Machine Learning Experiment Tracking

Why is experiment tracking so important for doing real world machine learning?

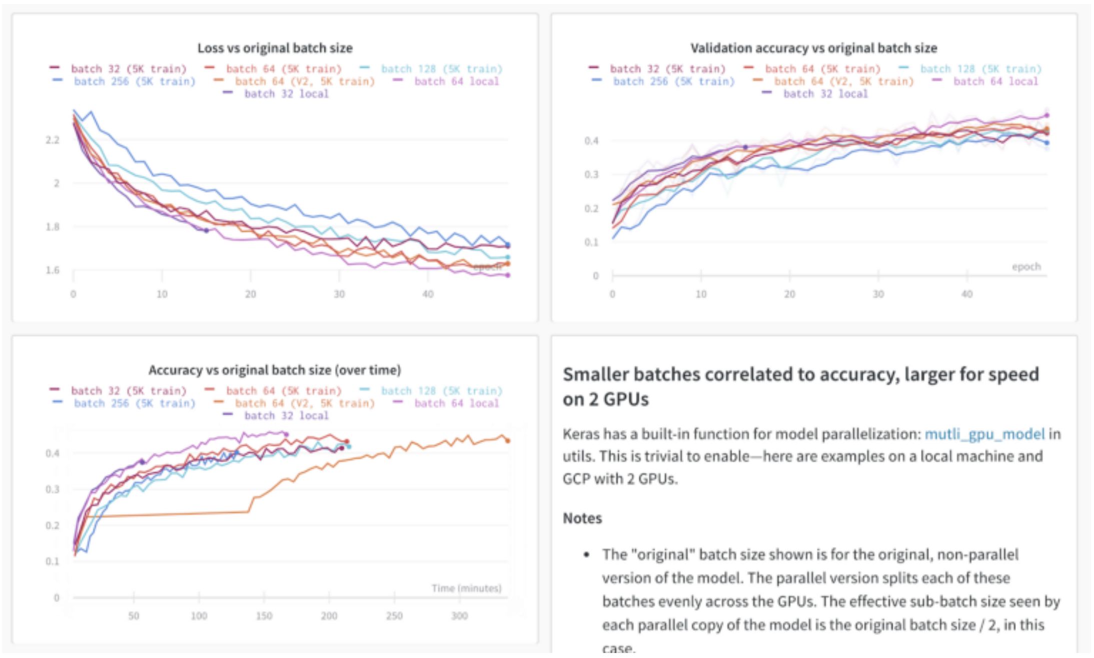
Lukas Biewald Mar 23, 2020 · 6 min read ★

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:

1. Machine Learning projects have far more branching and experimentation than a typical software project.
2. Machine Learning code generally doesn't throw errors, it just underperforms, making debugging extra difficult and time consuming.
3. 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.
4. Running machine learning experiments can be time consuming and just the compute costs can get expensive.

Tracking experiments in an organized way helps with all of these core issues. Weights and Biases (wandb) is a simple tool that helps individuals to track their experiments — I talked to several machine learning leaders of different size teams about how they use wandb to track their experiments.

Getting started with experiment tracking with wandb



Smaller batches correlated to accuracy, larger for speed on 2 GPUs

Keras has a built-in function for model parallelization: mutli_gpu_model in utils. This is trivial to enable—here are examples on a local machine and GCP with 2 GPUs.

Notes

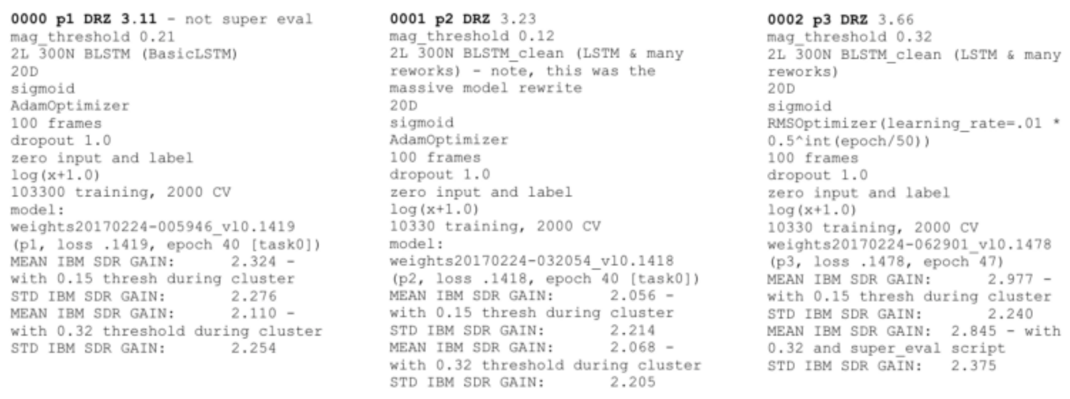
- The "original" batch size shown is for the original, non-parallel version of the model. The parallel version splits each of these batches evenly across the GPUs. The effective sub-batch size seen by each parallel copy of the model is the original batch size / 2, in this case.

View live dashboard

The essential unit of progress in an ML project is an experiment, so most people track what they're doing somehow — generally I see practitioners start with a spreadsheet or a text file to keep track of what they're doing.

Spreadsheets and docs are incredibly flexible — what's wrong with this approach?

Here is a Google doc I was using for a project a few years ago:



