prep & Visualize Data	Version Data & Models	Experiment Tracking	Manage Model Pipeline	Model CI/CD	Production monitoring
Custom apps Notebooks	Files in S3	Text files Screenshots	Text files	Custom scripts	Nothing

Slide Credit: Lavanya Shukla

Why do we need experiment tracking tools?

Suppose you are training a model with different options on;

- Training-validation-test split
- Device (RTX2080Ti, TPU v2 pod, etc.)
- Hyperparameters
- Training procedure (eg. FPN for object detection)
- ...


TEST_LOGS(1)
TEST_LOGS(2)
TEST_LOGS(3)
TEST_LOGS(4)
TEST_LOGS(5)
TEST_LOGS(6)
TEST_LOGS(7)
TEST_LOGS(8)
TEST_LOGS(9)
TEST_LOGS(10)
TEST_LOGS(12)
TEST_LOGS(13)
TEST_LOGS(14)
TEST_LOGS(15)
TEST_LOGS(16)
TEST_LOGS(17)
TEST_LOGS(18)
training_logs(1)
training_logs(2)
training_logs(3)
training_logs(5)
training_logs(6)
training_logs(7)
training_logs(8)
training_logs(9)
training_logs(10)
training_logs(11)
training_logs(12)
training_logs(13)
training_logs(14)
training_logs(16)
training_logs(17)
training_logs(18)
training_logs(19)
training_logs(20)
training_logs(21)
training_logs(22)
training_logs(23)
training_logs(24)
training_logs(25)
training_logs(27)
training_logs(28)
training_logs(29)
training_logs(30)
training_logs(31)
training_logs(32)
training_logs(33)
training_logs(34)
training_logs(35)
training_logs(36)
training_logs(38)
training_logs(39)
training_logs(40)
training_logs(41)
training_logs(42)
training_logs(43)
training_logs(44)
training_logs(45)
training_logs(46)
training_logs(47)
training_logs(49)
training_logs(50)
training_logs(51)
training_logs(52)
training_logs(53)
training_logs(54)
training_logs(55)
training_logs(56)
training_logs(57)
training_logs(58)
training_logs(60)
training_logs(61)
training_logs(62)
training_logs(63)
training_logs(64)
training_logs(65)
training_logs(66)
training_logs(67)
training_logs(68)
training_logs(69)
training_logs(71)
training_logs(72)
training_logs(73)
training_logs(74)

Why do we need experiment tracking tools?

These tools take away the burden of;

- Manually managing experiment runs
- Finding logs of specific hyper-parameters or configurations.
- Managing version of dataset (optional)
- Managing orchestration (optional)

	Neptune	Weights & Biases	Comet	Sacred + Omniboard	MLflow	TensorBoard	Guild AI	PolyAxon	TRAINS	Valohai	Pachyderm	Kubeflow	Verta.ai	SageMaker Studio	DVC
Overview															
Focus	Experiment Management	Experiment Management	Experiment Management	Experiment Management	Entire Lifecycle	Experiment Management	Experiment Management	Experiment Management	Experiment Management	Entire Lifecycle	Entire Lifecycle	Run orchestration	Entire Lifecycle	Entire Lifecycle	Data Versioning
Price	• Free • Team Trial: 1 project • Team Research: $0 • Team Startup: $39 per user • Team Pro: $79 per user • Enterprise: starts at $1799	• Personal • Startup: $35 per user • Team: $100 per user • Enterprise: NA	• Free • Startup: $39 per user • Teams: $179 per user • Enterprise: NA	Free	Free	Free	Free	Enterprise: NA	Free	• Free • Pro: $649 • Enterprise: NA	• Community Edition • Enterprise Edition: NA	Free	Enterprise: NA	You pay extra on top of compute	Free
Free plan limitation	• Free: 1 user • Team Free: 1 project • Team Research: no limits	• 1 user • limited support	• 1 user • no private links (sharing)							only 3 days of data available					
Open – source				✓	✓	✓	✓	limited	✓		limited	✓			✓
Easy integration	✓	✓	✓	• Somewhat difficult	✓	✓	✓								✓
				• Depends on the on backend											
Lightweight	✓	✓	✓	Somewhat	✓	✓	✓		✓						✓

source: https://neptune.ai/wp-content/uploads/best-machine-learning-experiment-tracking-apps.pdf

Benefits over Tensorboard

Cloud storage of results

- Remote tracking of experiments.

Hyperparameter Tuning

- Self contained or integration with other tools.

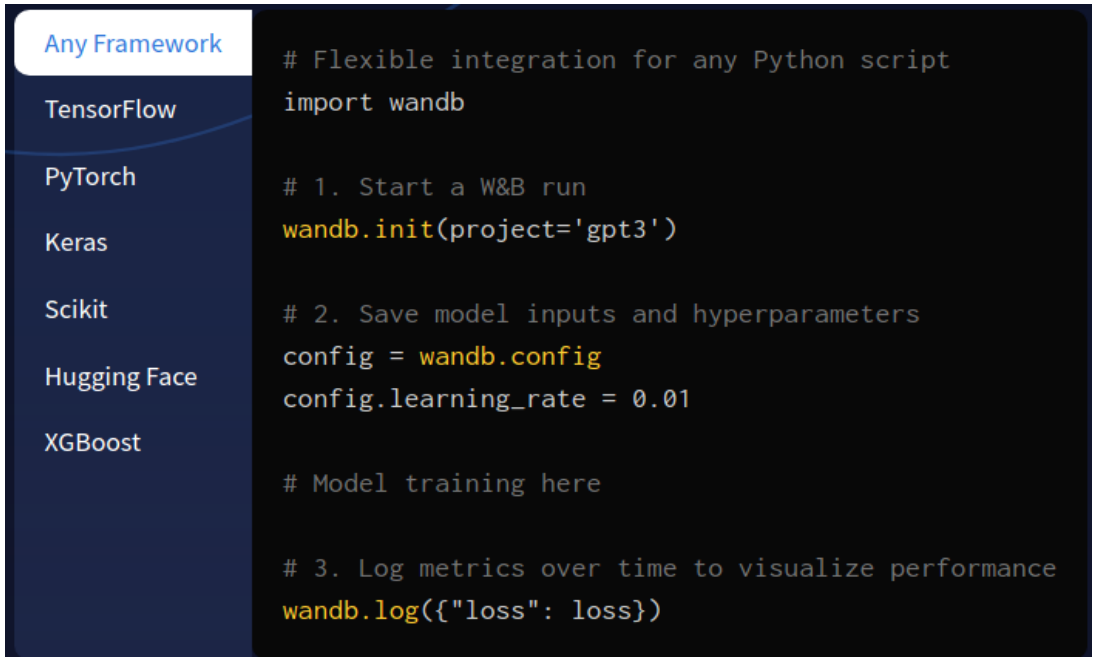
Easier tracking

- Better UI
- Comprehensible statistics over hardware usage and parameter distributions

