```
/*
    I deleted all the training log (train loss) since there were too many and ju
st leave the validation loss log
    It should be easier to see the train/validation loss graph to see the traini
ng summary
    Test set result is at the very bottom
    To be honest, I think I can get better result with better blocks_per_stage a
nd layer_depths (more like resnet-18 or resnet-34 that I used for CIFAR-100) but
 I didn't have time to re-train it.

    Started:        2025-10-04T23:02:32.478968Z
    Ended:          2025-10-05T03:43:35.804946Z
*/

{"settings": {"debug": false, "random_seed": 31415, "model": {"input_depth": 3,
"layer_depths": [16, 32, 64], "blocks_per_stage": [3, 3, 3], "num_classes": 10},
 "data": {"batch_size": 64, "validation_size": 5000, "shuffle_buffer": 10000}, "
training": {"num_epochs": 40, "learning_rate": 0.001, "weight_decay": 0.0001, "l
og_interval": 100, "final_test": false, "checkpoint_dir": "PosixPath('hw04/check
points')"}, "plotting": {"output_dir": "PosixPath('hw04/artifacts')", "figsize":
 [5, 3], "dpi": 200}}, "event": "Settings loaded", "logger": "hw04", "level": "i
nfo", "timestamp": "2025-10-04T23:01:52.118846Z"}
{"event": "Variant folder /root/tensorflow_datasets/cifar10/3.0.2 has no dataset
_info.json"}
{"event": "Generating dataset cifar10 (/root/tensorflow_datasets/cifar10/3.0.2)"
}
{"event": "Downloading https://www.cs.toronto.edu/~kriz/cifar-10-binary.tar.gz i
nto /root/tensorflow_datasets/downloads/cifar10/cs.toronto.edu_kriz_cifar-10-bin
aryODHPtIjLh3oLcXirEISTO7dkzyKjRCuol6lV8Wc6C7s.tar.gz.tmp.e1cbf29ba6684e91b56e41
462da23de0..."}
{"event": "Done writing /root/tensorflow_datasets/cifar10/incomplete.4A5IR3_3.0.
2/cifar10-train.tfrecord*. Number of examples: 50000 (shards: [50000])"}
{"event": "Done writing /root/tensorflow_datasets/cifar10/incomplete.4A5IR3_3.0.
2/cifar10-test.tfrecord*. Number of examples: 10000 (shards: [10000])"}
{"event": "Creating a tf.data.Dataset reading 1 files located in folders: /root/
tensorflow_datasets/cifar10/3.0.2."}
{"event": "Constructing tf.data.Dataset cifar10 for split train, from /root/tens
orflow_datasets/cifar10/3.0.2"}
{"event": "Load dataset info from /root/tensorflow_datasets/cifar10/3.0.2"}
{"event": "Creating a tf.data.Dataset reading 1 files located in folders: /root/
tensorflow_datasets/cifar10/3.0.2."}
{"event": "Constructing tf.data.Dataset cifar10 for split test, from /root/tenso
rflow_datasets/cifar10/3.0.2"}
{"event": "Unable to initialize backend 'tpu': INTERNAL: Failed to open libtpu.s
o: libtpu.so: cannot open shared object file: No such file or directory"}
{"num_params": 272938, "event": "Model created", "logger": "hw04", "level": "inf
o", "timestamp": "2025-10-04T23:02:32.478968Z"}
{"epoch": 1, "step": 704, "val_loss": 1.6888156540786163, "val_acc": 0.388844936
7088608, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "
timestamp": "2025-10-04T23:04:50.694858Z"}
{"epoch": 2, "step": 1408, "val_loss": 1.317224457294126, "val_acc": 0.517207278
4810127, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "
timestamp": "2025-10-04T23:07:19.472678Z"}
{"epoch": 3, "step": 2112, "val_loss": 1.2149084308479405, "val_acc": 0.55992879
74683544, "event": "epoch_summary", "logger": "hw04.training", "level": "info",
"timestamp": "2025-10-04T23:09:49.454928Z"}
{"epoch": 4, "step": 2816, "val_loss": 1.088192823566968, "val_acc": 0.612737341
7721519, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "
timestamp": "2025-10-04T23:43:07.676783Z"}
{"event": "Finished blocking save. Time taken: 0.397554s. Continuing background
save to /work/hw04/checkpoints/best_state_epoch_5."}
{"epoch": 5, "step": 3520, "val_loss": 0.9746726042107691, "val_acc": 0.65031645
56962026, "event": "epoch_summary", "logger": "hw04.training", "level": "info",
"timestamp": "2025-10-05T00:24:08.390924Z"}
{"epoch": 6, "step": 4224, "val_loss": 1.0020704873000519, "val_acc": 0.64477848
10126582, "event": "epoch_summary", "logger": "hw04.training", "level": "info",
"timestamp": "2025-10-05T01:12:26.285167Z"}
{"epoch": 7, "step": 4928, "val_loss": 0.8331410703779776, "val_acc": 0.71182753