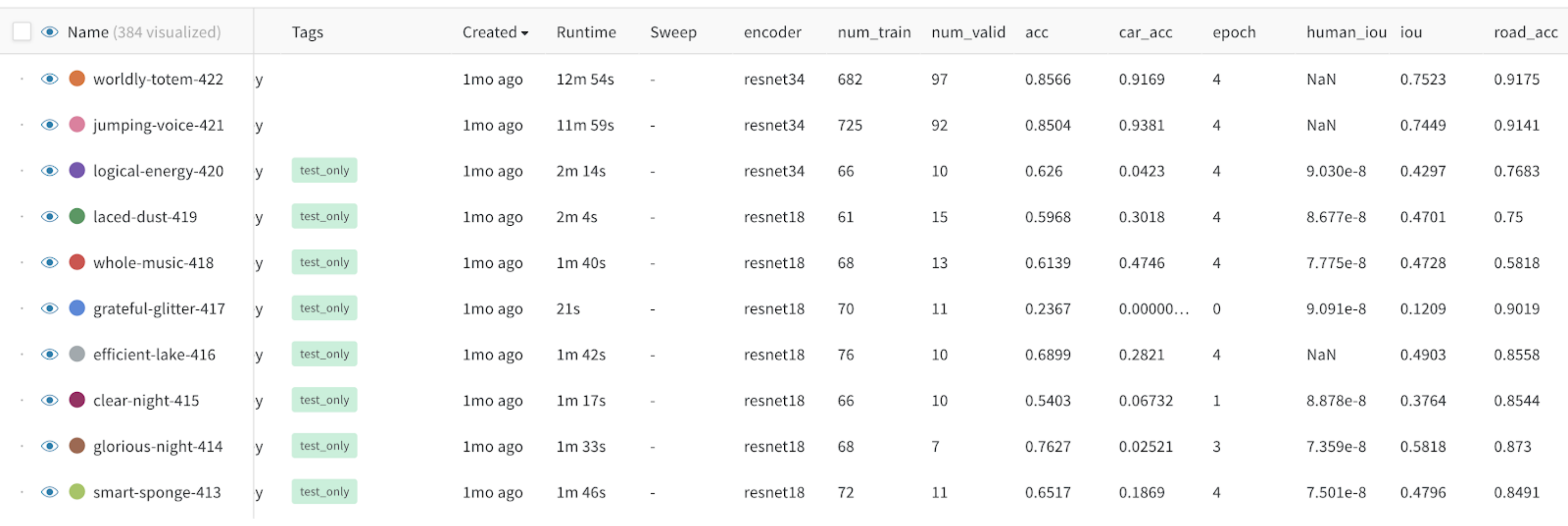
```
0000 p1 DRZ 3.11 - not super eval
mag_threshold 0.21
2L 300N BLSTM (BasicLSTM)
20D
sigmoid
AdamOptimizer
100 frames
dropout 1.0
zero input and label
log(x+1.0)
103300 training, 2000 CV
model:
weights20170224-005946_v10.1419
(p1, loss .1419, epoch 40 [task0])
MEAN IBM SDR GAIN:        2.324 -
with 0.15 thresh during cluster
STD IBM SDR GAIN:         2.276
MEAN IBM SDR GAIN:        2.110 -
with 0.32 threshold during cluster
STD IBM SDR GAIN:         2.254
```

```
0001 p2 DRZ 3.23
mag_threshold 0.12
2L 300N BLSTM_clean (LSTM & many
reworks) - note, this was the
massive model rewrite
20D
sigmoid
AdamOptimizer
100 frames
dropout 1.0
zero input and label
log(x+1.0)
10330 training, 2000 CV
model:
weights20170224-032054_v10.1418
(p2, loss .1418, epoch 40 [task0])
MEAN IBM SDR GAIN:        2.056 -
with 0.15 thresh during cluster
STD IBM SDR GAIN:         2.214
MEAN IBM SDR GAIN:        2.068 -
with 0.32 threshold during cluster
STD IBM SDR GAIN:         2.205
```

```
0002 p3 DRZ 3.66
mag_threshold 0.32
2L 300N BLSTM_clean (LSTM & many
reworks)
20D
sigmoid
RMSOptimizer(learning_rate=.01 *
0.5^int(epoch/50))
100 frames
dropout 1.0
zero input and label
log(x+1.0)
10330 training, 2000 CV
weights20170224-062901_v10.1478
(p3, loss .1478, epoch 47)
MEAN IBM SDR GAIN:        2.977 -
with 0.15 thresh during cluster
STD IBM SDR GAIN:         2.240
MEAN IBM SDR GAIN:  2.845 - with
0.32 and super_eval script
STD IBM SDR GAIN:   2.375
```

I'm sure these notes were important at the time, but now I have no idea what these notes mean.

Weights and Biases makes it very simple to automatically record all of the hyperparameters (inputs) and metrics (outputs).



Name (384 visualized)		Tags	Created	Runtime	Sweep	encoder	num_train	num_valid	acc	car_acc	epoch	human_iou	iou	road_acc
worldly-totem-422	y		1mo ago	12m 54s	-	resnet34	682	97	0.8566	0.9169	4	NaN	0.7523	0.9175
jumping-voice-421	y		1mo ago	11m 59s	-	resnet34	725	92	0.8504	0.9381	4	NaN	0.7449	0.9141
logical-energy-420	y	test_only	1mo ago	2m 14s	-	resnet34	66	10	0.626	0.0423	4	9.030e-8	0.4297	0.7683
laced-dust-419	y	test_only	1mo ago	2m 4s	-	resnet18	61	15	0.5968	0.3018	4	8.677e-8	0.4701	0.75
whole-music-418	y	test_only	1mo ago	1m 40s	-	resnet18	68	13	0.6139	0.4746	4	7.775e-8	0.4728	0.5818
grateful-glitter-417	y	test_only	1mo ago	21s	-	resnet18	70	11	0.2367	0.00000…	0	9.091e-8	0.1209	0.9019
efficient-lake-416	y	test_only	1mo ago	1m 42s	-	resnet18	76	10	0.6899	0.2821	4	NaN	0.4903	0.8558
clear-night-415	y	test_only	1mo ago	1m 17s	-	resnet18	66	10	0.5403	0.06732	1	8.878e-8	0.3764	0.8544
glorious-night-414	y	test_only	1mo ago	1m 33s	-	resnet18	68	7	0.7627	0.02521	3	7.359e-8	0.5818	0.873
smart-sponge-413	y	test_only	1mo ago	1m 46s	-	resnet18	72	11	0.6517	0.1869	4	7.501e-8	0.4796	0.8491

A typical project in wandb.

This is how you would setup wandb in pytorch (you can find other common ML frameworks in the documentation).