Framework Agnostic

- Can work with any framework (Even works without any framework)

Weights & Biases



Weights & Biases

Any Framework

TensorFlow

PyTorch

Keras

Scikit

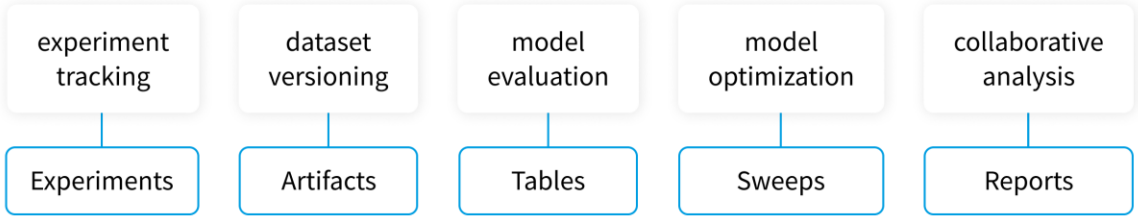
Hugging Face

XGBoost

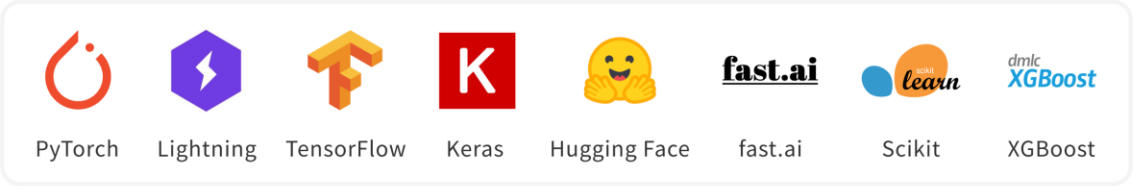
```
import wandb

# 1. Start a new run
wandb.init(project="gpt-3")

# 2. Save model inputs and hyperparameters
config = wandb.config
config.learning_rate = 0.01

# 3. Log gradients and model parameters
wandb.watch(model)
for batch_idx, (data, target) in
enumerate(train_loader):
  ...
  if batch_idx % args.log_interval == 0:
    # 4. Log metrics to visualize performance
    wandb.log({"loss": loss})
```

ML Developer Tools for the entire ML workflow

W&B PLATFORM



FRAMEWORK AGNOSTIC



ENVIRONMENT AGNOSTIC

Weights & Biases

1. Install the lightweight library
2. Track metrics and artifacts
3. Compare experiments fast
4. Optimize hyperparameters easily
5. Collaborate on reproducible models

Light integration

Try W&B in 5 minutes:

- Intro notebook
  bit.ly/intro-wb
- Quickstart docs
  docs.wandb.com

PYTORCH

```
import torch

model.train()
```

```
import torch
import wandb

wandb.watch(model)
model.train()
```

TENSORFLOW

```
import tensorflow as tf

classifier.train()
```

```
import tensorflow as tf
import wandb

wandb.init(sync_tensorboard=True)

classifier.train()
```

KERAS

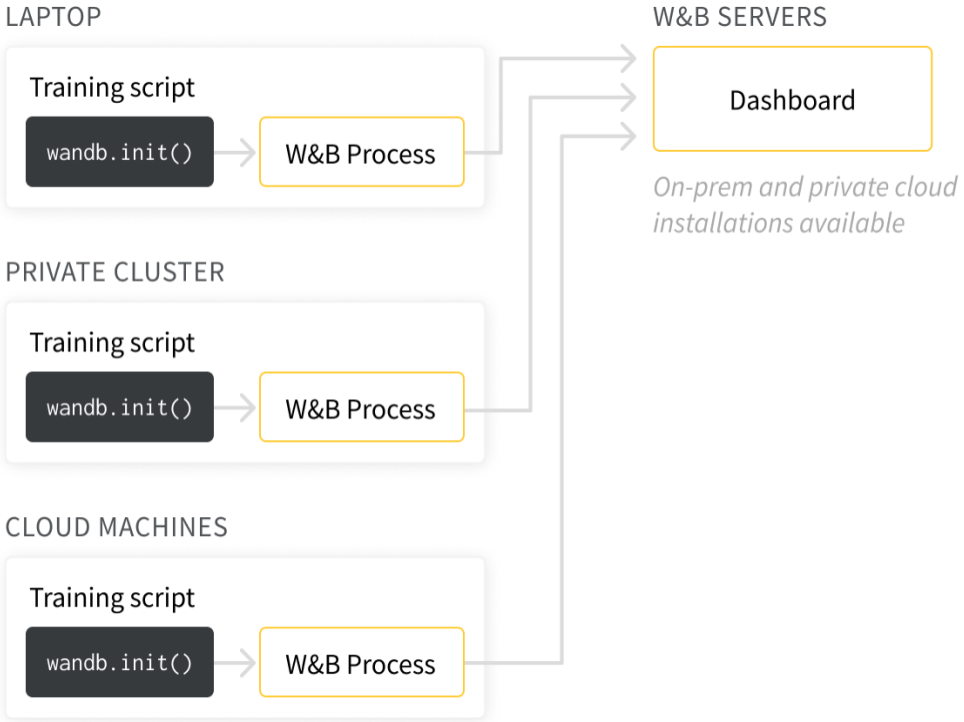
```
from tensorflow import keras

model.fit(X, y)
```

```
from tensorflow import keras
import wandb

model.fit(X, y,
callbacks=[WandbCallback()])
```

Infra agnostic

W&B can be used

- In their hosted cloud
- In a private cloud
- On-premise-on your own machines



Track experiments

See live updates on model performance, check for overfitting, and visualize how a model performs on different classes.