16455697, "event": "epoch_summary", "logger": "hw04.training", "level": "info",
```



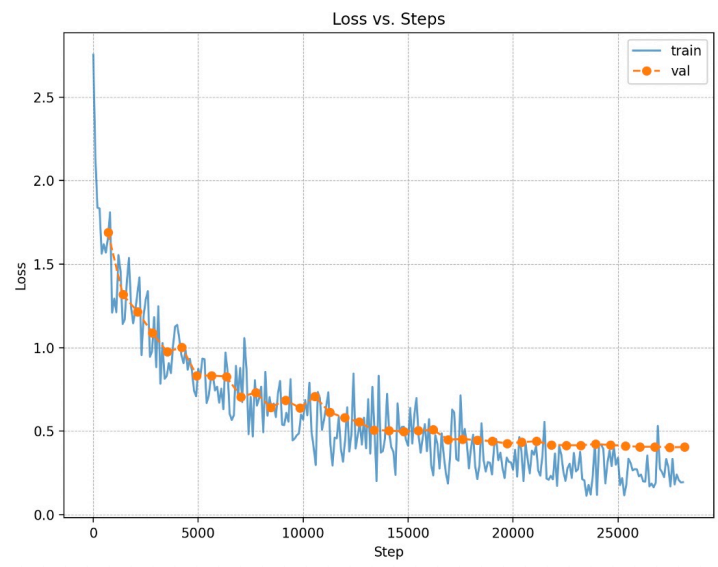
```
"timestamp": "2025-10-05T02:11:35.316912Z"}
{"epoch": 8, "step": 5632, "val_loss": 0.8326036997988254, "val_acc": 0.70431170
88607594, "event": "epoch_summary", "logger": "hw04.training", "level": "info",
"timestamp": "2025-10-05T02:23:58.950863Z"}
{"epoch": 9, "step": 6336, "val_loss": 0.8269730994973001, "val_acc": 0.70530063
29113924, "event": "epoch_summary", "logger": "hw04.training", "level": "info",
"timestamp": "2025-10-05T02:25:53.734331Z"}
{"event": "Finished blocking save. Time taken: 0.291334s. Continuing background
save to /work/hw04/checkpoints/best_state_epoch_10."}
{"epoch": 10, "step": 7040, "val_loss": 0.7053691990013364, "val_acc": 0.7565268
987341772, "event": "epoch_summary", "logger": "hw04.training", "level": "info",
 "timestamp": "2025-10-05T02:28:01.743996Z"}
{"epoch": 11, "step": 7744, "val_loss": 0.7308083606671684, "val_acc": 0.7488132
911392406, "event": "epoch_summary", "logger": "hw04.training", "level": "info",
 "timestamp": "2025-10-05T02:30:21.243522Z"}
{"epoch": 12, "step": 8448, "val_loss": 0.6420914360994026, "val_acc": 0.7759098
101265823, "event": "epoch_summary", "logger": "hw04.training", "level": "info",
 "timestamp": "2025-10-05T02:32:36.519162Z"}
{"epoch": 13, "step": 9152, "val_loss": 0.6851127589050727, "val_acc": 0.7650316
455696202, "event": "epoch_summary", "logger": "hw04.training", "level": "info",
 "timestamp": "2025-10-05T02:35:02.441586Z"}
{"epoch": 14, "step": 9856, "val_loss": 0.6394457692586923, "val_acc": 0.7806566
455696202, "event": "epoch_summary", "logger": "hw04.training", "level": "info",
 "timestamp": "2025-10-05T02:37:23.746143Z"}
{"epoch": 15, "step": 10560, "val_loss": 0.7071164020254642, "val_acc": 0.756329
1139240507, "event": "epoch_summary", "logger": "hw04.training", "level": "info"
, "timestamp": "2025-10-05T02:39:50.385917Z"}
{"event": "Finished async_save (blocking + background). Time taken: 0.507577s. d
irectory=/work/hw04/checkpoints/state_epoch_15"}
{"event": "[process=0][thread=async_save] Background save thread done. Time take
n: 0.422445s."}
{"epoch": 16, "step": 11264, "val_loss": 0.6119468215900131, "val_acc": 0.789952
5316455697, "event": "epoch_summary", "logger": "hw04.training", "level": "info"
, "timestamp": "2025-10-05T02:42:19.938507Z"}
{"epoch": 17, "step": 11968, "val_loss": 0.5802336259733273, "val_acc": 0.801424
0506329114, "event": "epoch_summary", "logger": "hw04.training", "level": "info"
, "timestamp": "2025-10-05T02:45:14.132329Z"}
{"epoch": 18, "step": 12672, "val_loss": 0.5539602588249158, "val_acc": 0.815466
7721518988, "event": "epoch_summary", "logger": "hw04.training", "level": "info"
, "timestamp": "2025-10-05T02:47:53.772006Z"}
{"epoch": 19, "step": 13376, "val_loss": 0.5059745707843877, "val_acc": 0.822587
0253164557, "event": "epoch_summary", "logger": "hw04.training", "level": "info"
, "timestamp": "2025-10-05T02:50:45.324890Z"}
{"event": "Finished blocking save. Time taken: 0.045058s. Continuing background
save to /work/hw04/checkpoints/best_state_epoch_20."}
{"epoch": 20, "step": 14080, "val_loss": 0.5034029578106313, "val_acc": 0.824960
4430379747, "event": "epoch_summary", "logger": "hw04.training", "level": "info"
, "timestamp": "2025-10-05T02:53:26.954941Z"}
{"event": "Finished async_save (blocking + background). Time taken: 0.561195s. d
irectory=/work/hw04/checkpoints/best_state_epoch_20"}
{"event": "[process=0][thread=async_save] Background save thread done. Time take
n: 0.509544s."}
{"epoch": 21, "step": 14784, "val_loss": 0.4975973575175563, "val_acc": 0.830893
9873417721, "event": "epoch_summary", "logger": "hw04.training", "level": "info"
, "timestamp": "2025-10-05T02:55:42.652465Z"}
{"epoch": 22, "step": 15488, "val_loss": 0.5027272561682931, "val_acc": 0.830300
6329113924, "event": "epoch_summary", "logger": "hw04.training", "level": "info"
, "timestamp": "2025-10-05T02:57:59.114260Z"}
{"epoch": 23, "step": 16192, "val_loss": 0.5082427591462678, "val_acc": 0.827136