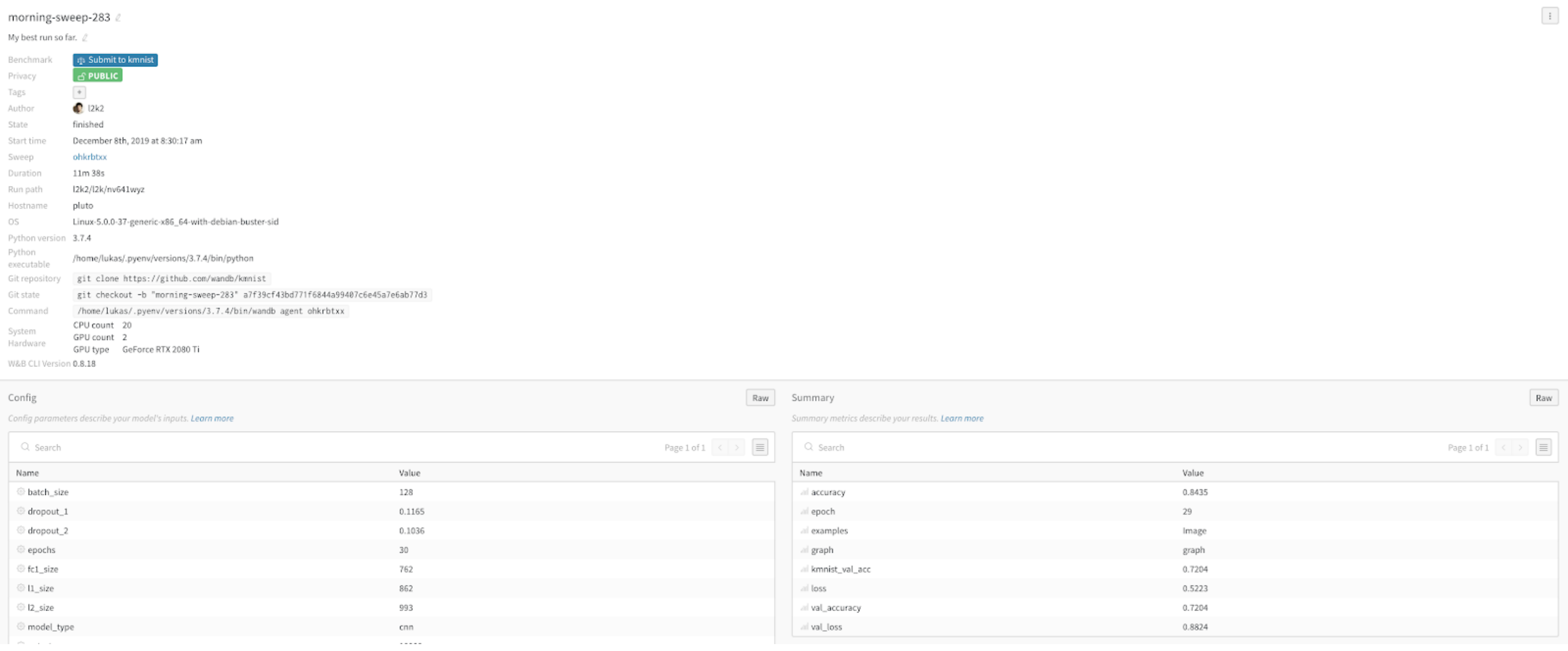
```
import wandb

wandb.init(config=args) # track hyperparameters
wandb.watch(model) # track model metrics


model.train()
for batch_idx, (data, target) in enumerate(train_loader):
  output = model(data)
  loss = F.nll_loss(output, target)
  loss.backward()
  optimizer.step()


  wandb.log({“loss”: loss}) # track a specific metric
```

Once set up, Weights and Biases monitors a lot of things by default. Any command line argument becomes a saved hyperparameter. Any value made available by pytorch becomes a metric. The experiment can automatically be linked to the latest git commit or the exact state of the training code. Collecting information passively is really important because it’s nearly impossible to consistently write down all the things you might care about.


morning-sweep-283

My best run so far.

Benchmark	Submit to kmnist
Privacy	PUBLIC
Tags	+
Author	l2k2
State	finished
Start time	December 8th, 2019 at 8:30:17 am
Sweep	ohkrbtxx
Duration	11m 38s
Run path	l2k2/l2k/nv641wyz
Hostname	pluto
OS	Linux-5.0.0-37-generic-x86_64-with-debian-buster-sid
Python version	3.7.4
Python executable	/home/lukas/.pyenv/versions/3.7.4/bin/python
Git repository	`git clone https://github.com/wandb/kmnist`
Git state	`git checkout -b "morning-sweep-283" a7f39cf43bd771f6844a99407c6e45a7e6ab77d3`
Command	`/home/lukas/.pyenv/versions/3.7.4/bin/wandb agent ohkrbtxx`
System Hardware	CPU count 20 GPU count 2 GPU type GeForce RTX 2080 Ti
W&B CLI Version	0.8.18

Config

Config parameters describe your model's inputs. Learn more

Name	Value
batch_size	128
dropout_1	0.1165
dropout_2	0.1036
epochs	30
fc1_size	762
l1_size	862
l2_size	993
model_type	cnn