bit.ly/demo-run

Track experiments – Before W&B

```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
conv2d_1_input (InputLayer)  (None, 299, 299, 3)  0
__________________________________________________________________________________________________
lambda_1 (Lambda)            (None, 299, 299, 3)  0           conv2d_1_input[0][0]
__________________________________________________________________________________________________
lambda_2 (Lambda)            (None, 299, 299, 3)  0           conv2d_1_input[0][0]
__________________________________________________________________________________________________
sequential_1 (Sequential)    (None, 10)            910922      lambda_1[0][0]
                                                               lambda_2[0][0]
__________________________________________________________________________________________________
activation_7 (Concatenate)   (None, 10)            0           sequential_1[1][0]
                                                               sequential_1[2][0]
==================================================================================================
Total params: 910,922
Trainable params: 910,922
Non-trainable params: 0
__________________________________________________________________________________________________
None
Found 49999 images belonging to 10 classes.
Found 8000 images belonging to 10 classes.
Epoch 1/50
39/39 [==============================] - 270s 7s/step - loss: 2.3153 - acc: 0.1178 - val_loss: 2.2428 - val_acc: 0.1589
Epoch 2/50
39/39 [==============================] - 263s 7s/step - loss: 2.2588 - acc: 0.1538 - val_loss: 2.1890 - val_acc: 0.2174
Epoch 3/50
39/39 [==============================] - 262s 7s/step - loss: 2.2060 - acc: 0.1827 - val_loss: 2.1767 - val_acc: 0.2096
Epoch 4/50
39/39 [==============================] - 263s 7s/step - loss: 2.1640 - acc: 0.2107 - val_loss: 2.1133 - val_acc: 0.2357
Epoch 5/50
39/39 [==============================] - 261s 7s/step - loss: 2.0827 - acc: 0.2432 - val_loss: 2.1499 - val_acc: 0.2344
Epoch 6/50
39/39 [==============================] - 262s 7s/step - loss: 2.0810 - acc: 0.2508 - val_loss: 2.0289 - val_acc: 0.2604
Epoch 7/50
39/39 [==============================] - 262s 7s/step - loss: 2.0575 - acc: 0.2602 - val_loss: 1.9912 - val_acc: 0.2865
Epoch 8/50
39/39 [==============================] - 261s 7s/step - loss: 2.0355 - acc: 0.2734 - val_loss: 2.0319 - val_acc: 0.2409
Epoch 9/50
39/39 [==============================] - 263s 7s/step - loss: 2.0254 - acc: 0.2829 - val_loss: 1.9364 - val_acc: 0.3307
Epoch 10/50
39/39 [==============================] - 262s 7s/step - loss: 2.0056 - acc: 0.2804 - val_loss: 1.9934 - val_acc: 0.2773
Epoch 11/50
39/39 [==============================] - 256s 7s/step - loss: 1.9678 - acc: 0.3048 - val_loss: 1.9048 - val_acc: 0.3352
Epoch 12/50
39/39 [==============================] - 260s 7s/step - loss: 1.9634 - acc: 0.3109 - val_loss: 1.8758 - val_acc: 0.3568
```

Table 3: Detection results on **PASCAL VOC 2007 test set**. The detector is Fast R-CNN and VGG-16. Training data: "07": VOC 2007 trainval, "07+12": union set of VOC 2007 trainval and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000. †: this number was reported in [2]; using the repository provided by this paper, this result is higher (68.1).

method	# proposals	data	mAP (%)
SS	2000	07	66.9†
SS	2000	07+12	70.0
RPN+VGG, unshared	300	07	68.5
RPN+VGG, shared	300	07	69.9
RPN+VGG, shared	300	07+12	**73.2**
RPN+VGG, shared	300	COCO+07+12	**78.8**

Table 4: Detection results on **PASCAL VOC 2012 test set**. The detector is Fast R-CNN and VGG-16. Training data: "07": VOC 2007 trainval, "07++12": union set of VOC 2007 trainval+test and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000. †: http://host.robots.ox.ac.uk:8080/anonymous/HZJTQA.html. ‡: http://host.robots.ox.ac.uk:8080/anonymous/YNPLXB.html. §: http://host.robots.ox.ac.uk:8080/anonymous/XEDH10.html.

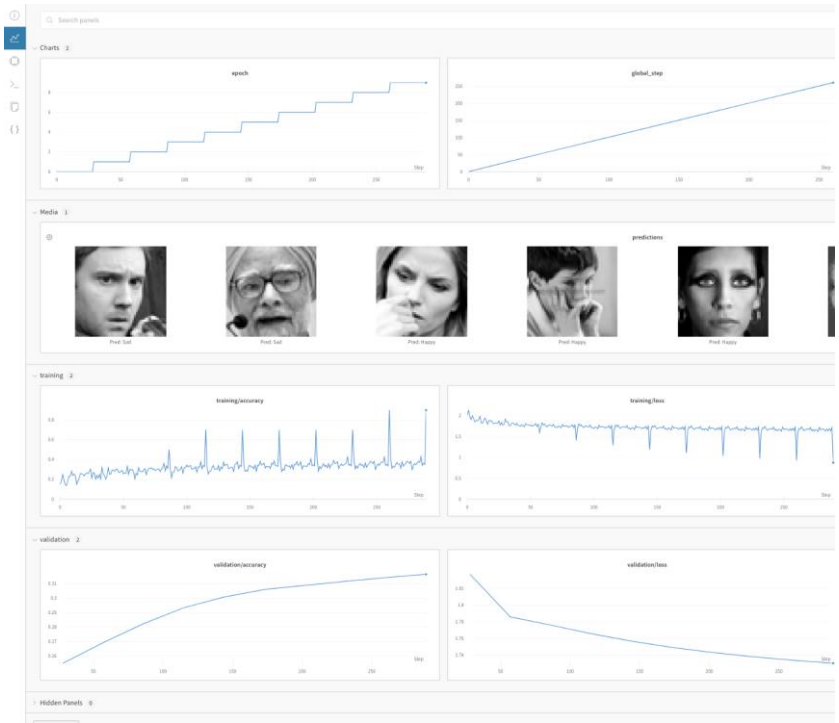
method	# proposals	data	mAP (%)
SS	2000	12	65.7
SS	2000	07++12	68.4
RPN+VGG, shared†	300	12	67.0
RPN+VGG, shared‡	300	07++12	**70.4**
RPN+VGG, shared§	300	COCO+07++12	**75.9**

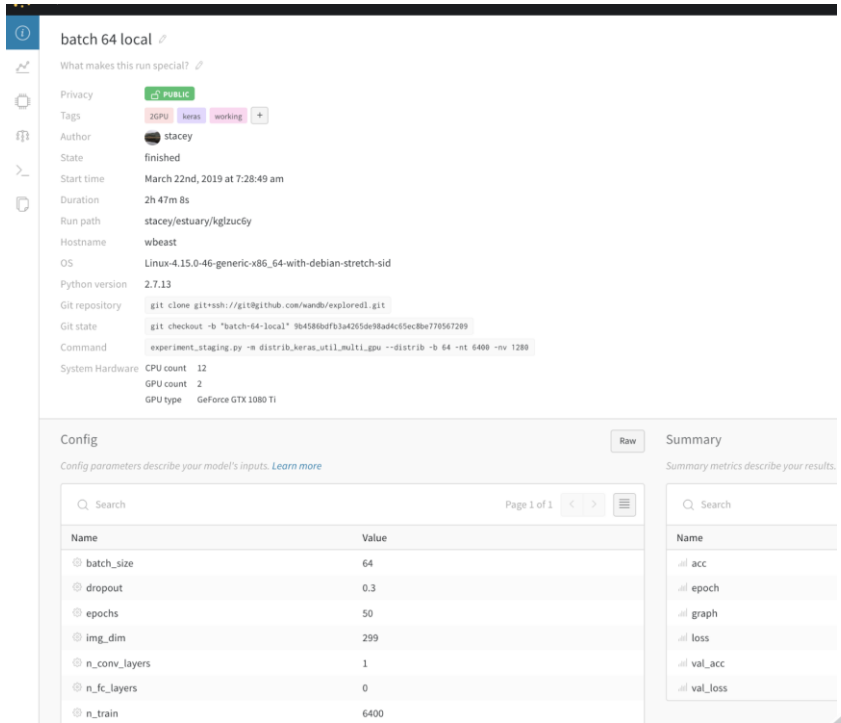
Table 5: **Timing** (ms) on a K40 GPU, except SS proposal is evaluated in a CPU. "Region-wise" includes NMS, pooling, fully-connected, and softmax layers. See our released code for the profiling of running time.