0759493671, "event": "epoch_summary", "logger": "hw04.training", "level": "info"
, "timestamp": "2025-10-05T03:00:27.786364Z"}
{"epoch": 24, "step": 16896, "val_loss": 0.4491530262593982, "val_acc": 0.842167
7215189873, "event": "epoch_summary", "logger": "hw04.training", "level": "info"
, "timestamp": "2025-10-05T03:02:49.746914Z"}
{"event": "Finished blocking save. Time taken: 0.070533s. Continuing background
save to /work/hw04/checkpoints/best_state_epoch_25."}
{"epoch": 25, "step": 17600, "val_loss": 0.45212958206104326, "val_acc": 0.84276
10759493671, "event": "epoch_summary", "logger": "hw04.training", "level": "info
", "timestamp": "2025-10-05T03:05:19.481743Z"}
```

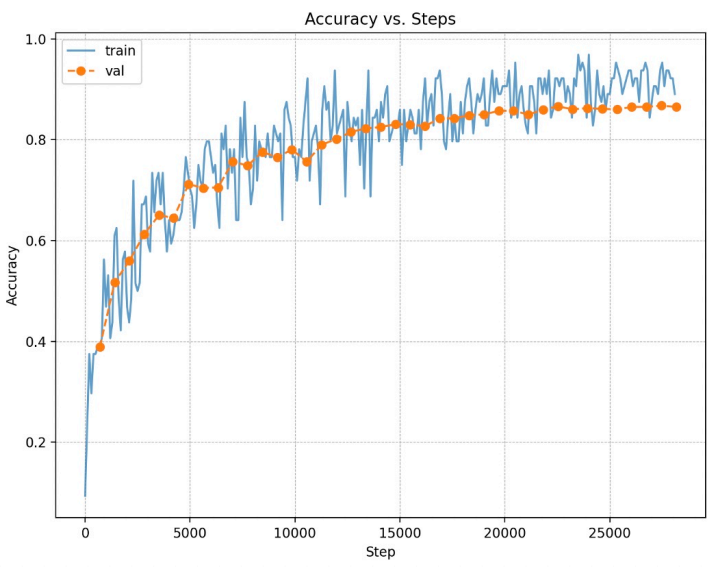
```
{"event": "Finished async_save (blocking + background). Time taken: 0.538781s. directory=/work/hw04/checkpoints/best_state_epoch_25"}
{"event": "[process=0][thread=async_save] Background save thread done. Time taken: 0.454450s."}
{"epoch": 26, "step": 18304, "val_loss": 0.44651460760756384, "val_acc": 0.8482990506329114, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-05T03:08:14.108608Z"}
{"epoch": 27, "step": 19008, "val_loss": 0.43895917282074315, "val_acc": 0.8502768987341772, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-05T03:10:33.586123Z"}
{"epoch": 28, "step": 19712, "val_loss": 0.4258923664500442, "val_acc": 0.8573971518987342, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-05T03:12:58.646980Z"}
{"epoch": 29, "step": 20416, "val_loss": 0.4334387503847291, "val_acc": 0.8575949367088608, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-05T03:15:29.196751Z"}
{"event": "Finished blocking save. Time taken: 0.114508s. Continuing background save to /work/hw04/checkpoints/best_state_epoch_30."}
{"epoch": 30, "step": 21120, "val_loss": 0.43960481146468394, "val_acc": 0.8498813291139324, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-05T03:18:00.676294Z"}
{"event": "Finished async_save (blocking + background). Time taken: 0.500118s. directory=/work/hw04/checkpoints/best_state_epoch_30"}
{"epoch": 31, "step": 21824, "val_loss": 0.41756087937687014, "val_acc": 0.8597705696202531, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-05T03:20:23.191371Z"}
{"epoch": 32, "step": 22528, "val_loss": 0.4153142190432247, "val_acc": 0.8662974683544303, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-05T03:22:51.648931Z"}
{"epoch": 33, "step": 23232, "val_loss": 0.41485248147686826, "val_acc": 0.8603639240506329, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-05T03:25:20.120943Z"}
{"epoch": 34, "step": 23936, "val_loss": 0.4226948241644268, "val_acc": 0.8627373417721519, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-05T03:28:03.197280Z"}
{"epoch": 35, "step": 24640, "val_loss": 0.41677922802635387, "val_acc": 0.8611550632911392, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-05T03:30:31.283716Z"}
{"event": "Finished async_save (blocking + background). Time taken: 0.437339s. directory=/work/hw04/checkpoints/best_state_epoch_35"}
{"epoch": 36, "step": 25344, "val_loss": 0.41137097775936127, "val_acc": 0.8605617088607594, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-05T03:33:27.668778Z"}
{"epoch": 37, "step": 26048, "val_loss": 0.40533167953732646, "val_acc": 0.8649129746835443, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-05T03:35:56.804861Z"}
{"epoch": 38, "step": 26752, "val_loss": 0.4060941059378129, "val_acc": 0.8651107594936709, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-05T03:38:24.808950Z"}
{"epoch": 39, "step": 27456, "val_loss": 0.4030414829148522, "val_acc": 0.8676819620253164, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-05T03:40:52.277778Z"}
{"event": "Finished blocking save. Time taken: 0.118553s. Continuing background save to /work/hw04/checkpoints/best_state_epoch_40."}
{"epoch": 40, "step": 28160, "val_loss": 0.404538533166994, "val_acc": 0.8651107594936709, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-05T03:43:35.801138Z"}
{"event": "Training finished. Generating final plots...", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-05T03:43:35.804946Z"}