Summary

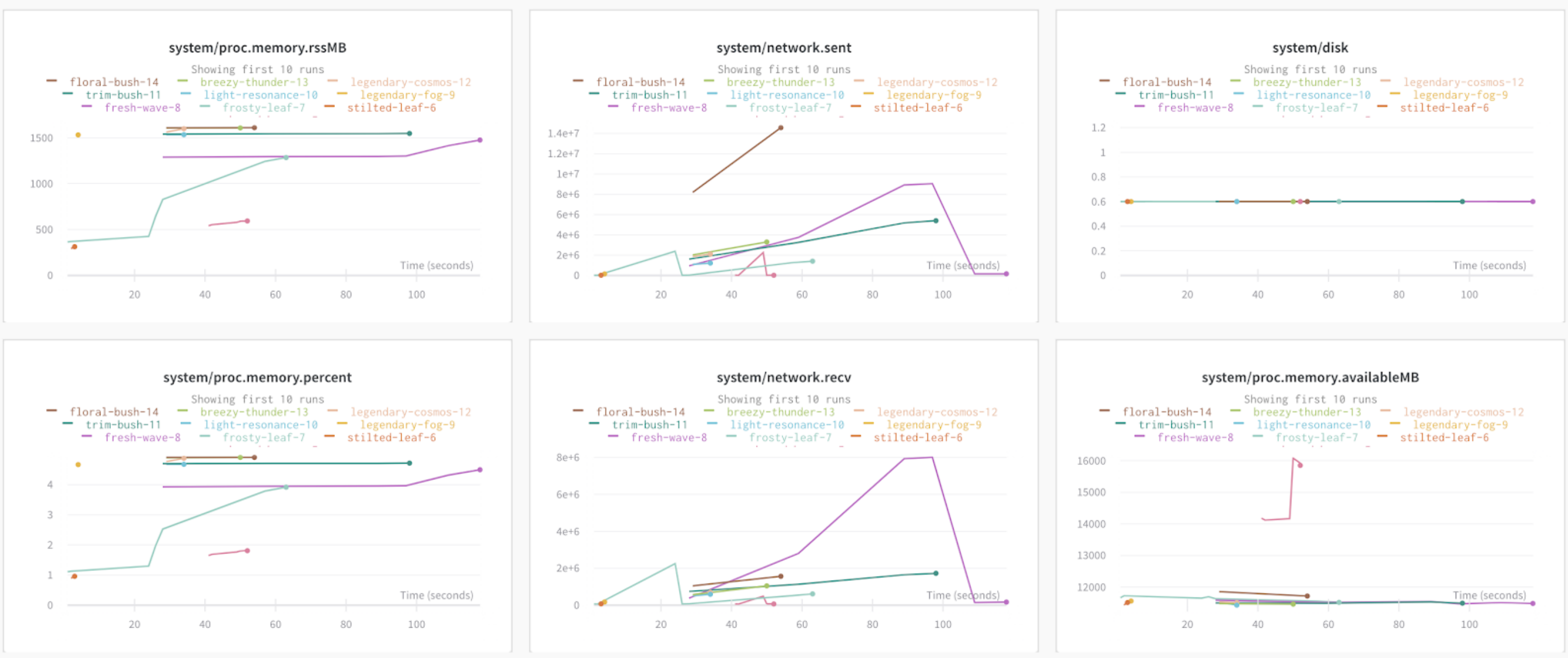
Summary metrics describe your results. Learn more

Name	Value
accuracy	0.8435
epoch	29
examples	Image
graph	graph
kmnist_val_acc	0.7204
loss	0.5223
val_accuracy	0.7204
val_loss	0.8824

Experiment overview in Weights & Biases

It's also extremely important to write down qualitative notes about the models you build for later. In Weights and Biases, you can create reports to track notes alongside model metrics, and to share your findings and progress with your team.

Collecting system metrics in the background is a good example. Wandb collects system usage metrics in the background — things like GPU Memory Allocated, Network Traffic and Disk Usage. Most of the time you don't need to look at all of this information, but it's easy to make a change where you are no longer using most of your GPUs memory and hard to track down when you made that change. If you instrument your training code with wandb once, you will be able to look back at all of your experiments and see where the usage changed.



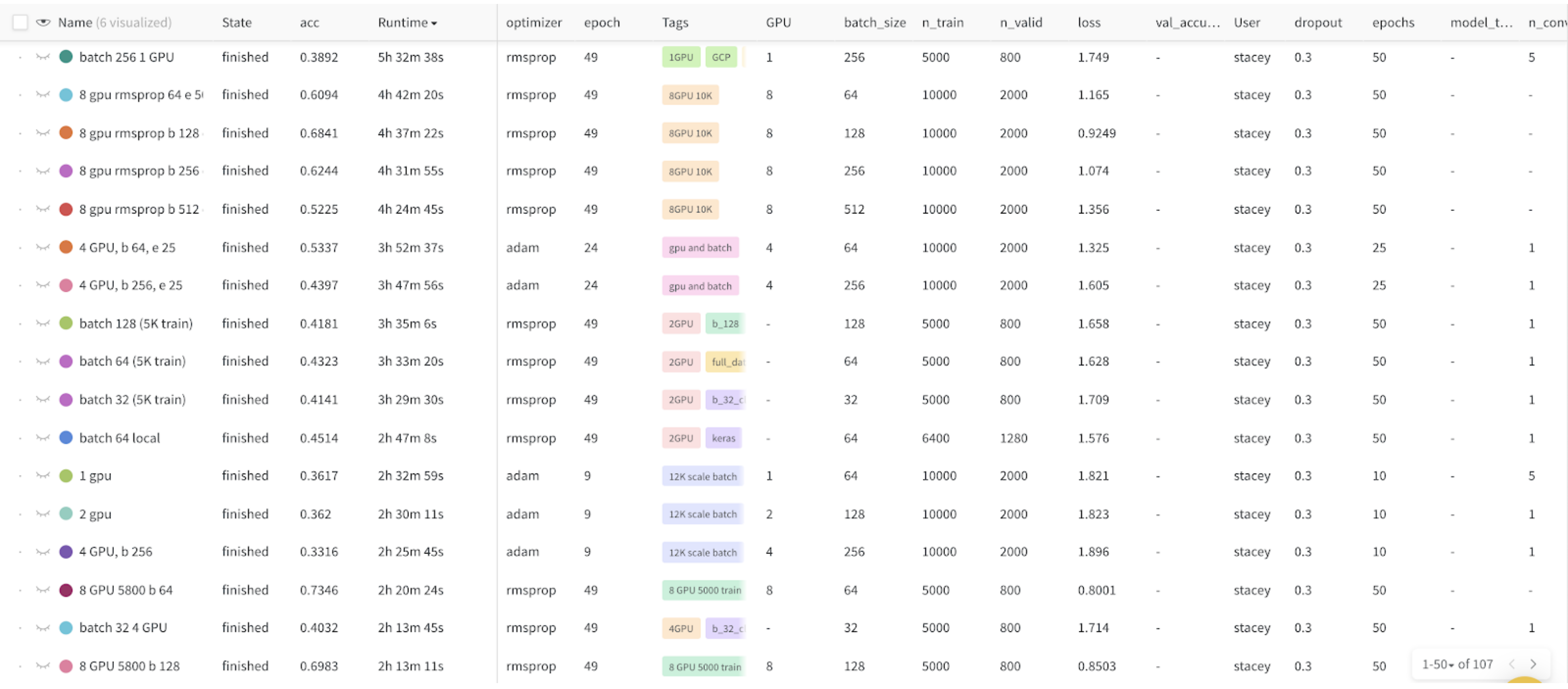
View in live dashboard

Using experiment tracking to compare results across experiments

A typical ML workflow involves running lots of experiments. We've found that looking at results in the context of other results is much more meaningful than looking at a single experiment alone.