model	system	conv	proposal	region-wise	total	rate
VGG	SS + Fast R-CNN	146	1510	174	1830	0.5 fps
VGG	RPN + Fast R-CNN	141	**10**	47	**198**	**5 fps**
ZF	RPN + Fast R-CNN	31	**3**	25	**59**	**17 fps**

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [5]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [8]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [6]	Inception-ResNet-v2 [21]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [20]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [15]	DarkNet-19 [15]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [11, 3]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [3]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [9]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [9]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

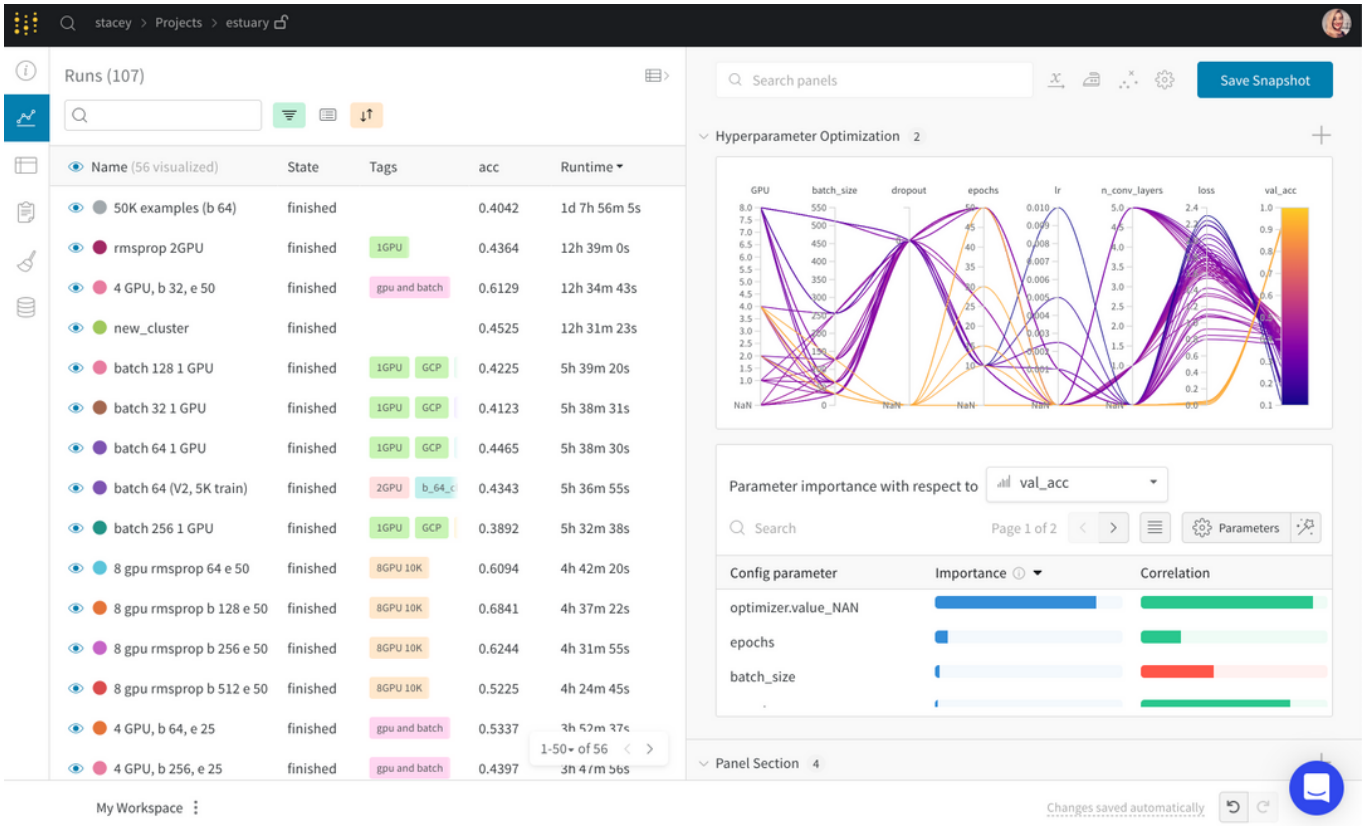
Track experiments – After W&B





[Live Dashboard]