{"settings": {"debug": false, "random_seed": 31415, "model": {"input_depth": 3, "layer_depths": [16, 32, 64], "blocks_per_stage": [3, 3, 3], "num_classes": 10}, "data": {"batch_size": 64, "validation_size": 5000, "shuffle_buffer": 10000, "cifar100": false}, "training": {"num_epochs": 40, "learning_rate": 0.001, "weight_decay": 0.0001, "log_interval": 100, "final_test": true, "checkpoint_dir": "PosixPath('hw04/checkpoints_cifar10')"}, "plotting": {"output_dir": "PosixPath('hw04/artifacts')", "figsize": [5, 3], "dpi": 200}}, "event": "Settings loaded", "logger": "hw04", "level": "info", "timestamp": "2025-10-08T20:07:20.965938Z"}
{"event": "Variant folder /root/tensorflow_datasets/cifar10/3.0.2 has no dataset_info.json"}
```



```
{"event": "Generating dataset cifar10 (/root/tensorflow_datasets/cifar10/3.0.2)"}
{"event": "Downloading https://www.cs.toronto.edu/~kriz/cifar-10-binary.tar.gz into /root/tensorflow_datasets/downloads/cifar10/cs.toronto.edu_kriz_cifar-10-binaryODHPtIjLh3oLcXirEISTO7dkzyKjRCuol6lV8Wc6C7s.tar.gz.tmp.f89797db97604e0c9d9c28774f134f32..."}
{"event": "Done writing /root/tensorflow_datasets/cifar10/incomplete.DJLYYH_3.0.2/cifar10-train.tfrecord*. Number of examples: 50000 (shards: [50000])"}
{"event": "Done writing /root/tensorflow_datasets/cifar10/incomplete.DJLYYH_3.0.2/cifar10-test.tfrecord*. Number of examples: 10000 (shards: [10000])"}
{"event": "Creating a tf.data.Dataset reading 1 files located in folders: /root/tensorflow_datasets/cifar10/3.0.2."}
{"event": "Constructing tf.data.Dataset cifar10 for split train, from /root/tensorflow_datasets/cifar10/3.0.2"}
{"event": "Load dataset info from /root/tensorflow_datasets/cifar10/3.0.2"}
{"event": "Creating a tf.data.Dataset reading 1 files located in folders: /root/tensorflow_datasets/cifar10/3.0.2."}
{"event": "Constructing tf.data.Dataset cifar10 for split test, from /root/tensorflow_datasets/cifar10/3.0.2"}
{"event": "Unable to initialize backend 'tpu': INTERNAL: Failed to open libtpu.so: libtpu.so: cannot open shared object file: No such file or directory"}
{"num_params": 272938, "event": "Model created", "logger": "hw04", "level": "info", "timestamp": "2025-10-08T20:08:05.943964Z"}
{"event": "Starting final testing...", "logger": "hw04", "level": "info", "timestamp": "2025-10-08T20:08:06.201148Z"}
{"event": "orbax-checkpoint version: 0.11.25"}
{"event": "Created BasePyTreeCheckpointHandler: use_ocdbt=True, use_zarr3=False, pytree_metadata_options=PyTreeMetadataOptions(support_rich_types=False), array_metadata_store=<orbax.checkpoint._src.metadata.array_metadata_store.Store object at 0x7f2ffefd9b90>, enable_pinned_host_transfer=False, save_concurrent_bytes: 96000000000 (89.4 GiB), restore_concurrent_bytes: 96000000000 (89.4 GiB)"}
{"event": "[thread=MainThread] Failed to get flag value for EXPERIMENTAL_ORBAX_USE_DISTRIBUTED_PROCESS_ID."}
{"event": "[process=0][thread=MainThread] Using barrier_sync_fn: <function get_barrier_sync_fn.<locals>.<lambda> at 0x7f2f7a966160> timeout: 600 secs and primary_host=0 for async checkpoint writes"}
{"event": "Restoring checkpoint from /work/hw04/checkpoints_cifar10/best_state_epoch_40."}
{"event": "[process=0] /jax/checkpoint/read/gbytes_per_sec: 0 Bytes/s (total gbytes: 1.0 MiB) (time elapsed: 296 milliseconds) (per-host)"}
{"event": "Finished restoring checkpoint in 0.30 seconds from /work/hw04/checkpoints_cifar10/best_state_epoch_40."}
{"test_loss": 1.4637850788748188, "test_acc": 0.8576831210191083, "event": "test_results", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-08T20:08:17.975697Z"}
{"test_loss": 1.4637850788748188, "test_acc": 0.8576831210191083, "event": "Final Test Results", "logger": "hw04", "level": "info", "timestamp": "2025-10-08T20:08:18.006717Z"}
```

Loss and Accuracy Graphs for CIFAR-10 Training





```
/*
    I deleted all the training log (train loss) since there were too many and ju
st leave the validation loss log
    It should be easier to see the train/validation loss graph to see the traini
ng summary
    I see some overfitting from the validation loss graphs.