Looking across lots of experiments at once gets messy quickly. There are lots of inputs changing and lots of different possible outputs. Some runs inevitably fail early.

Different experimentation styles lead to different workflows, but we've found that logging every metric that you might care about and tagging experiments with a few consistent tags meaningful to you can keep things much more organized later.

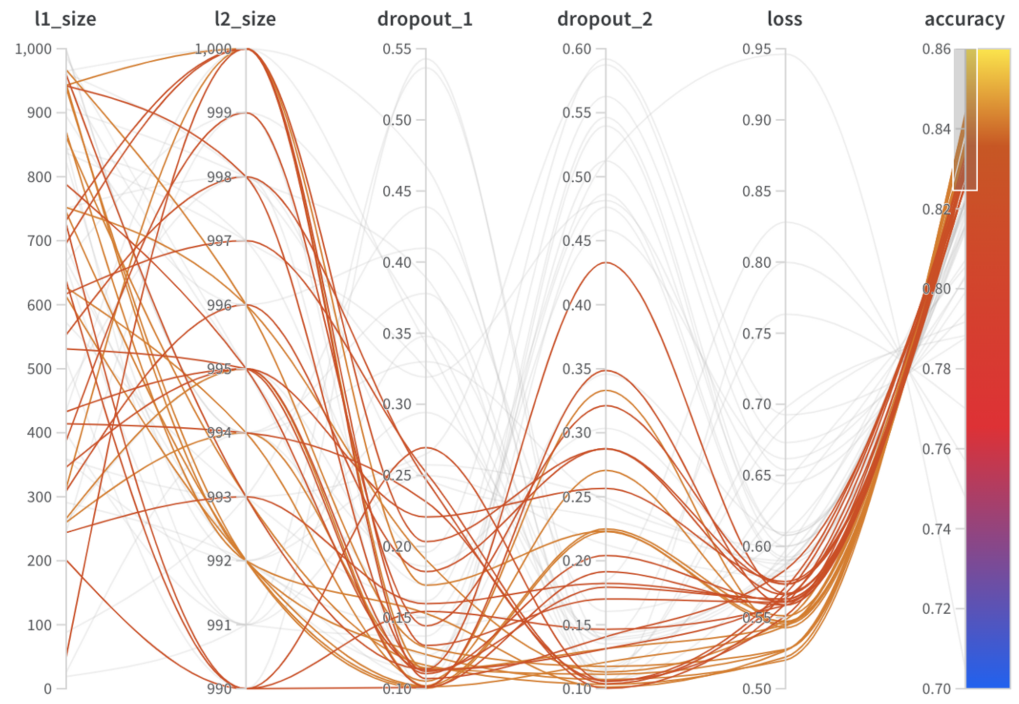


Name (6 visualized)	State	acc	Runtime	optimizer	epoch	Tags	GPU	batch_size	n_train	n_valid	loss	val_accu…	User	dropout	epochs	model_t…	n_conv
batch 256 1 GPU	finished	0.3892	5h 32m 38s	rmsprop	49	1GPU GCP	1	256	5000	800	1.749	-	stacey	0.3	50	-	5
8 gpu rmsprop 64 e 5	finished	0.6094	4h 42m 20s	rmsprop	49	8GPU 10K	8	64	10000	2000	1.165	-	stacey	0.3	50	-	-
8 gpu rmsprop b 128	finished	0.6841	4h 37m 22s	rmsprop	49	8GPU 10K	8	128	10000	2000	0.9249	-	stacey	0.3	50	-	-
8 gpu rmsprop b 256	finished	0.6244	4h 31m 55s	rmsprop	49	8GPU 10K	8	256	10000	2000	1.074	-	stacey	0.3	50	-	-
8 gpu rmsprop b 512	finished	0.5225	4h 24m 45s	rmsprop	49	8GPU 10K	8	512	10000	2000	1.356	-	stacey	0.3	50	-	-
4 GPU, b 64, e 25	finished	0.5337	3h 52m 37s	adam	24	gpu and batch	4	64	10000	2000	1.325	-	stacey	0.3	25	-	1
4 GPU, b 256, e 25	finished	0.4397	3h 47m 56s	adam	24	gpu and batch	4	256	10000	2000	1.605	-	stacey	0.3	25	-	1
batch 128 (5K train)	finished	0.4181	3h 35m 6s	rmsprop	49	2GPU b_128	-	128	5000	800	1.658	-	stacey	0.3	50	-	1
batch 64 (5K train)	finished	0.4323	3h 33m 20s	rmsprop	49	2GPU full_da	-	64	5000	800	1.628	-	stacey	0.3	50	-	1
batch 32 (5K train)	finished	0.4141	3h 29m 30s	rmsprop	49	2GPU b_32_	-	32	5000	800	1.709	-	stacey	0.3	50	-	1
batch 64 local	finished	0.4514	2h 47m 8s	rmsprop	49	2GPU keras	-	64	6400	1280	1.576	-	stacey	0.3	50	-	1
1 gpu	finished	0.3617	2h 32m 59s	adam	9	12K scale batch	1	64	10000	2000	1.821	-	stacey	0.3	10	-	5
2 gpu	finished	0.362	2h 30m 11s	adam	9	12K scale batch	2	128	10000	2000	1.823	-	stacey	0.3	10	-	1
4 GPU, b 256	finished	0.3316	2h 25m 45s	adam	9	12K scale batch	4	256	10000	2000	1.896	-	stacey	0.3	10	-	1
8 GPU 5800 b 64	finished	0.7346	2h 20m 24s	rmsprop	49	8 GPU 5000 train	8	64	5000	800	0.8001	-	stacey	0.3	50	-	-
batch 32 4 GPU	finished	0.4032	2h 13m 45s	rmsprop	49	4GPU b_32_	-	32	5000	800	1.714	-	stacey	0.3	50	-	1
8 GPU 5800 b 128	finished	0.6983	2h 13m 11s	rmsprop	49	8 GPU 5000 train	8	128	5000	800	0.8503	-	stacey	0.3	50		

1-50 of 107

View in live dashboard.