Track experiments – After W&B

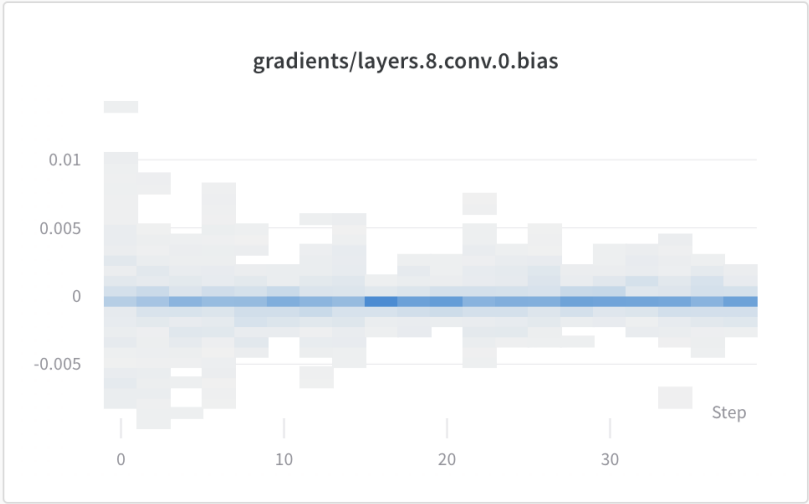


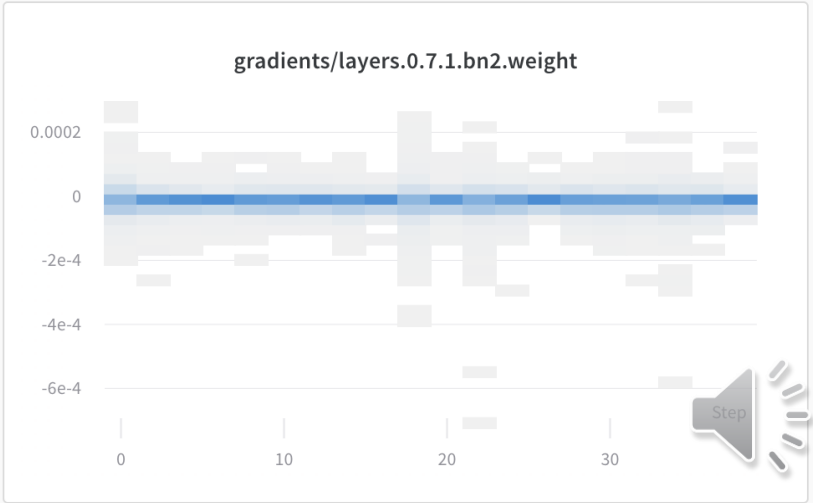
Live Dashboard

bit.ly/demo-run

Visualize gradients

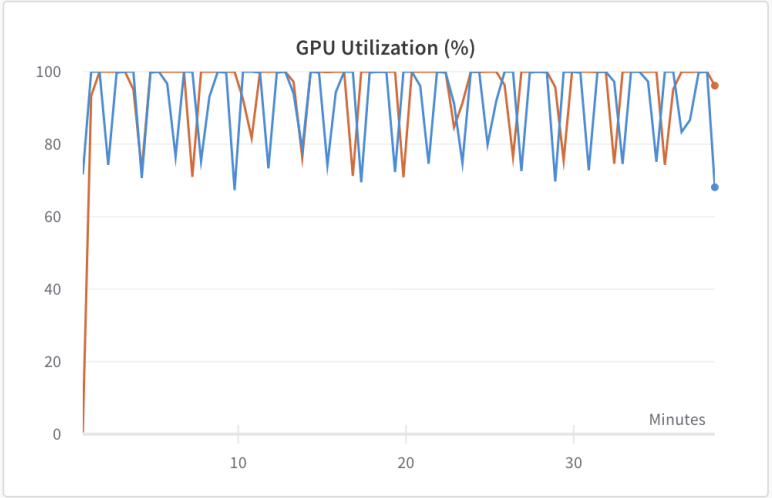
See the inner workings of your model, look for convergence during training, and check for exploding gradients.

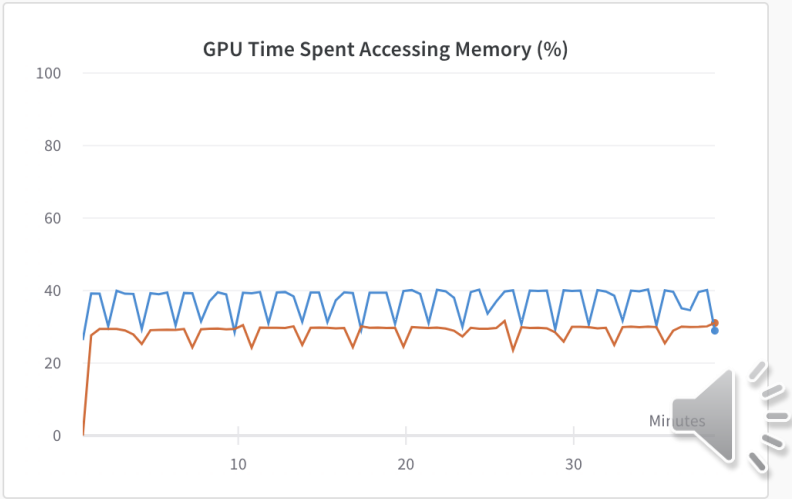




System metrics

Get the most out of your GPUs and identify opportunities for optimizing hardware utilization.





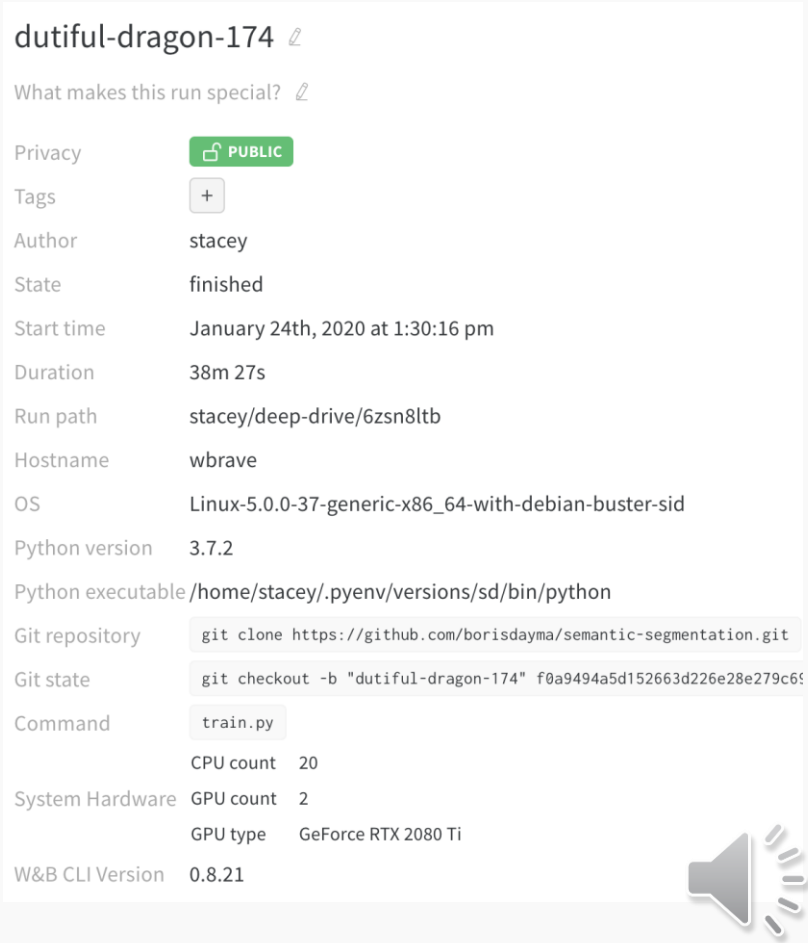
bit.ly/demo-run

Capture the code

Save the most recent git commit, the command and args, and system hardware setup.