    Stronger augmentation and Dropout could be helpful
    Test set result is at the very bottom
    I tried to use renet-34 like structure but with less layer depths.
    Training took about 7 hours in my cpu.


    Started:    2025-10-08T02:04:23.076490Z
    Ended:      2025-10-08T09:18:38.831492Z

*/
{"settings": {"debug": false, "random_seed": 31415, "model": {"input_depth": 3,
"layer_depths": [32, 64, 128, 256], "blocks_per_stage": [3, 4, 6, 3], "num_class
es": 100}, "data": {"batch_size": 64, "validation_size": 5000, "shuffle_buffer":
 10000, "cifar100": true}, "training": {"num_epochs": 40, "learning_rate": 0.001
, "weight_decay": 0.0001, "log_interval": 100, "final_test": false, "checkpoint_
dir": "PosixPath('hw04/checkpoints')"}, "plotting": {"output_dir": "PosixPath('h
w04/artifacts')", "figsize": [5, 3], "dpi": 200}}, "event": "Settings loaded", "
logger": "hw04", "level": "info", "timestamp": "2025-10-08T02:03:14.897531Z"}
{"event": "Variant folder /root/tensorflow_datasets/cifar100/3.0.2 has no datase
t_info.json"}
{"event": "Generating dataset cifar100 (/root/tensorflow_datasets/cifar100/3.0.2
)"}
{"event": "Downloading https://www.cs.toronto.edu/~kriz/cifar-100-binary.tar.gz
into /root/tensorflow_datasets/downloads/cifar100/cs.toronto.edu_kriz_cifar-100-
binaryzK0jb7CkNxmV4pH2clu5WdAlIotsPlZhrMxx9-DELEk.tar.gz.tmp.05f7acbe3c27433ab9d
9a7be715a262a..."}
{"event": "Done writing /root/tensorflow_datasets/cifar100/incomplete.CJ1809_3.0
.2/cifar100-train.tfrecord*. Number of examples: 50000 (shards: [50000])"}
{"event": "Done writing /root/tensorflow_datasets/cifar100/incomplete.CJ1809_3.0
.2/cifar100-test.tfrecord*. Number of examples: 10000 (shards: [10000])"}
{"event": "Creating a tf.data.Dataset reading 1 files located in folders: /root/
tensorflow_datasets/cifar100/3.0.2."}
{"event": "Constructing tf.data.Dataset cifar100 for split train, from /root/ten
sorflow_datasets/cifar100/3.0.2"}
{"event": "Load dataset info from /root/tensorflow_datasets/cifar100/3.0.2"}
{"event": "Creating a tf.data.Dataset reading 1 files located in folders: /root/
tensorflow_datasets/cifar100/3.0.2."}
{"event": "Constructing tf.data.Dataset cifar100 for split test, from /root/tens
orflow_datasets/cifar100/3.0.2"}
{"event": "Unable to initialize backend 'tpu': INTERNAL: Failed to open libtpu.s
o: libtpu.so: cannot open shared object file: No such file or directory"}
{"num_params": 5352484, "event": "Model created", "logger": "hw04", "level": "in
fo", "timestamp": "2025-10-08T02:04:23.076490Z"}
{"epoch": 1, "step": 704, "val_loss": 3.976373268079154, "val_acc": 0.1097705696
2025317, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "
timestamp": "2025-10-08T02:12:48.870353Z"}
{"epoch": 2, "step": 1408, "val_loss": 3.725985680954366, "val_acc": 0.164556962
02531644, "event": "epoch_summary", "logger": "hw04.training", "level": "info",
"timestamp": "2025-10-08T02:23:02.027043Z"}
{"epoch": 3, "step": 2112, "val_loss": 3.4336151533488986, "val_acc": 0.22329905
06329114, "event": "epoch_summary", "logger": "hw04.training", "level": "info",
"timestamp": "2025-10-08T02:35:52.401048Z"}
{"epoch": 4, "step": 2816, "val_loss": 3.315141044085539, "val_acc": 0.244066455
69620253, "event": "epoch_summary", "logger": "hw04.training", "level": "info",
"timestamp": "2025-10-08T02:49:28.374315Z"}
{"event": "Finished blocking save. Time taken: 0.405937s. Continuing background
save to /work/hw04/checkpoints/best_state_epoch_5."}
{"epoch": 5, "step": 3520, "val_loss": 3.023668651339374, "val_acc": 0.341969936
7088608, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "
timestamp": "2025-10-08T03:03:20.608685Z"}
{"epoch": 6, "step": 4224, "val_loss": 2.891840086707586, "val_acc": 0.368473101
2658228, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "
timestamp": "2025-10-08T03:15:15.281783Z"}
```

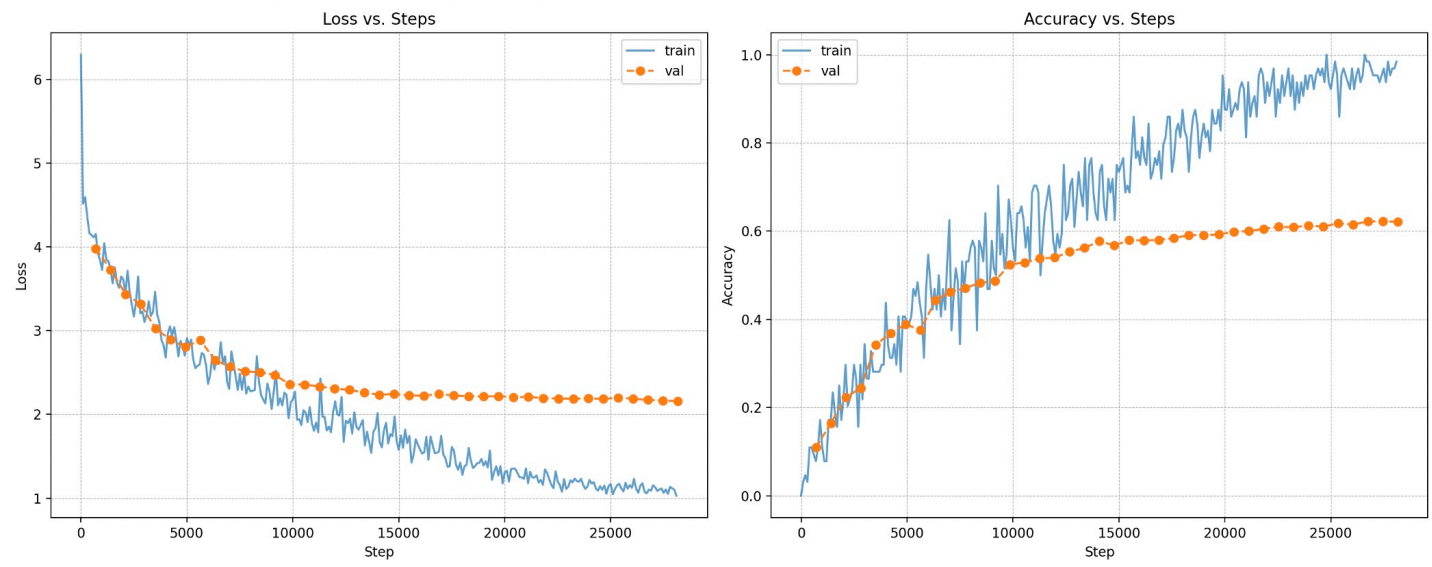


```
{"epoch": 7, "step": 4928, "val_loss": 2.80468063716647, "val_acc": 0.3890427215
1898733, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "
timestamp": "2025-10-08T03:30:54.135918Z"}
{"epoch": 8, "step": 5632, "val_loss": 2.8846074689792682, "val_acc": 0.37539556
962025317, "event": "epoch_summary", "logger": "hw04.training", "level": "info",
 "timestamp": "2025-10-08T03:43:39.004092Z"}
{"epoch": 9, "step": 6336, "val_loss": 2.644265703008145, "val_acc": 0.442840189
8734177, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "
timestamp": "2025-10-08T03:57:08.722001Z"}
{"event": "Finished blocking save. Time taken: 0.287610s. Continuing background
save to /work/hw04/checkpoints/best_state_epoch_10."}
{"epoch": 10, "step": 7040, "val_loss": 2.5681378705592097, "val_acc": 0.4622231
012658228, "event": "epoch_summary", "logger": "hw04.training", "level": "info",
 "timestamp": "2025-10-08T04:11:06.506207Z"}
{"epoch": 11, "step": 7744, "val_loss": 2.513141556631161, "val_acc": 0.47072784
810126583, "event": "epoch_summary", "logger": "hw04.training", "level": "info",
 "timestamp": "2025-10-08T04:24:44.379628Z"}
{"epoch": 12, "step": 8448, "val_loss": 2.5028848678250855, "val_acc": 0.4827927
2151898733, "event": "epoch_summary", "logger": "hw04.training", "level": "info"
, "timestamp": "2025-10-08T04:37:42.487969Z"}
{"epoch": 13, "step": 9152, "val_loss": 2.4689806473406057, "val_acc": 0.4869462
0253164556, "event": "epoch_summary", "logger": "hw04.training", "level": "info"
, "timestamp": "2025-10-08T04:51:23.553805Z"}
{"epoch": 14, "step": 9856, "val_loss": 2.3596031695981567, "val_acc": 0.5245253
164556962, "event": "epoch_summary", "logger": "hw04.training", "level": "info",
 "timestamp": "2025-10-08T05:05:20.270818Z"}
{"event": "Finished blocking save. Time taken: 0.353670s. Continuing background
save to /work/hw04/checkpoints/best_state_epoch_15."}
{"epoch": 15, "step": 10560, "val_loss": 2.3550007765806176, "val_acc": 0.528876
582278481, "event": "epoch_summary", "logger": "hw04.training", "level": "info",
 "timestamp": "2025-10-08T05:18:59.185930Z"}
{"epoch": 16, "step": 11264, "val_loss": 2.332110721853715, "val_acc": 0.5379746
835443038, "event": "epoch_summary", "logger": "hw04.training", "level": "info",
 "timestamp": "2025-10-08T05:31:56.445378Z"}
{"epoch": 17, "step": 11968, "val_loss": 2.3060759547390517, "val_acc": 0.539952
5316455697, "event": "epoch_summary", "logger": "hw04.training", "level": "info"
, "timestamp": "2025-10-08T05:42:01.686646Z"}
{"epoch": 18, "step": 12000, "loss": 2.1541953086853027, "acc": 0.546875, "event
": "train_iter", "logger": "hw04.training", "level": "info", "timestamp": "2025-
10-08T05:42:26.991727Z"}
{"epoch": 18, "step": 12100, "loss": 1.9982845783233643, "acc": 0.59375, "event"
: "train_iter", "logger": "hw04.training", "level": "info", "timestamp": "2025-1
0-08T05:43:40.246351Z"}
{"epoch": 18, "step": 12200, "loss": 1.9845942258834839, "acc": 0.5625, "event":
 "train_iter", "logger": "hw04.training", "level": "info", "timestamp": "2025-10
-08T05:44:54.006460Z"}
{"epoch": 18, "step": 12300, "loss": 2.2072534561157227, "acc": 0.59375, "event"
: "train_iter", "logger": "hw04.training", "level": "info", "timestamp": "2025-1
0-08T05:46:06.017281Z"}
{"epoch": 18, "step": 12400, "loss": 1.6721937656402588, "acc": 0.75, "event": "
train_iter", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-0
8T05:47:17.964478Z"}
{"epoch": 18, "step": 12500, "loss": 1.924797773361206, "acc": 0.625, "event": "
train_iter", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-0
8T05:48:27.335253Z"}
{"epoch": 18, "step": 12600, "loss": 1.895101547241211, "acc": 0.640625, "event"