Once you have a number of models logged, you have way more dimensions to examine than can be looked at all at once. One powerful visualization tool we've discovered is the parallel coordinates chart.



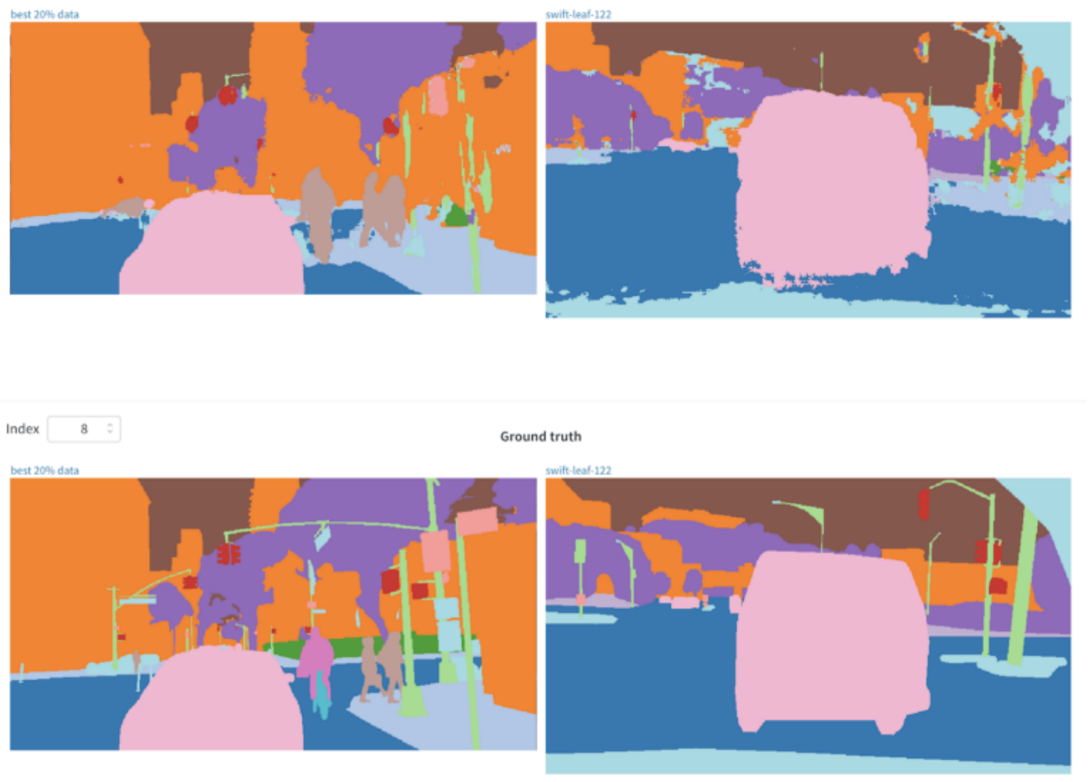
View live dashboard

Here each line is an individual experiment and each column is an input hyperparameter or an output metric. I've highlighted the top accuracy runs and it shows quite clearly that across all of my experiments that I've selected, high accuracy comes from low dropout values.

Looking across experiments is so important that wandb lets you build workspaces where you can select groups of graphs in visualizations like a scatterplot and then immediately view comparisons of the selected runs

Viewing specific examples

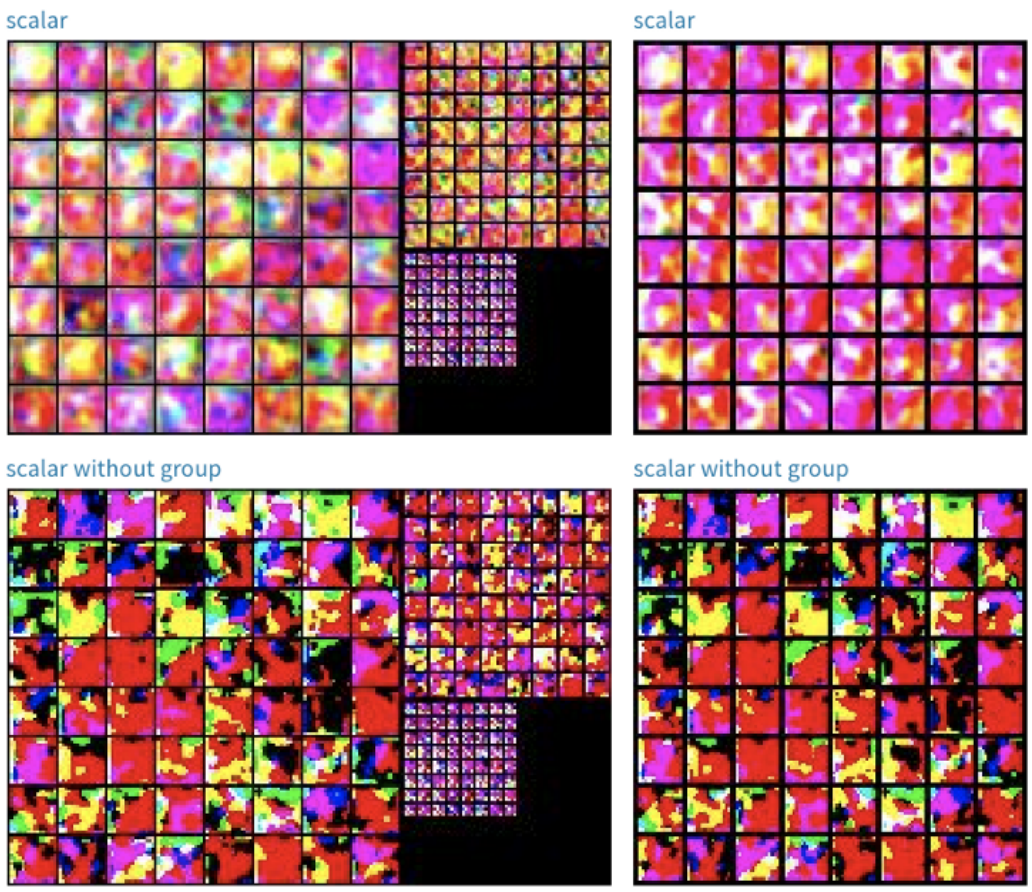
Aggregate metrics are good, but it is essential to look at specific examples. The function wandb.log() can handle all kinds of datatypes and automatically visualizes them.