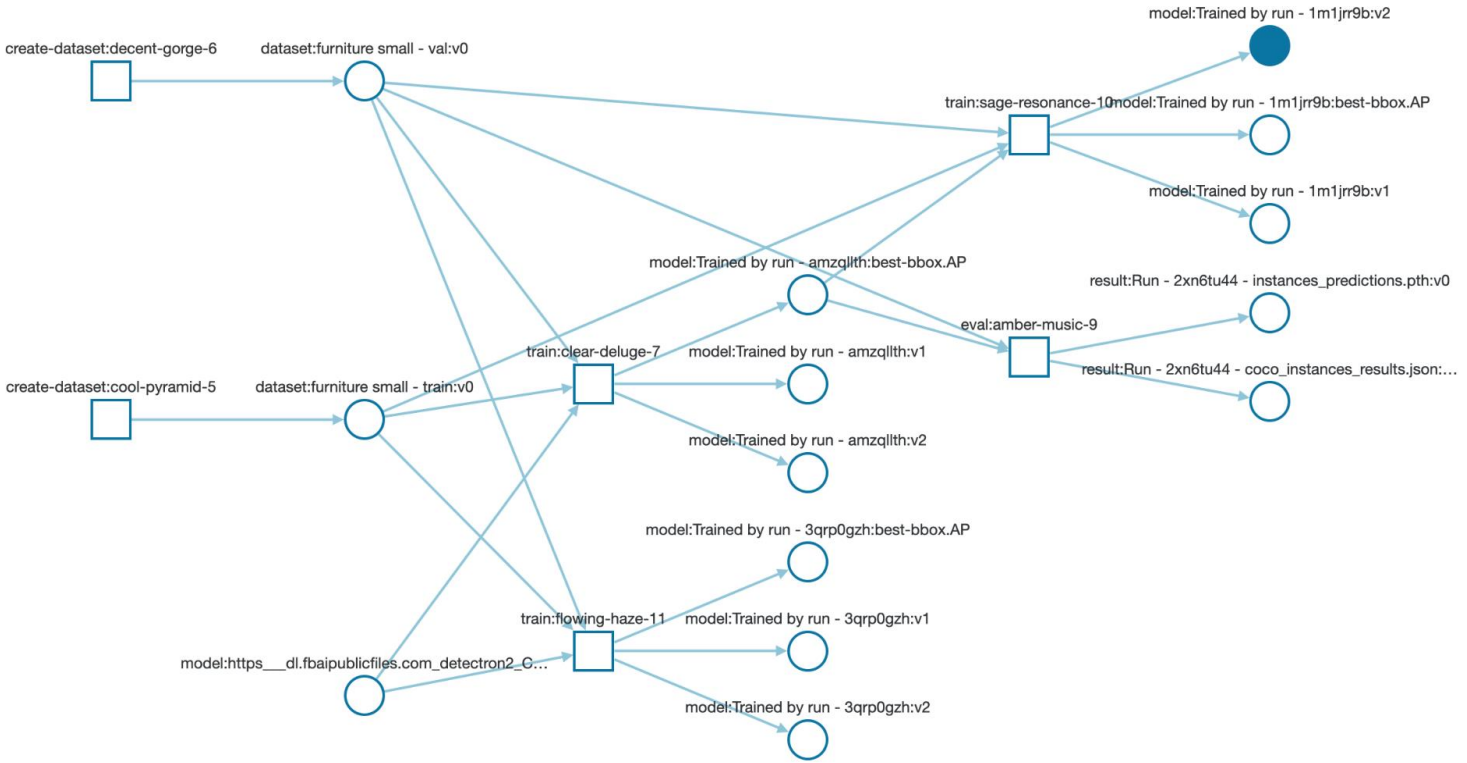
W&B also saves a patch file with uncommitted changes so you can reproduce the exact code that trained the model.