: "train_iter", "logger": "hw04.training", "level": "info", "timestamp": "2025-1
0-08T05:49:37.582857Z"}
{"epoch": 18, "step": 12672, "val_loss": 2.2927366072618507, "val_acc": 0.553006
329113924, "event": "epoch_summary", "logger": "hw04.training", "level": "info",
 "timestamp": "2025-10-08T05:50:42.273608Z"}
{"epoch": 19, "step": 13376, "val_loss": 2.25943695442586, "val_acc": 0.56190664
55696202, "event": "epoch_summary", "logger": "hw04.training", "level": "info",
"timestamp": "2025-10-08T06:02:30.057835Z"}
{"event": "Finished blocking save. Time taken: 0.302445s. Continuing background
save to /work/hw04/checkpoints/best_state_epoch_20."}
{"epoch": 20, "step": 14080, "val_loss": 2.2340246137184434, "val_acc": 0.577136
0759493671, "event": "epoch_summary", "logger": "hw04.training", "level": "info"
, "timestamp": "2025-10-08T06:13:47.481533Z"}
```

{"epoch": 21, "step": 14784, "val_loss": 2.243983276282685, "val_acc": 0.5674446202531646, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-08T06:23:34.818738Z"}
{"epoch": 22, "step": 15488, "val_loss": 2.2258135306684275, "val_acc": 0.5797072784810127, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-08T06:32:59.383679Z"}
{"epoch": 23, "step": 16192, "val_loss": 2.222693950315065, "val_acc": 0.5787183544303798, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-08T06:42:14.872808Z"}
{"epoch": 24, "step": 16896, "val_loss": 2.243938494332229, "val_acc": 0.5801028481012658, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-08T06:52:52.934472Z"}
{"event": "Finished blocking save. Time taken: 0.174913s. Continuing background save to /work/hw04/checkpoints/best_state_epoch_25."}
{"epoch": 25, "step": 17600, "val_loss": 2.224066859559168, "val_acc": 0.584256329113924, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-08T07:02:52.741474Z"}
{"epoch": 26, "step": 18304, "val_loss": 2.2172716629656057, "val_acc": 0.5909810126582279, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-08T07:13:10.497284Z"}
{"epoch": 27, "step": 19008, "val_loss": 2.2137932611417166, "val_acc": 0.5909810126582279, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-08T07:22:51.116493Z"}
{"epoch": 28, "step": 19712, "val_loss": 2.216762476329562, "val_acc": 0.5923655063291139, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-08T07:32:24.958280Z"}
{"epoch": 29, "step": 20416, "val_loss": 2.2051513798629183, "val_acc": 0.5986946202531646, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-08T07:42:18.802216Z"}
{"epoch": 30, "step": 21120, "val_loss": 2.2079105467735967, "val_acc": 0.599881329113924, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-08T07:52:27.256982Z"}
{"event": "Finished async_save (blocking + background). Time taken: 1.108641s. directory=/work/hw04/checkpoints/best_state_epoch_30"}
{"event": "[process=0][thread=async_save] Background save thread done. Time taken: 0.928715s."}
{"epoch": 31, "step": 21824, "val_loss": 2.1937194051621836, "val_acc": 0.6050237341772152, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-08T08:02:25.895055Z"}
{"epoch": 32, "step": 22528, "val_loss": 2.1885239006597783, "val_acc": 0.6099683544303798, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-08T08:12:54.682985Z"}
{"epoch": 33, "step": 23232, "val_loss": 2.186336153670202, "val_acc": 0.6087816455696202, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-08T08:23:14.888357Z"}
{"epoch": 34, "step": 23936, "val_loss": 2.1903039745137662, "val_acc": 0.6125395569620253, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-08T08:31:10.139681Z"}
{"event": "Finished blocking save. Time taken: 0.171701s. Continuing background save to /work/hw04/checkpoints/best_state_epoch_35."}
{"epoch": 35, "step": 24640, "val_loss": 2.183525858046133, "val_acc": 0.6101661392405063, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-08T08:39:03.468770Z"}