View live report

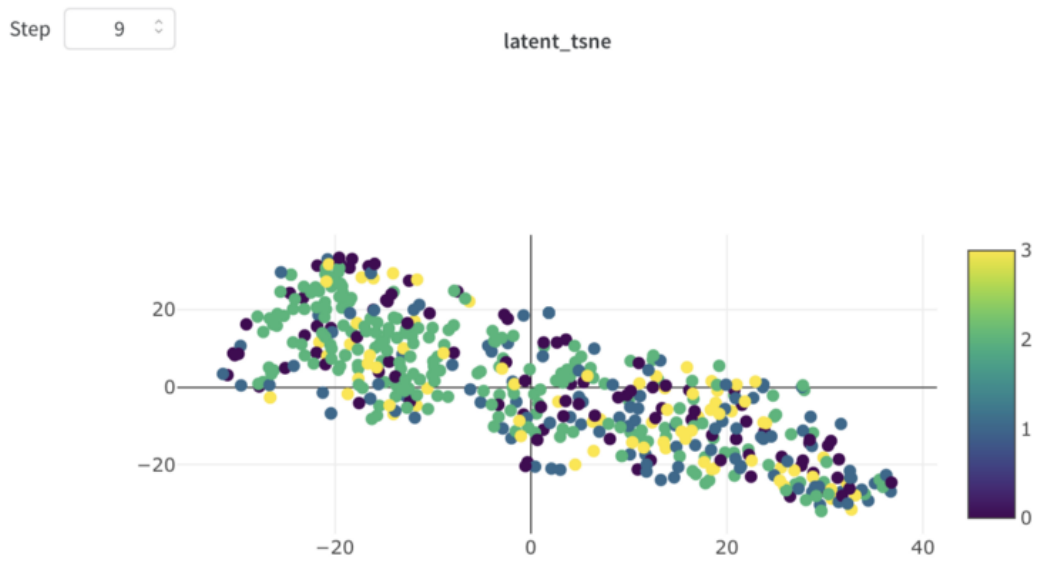
Logging Images

Logging images is important for many applications and it’s possible to see images across multiple runs. Here are different approaches to building a GAN and the results at various scales and timesteps.



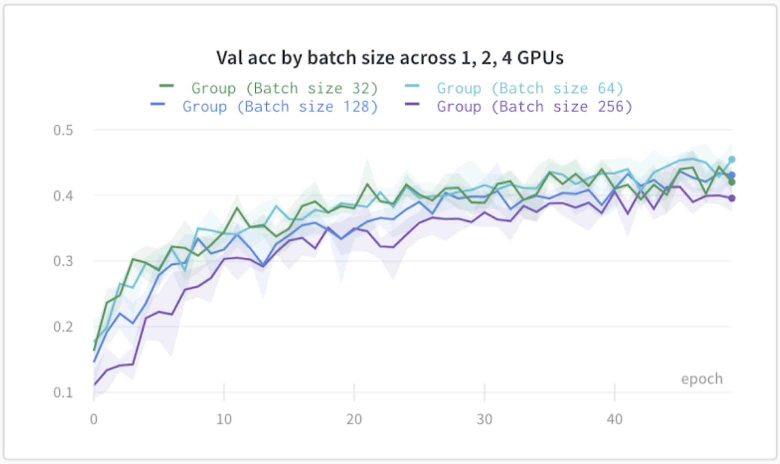
Logging Matplotlib Plots

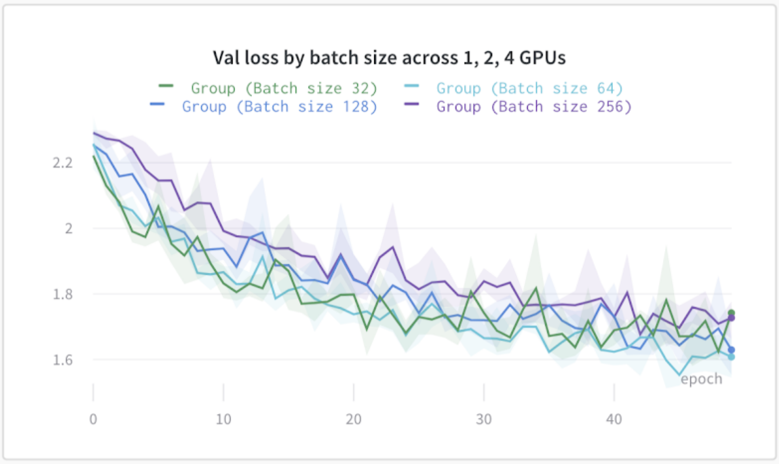
Often code already tracks things in matplotlib — if you log the chart it will be saved forever and easy to pull back up. In fact you can log a unique chart for every step of your training code.