bit.ly/demo-run

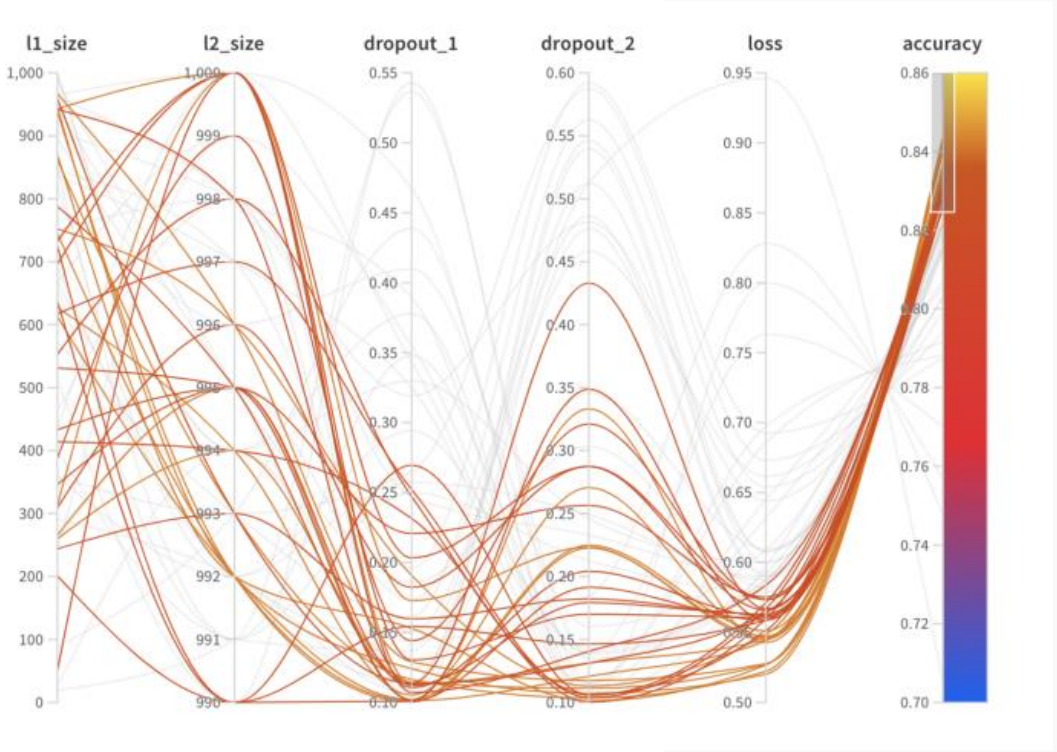


Track artifacts

Track artifacts



Compare runs

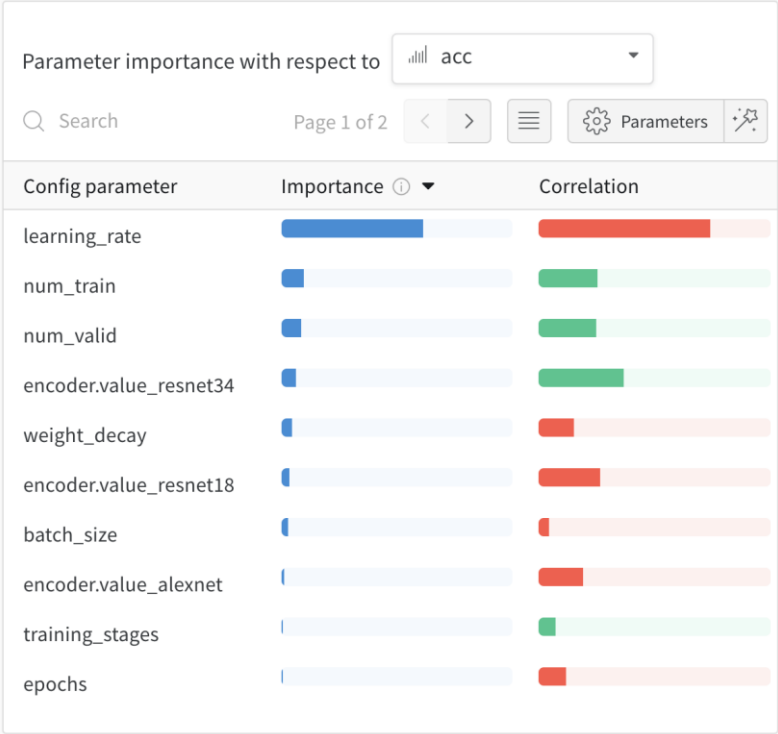


https://towardsdatascience.com/machine-learning-experiment-tracking-93b796e501b0

Compare runs

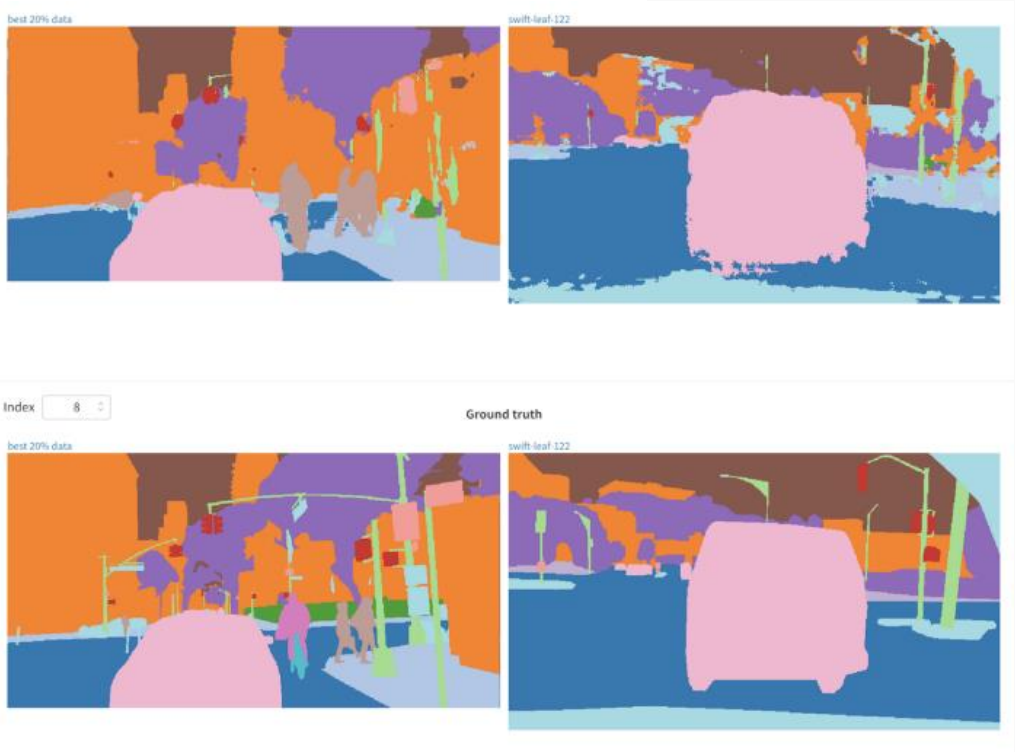
Visualize the relationships between hyperparameters and model metrics.

Explore the space of possible models quickly, without getting bogged down setting up manual visualizations.



bit.ly/deep-drive

Viewing Specific Examples



https://towardsdatascience.com/machine-learning-experiment-tracking-93b796e501b0

Persistent system of record

Query and filter across runs to keep projects organized.

Runs (395)

3 Filters | Group | Sort | Tag | Move | Columns

Name (228 visualized)	Tags	Runtime	batch_size	encoder	learning_ra	num_train	num_valid	weight_dec	iou	train_loss	valid_loss	acc	traffic_acc	road
best car acc (50% data)	seg_masks	47m 52s	6	resnet34	0.001311	3524	492	0.08173	0.7997	0.5375	0.4427	0.8823	0.8664	0.93
best traffic acc (50% data))	seg_masks	46m 42s	8	resnet18	0.001	3523	492	0.097	0.8073	0.4919	0.4203	0.888	0.8718	0.94
best human iou (50% data)	seg_masks	31m 34s	7	alexnet	0.0009084	1405	190	0.097	0.716	0.6222	0.6259	0.8334	0.8042	0.9
best overall IOU (20% data)	seg_masks	20m 23s	7	resnet34	0.001367	1376	205	0.06731	0.7948	0.5096	0.4659	0.8726	0.8593	0.93
vibrant-cherry-426		6m 12s	8	resnet34	0.001	347	42	0.097	0.752	0.7114	0.6055	0.8474	0.8282	0.95
worldly-totem-422		12m 54s	8	resnet34	0.001	682	97	0.097	0.7523	0.5774	0.5836	0.8566	0.8349	0.91
jumping-voice-421		11m 59s	8	resnet34	0.001	725	92	0.097	0.7449	0.5633	0.5334	0.8504	0.8296	0.91
logical-energy-420	test_only	2m 14s	8	resnet34	0.001	66	10	0.097	0.4297	1.459	1.221	0.626	0.5958	0.76

bit.ly/deep-drive

Hyperparameter Sweeping

- Hyperparameter sweep is **a way of finding the best hyperparameters for a model, given a set of data**.

- This is done by automatically searching through combinations of hyperparameter values (e.g. learning rate, batch size, number of hidden layers, optimizer type) to find the most optimal values.

Hyperparameter Sweeping

1. **Define the sweep:** Create a YAML file that specifies the parameters to search through, the search strategy and the optimization metric.

2. **Initialize the sweep:** Initialize the sweep and pass in the dictionary of sweep configurations: "sweep_id = wandb.sweep(sweep_config)“

3. **Run the sweep agent:** Call wandb.agent() and pass the sweep_id to run, along with a function that defines your model architecture and trains it: "wandb.agent(sweep_id, function=train)"

https://wandb.ai/site/articles/introduction-hyperparameter-sweeps

Hyperparameter Sweeping

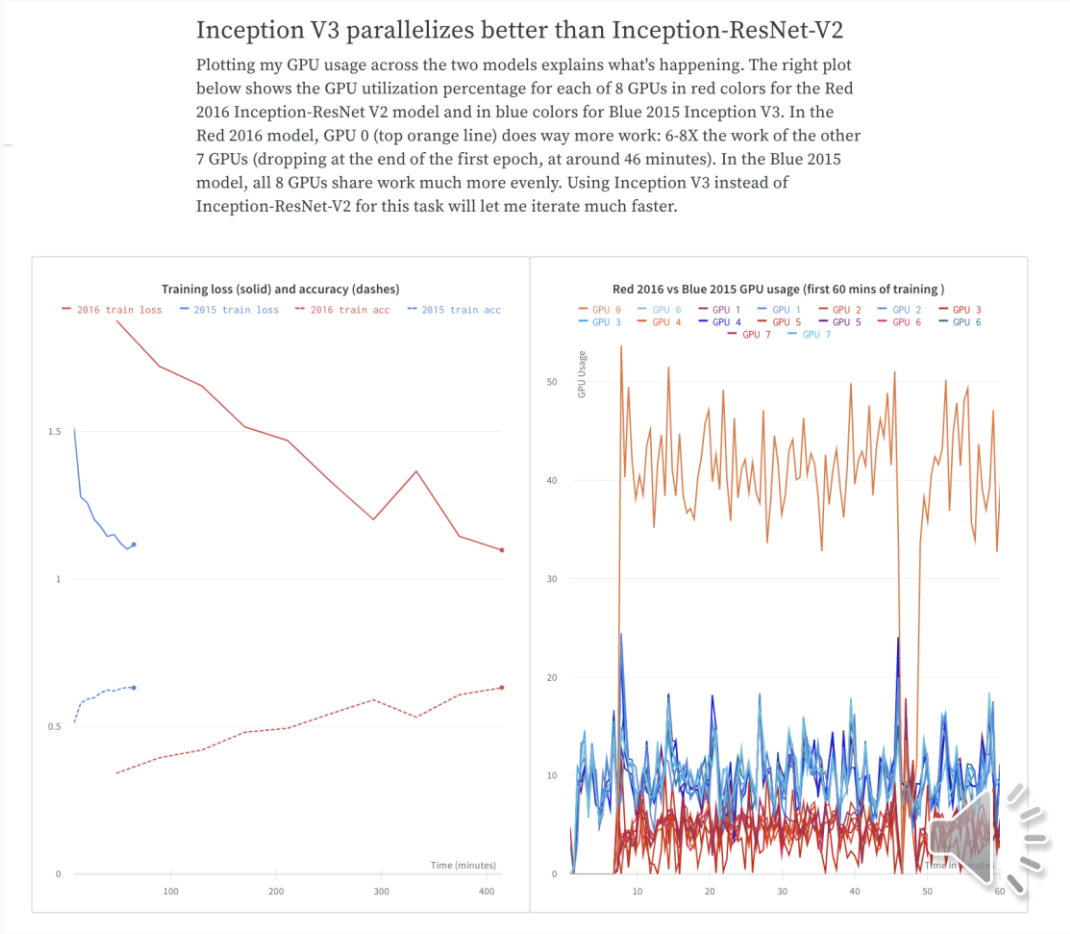
`method`	Description
`grid`	Grid search iterates over all possible combinations of parameter values.
`random`	Random search chooses a random set of values on each iteration.
`bayes`	Our Bayesian hyperparameter search method uses a Gaussian Process to model the relationship between the parameters and the model metric and chooses parameters to optimize the probability of improvement. This strategy requires the `metric` key to be specified.

Collaborative Reports

Reports let you organize and embed visualizations, describe your findings, share updates with collaborators.

Notes: Add a graph with a quick note to yourself.
Collaboration: Share findings with your colleagues.
Work log: Track what you've tried and plan next steps

bit.ly/deep-drive-report



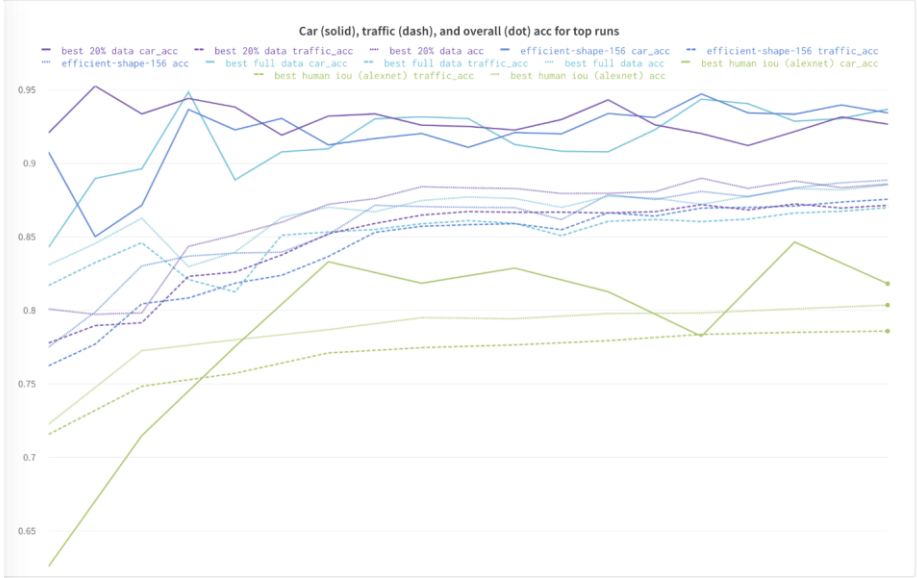
REPORTS

Live dashboards

Customize real-time views of model training and evaluation, and share auto-updating dashboards with your team to keep everyone in the loop.

bit.ly/deep-drive-report

Comparing per-class accuracies



Four top models 4

Name (4 visualized)	acc	car_acc	traffic_acc	train_loss	valid_loss	encoder	learning_ra	num_train	nur
best 20% data	0.8866	0.931	0.8724	0.4421	0.4284	resnet18	0.001	1421	183
efficient-shape-156	0.8863	0.9329	0.8739	0.4321	0.4394	resnet18	0.001	1409	20
best full data	0.8848	0.9356	0.8697	0.5218	0.4242	resnet18	0.001	700	100
best human iou (alexnet)	0.8036	0.8181	0.7859	0.6373	0.708	alexnet	0.0009084	699	10

1-4 of 4

Experiment Tracking In Practice OpenAI

- Reports
  - Every experiment the OpenAI team tries starts as a Report
  - “The ability to mix real data from experiments with context and commentary on the results was the primary selling point for us”
  - From Google docs with screenshots to interactive reports.

- Workflow
  - Create a new report with text at the top explaining the context, hypotheses under test, and experimental plan.
  - Launch one or more experiments aimed at testing the hypotheses.
  - Monitor the experiments. If enough data has been collected after the first set of experiments, add a conclusion to the top and share with the team. Otherwise, update the hypotheses and experimental plan and return to step 2.

https://wandb.ai/openai/published-work/Learning-Dexterity-End-to-End--VmlldzoxMTUyMDQ