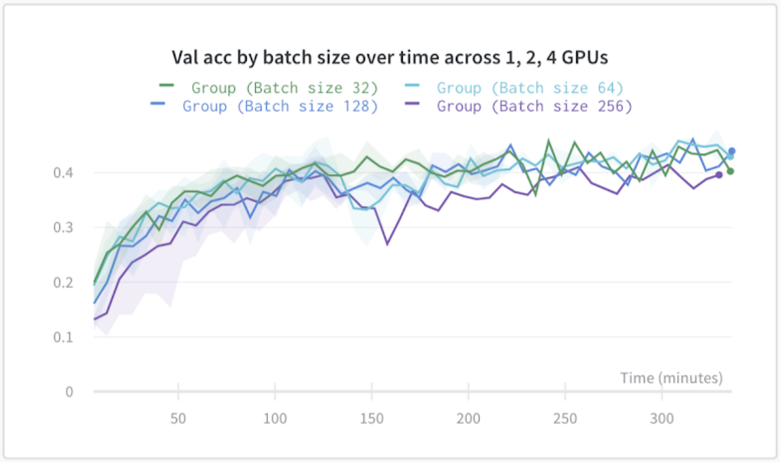


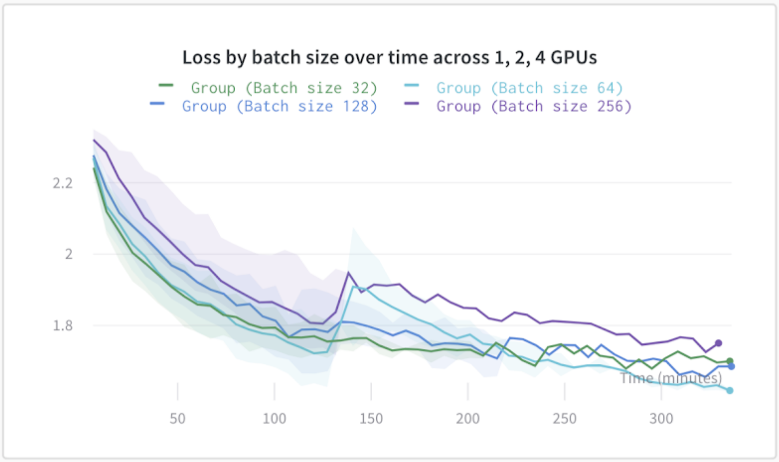
Using experiment tracking to manage distributed training

When doing distributed training, even just visualizing results can get even harder.









View live dashboard

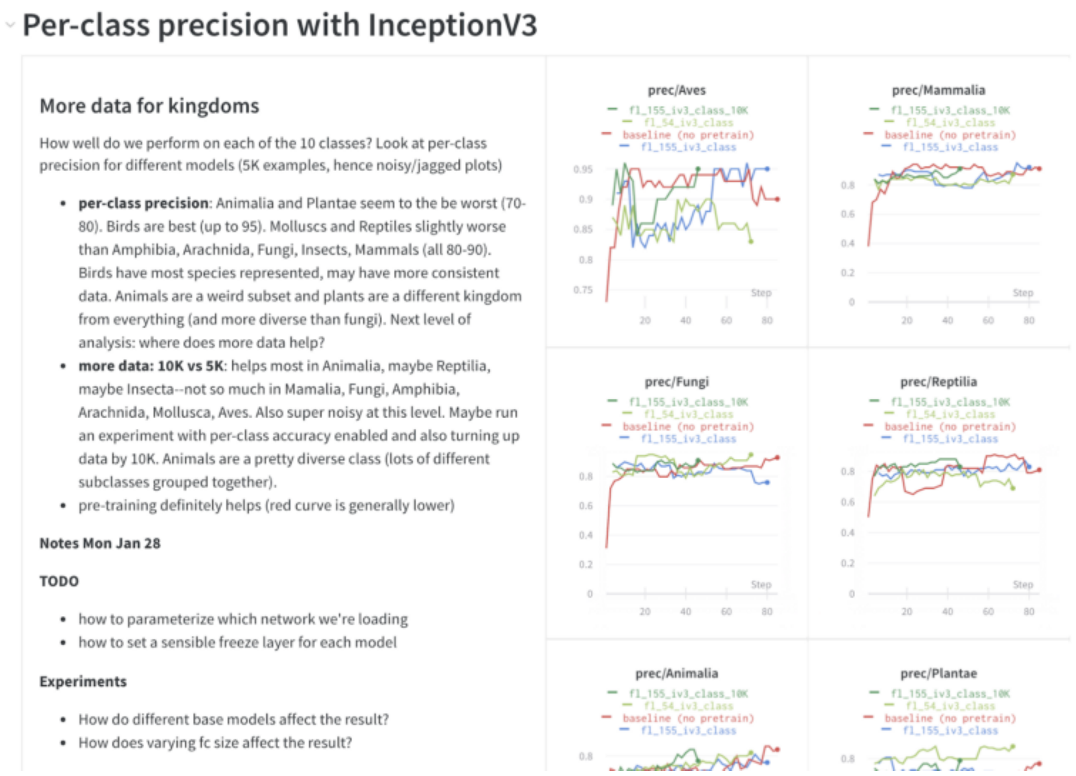
Wandb will show the metrics for all of the runs in the group aggregated together, but it's also possible to go in and look at the individual process and see how well they perform.

Using experiment tracking reports to manage team collaboration

As teams grow, tracking everything becomes more and more important. Wandb lets you build static reports that show exactly the experiments you've run with aggregate statistics and the ability to dig deeper.

On the OpenAI robotics team, wandb reports are the place where ML practitioners record the work they've done and share it with their colleagues. It is crucial for visualizing when a change may have inadvertently hurt progress.

At Latent Space, every team project meeting starts with a review of the latest wandb experiment report and a discussion around how well the current approach is working and what experiments should be tried next.



Per-class precision with InceptionV3

More data for kingdoms

How well do we perform on each of the 10 classes? Look at per-class precision for different models (5K examples, hence noisy/jagged plots)

- **per-class precision**: Animalia and Plantae seem to the be worst (70-80). Birds are best (up to 95). Molluscs and Reptiles slightly worse than Amphibia, Arachnida, Fungi, Insects, Mammals (all 80-90). Birds have most species represented, may have more consistent data. Animals are a weird subset and plants are a different kingdom from everything (and more diverse than fungi). Next level of analysis: where does more data help?
- **more data: 10K vs 5K**: helps most in Animalia, maybe Reptilia, maybe Insecta--not so much in Mamalia, Fungi, Amphibia, Arachnida, Mollusca, Aves. Also super noisy at this level. Maybe run an experiment with per-class accuracy enabled and also turning up data by 10K. Animals are a pretty diverse class (lots of different subclasses grouped together).
- pre-training definitely helps (red curve is generally lower)

Notes Mon Jan 28

TODO

- how to parameterize which network we're loading
- how to set a sensible freeze layer for each model

Experiments

- How do different base models affect the result?
- How does varying fc size affect the result?

View live report

Using experiment tracking as system of record for models

As teams grow and models become are deployed into production it becomes more and more important to have a record of everything that happened. At Toyota Research, the wandb experiment link is used as the official record of every ML model that gets built. If something happens downstream of a model build, they can trace the issue back to the wandb training run. Building a report from a set of experiments means there is a permanent record of the work done and teams can easily go back and review exactly

Machine Learning Tools Pytorch Keras Machine Learning Recipe