{"epoch": 36, "step": 25344, "val_loss": 2.197526878948453, "val_acc": 0.6176819620253164, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-08T08:46:58.679913Z"}
{"epoch": 37, "step": 26048, "val_loss": 2.1875499139858197, "val_acc": 0.6149129746835443, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-08T08:54:52.045278Z"}
{"epoch": 38, "step": 26752, "val_loss": 2.171840216540083, "val_acc": 0.6220332278481012, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-08T09:02:46.707634Z"}
{"epoch": 39, "step": 27456, "val_loss": 2.1635287939747676, "val_acc": 0.6218354430379747, "event": "epoch_summary", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-08T09:10:44.956816Z"}
{"event": "Finished blocking save. Time taken: 0.342879s. Continuing background save to /work/hw04/checkpoints/best_state_epoch_40."}
{"epoch": 40, "step": 28160, "val_loss": 2.158977393862567, "val_acc": 0.6214398734177216, "event": "epoch_summary", "logger": "hw04.training", "level": "info",

"timestamp": "2025-10-08T09:18:38.831492Z"}
{"num_params": 5352484, "event": "Model created", "logger": "hw04", "level": "info", "timestamp": "2025-10-08T19:58:42.569048Z"}
{"event": "Starting final testing...", "logger": "hw04", "level": "info", "timestamp": "2025-10-08T19:58:42.898115Z"}
{"event": "orbax-checkpoint version: 0.11.25"}
{"event": "Created BasePyTreeCheckpointHandler: use_ocdbt=True, use_zarr3=False, pytree_metadata_options=PyTreeMetadataOptions(support_rich_types=False), array_metadata_store=<orbax.checkpoint._src.metadata.array_metadata_store.Store object at 0x7f589abd6790>, enable_pinned_host_transfer=False, save_concurrent_bytes: 96000000000 (89.4 GiB), restore_concurrent_bytes: 96000000000 (89.4 GiB)"}
{"event": "[thread=MainThread] Failed to get flag value for EXPERIMENTAL_ORBAX_USE_DISTRIBUTED_PROCESS_ID."}
{"event": "[process=0][thread=MainThread] Using barrier_sync_fn: <function get_barrier_sync_fn.<locals>.<lambda> at 0x7f5816f520c0> timeout: 600 secs and primary_host=0 for async checkpoint writes"}
{"event": "Restoring checkpoint from /work/hw04/checkpoints/best_state_epoch_40."}
{"event": "[process=0] /jax/checkpoint/read/gbytes_per_sec: 0 Bytes/s (total gbytes: 20.4 MiB) (time elapsed: 549 milliseconds) (per-host)"}
{"event": "Finished restoring checkpoint in 0.56 seconds from /work/hw04/checkpoints/best_state_epoch_40."}
{"test_loss": 2.175596616830036, "test_acc": 0.6136544585987261, "event": "test_results", "logger": "hw04.training", "level": "info", "timestamp": "2025-10-08T19:59:12.667384Z"}
{"test_loss": 2.175596616830036, "test_acc": 0.6136544585987261, "event": "Final Test Results", "logger": "hw04", "level": "info", "timestamp": "2025-10-08T19:59:12.705750Z"}

Loss and Accuracy Graphs for CIFAR-100 Training



```
# pyproject.toml.jinja

[project]
name = "hw04"
version = "0.1.0"
description = "CIFARIO"
readme = "README.md"
authors = [
    { name = "Wongee (Freddy) Hong", email = "wongee.hong@cooper.edu" }
]
requires-python = ">=3.11"
dependencies = [
    "structlog",
    "numpy",
    "tensorflow",
    "tensorflow_datasets",
    "pydantic-settings",
    "matplotlib",
    "tqdm",
    "jax",
    "jaxlib",
    "flax",
    "optax",
    "scikit-learn",
]

[project.scripts]
hw04 = "hw04:main"

[build-system]
requires = ["uv_build>=0.8.3,<0.9.0"]
build-backend = "uv_build"
```

```python
import logging
import os
import sys
from pathlib import Path

import jax
import numpy as np
import structlog


class FormattedFloat(float):
    def __repr__(self) -> str:
        return f"{self:.4g}"


def custom_serializer_processor(logger, method_name, event_dict):
    for key, value in event_dict.items():
        if hasattr(value, "numpy"):
            value = value.numpy()
        if isinstance(value, jax.Array):
            value = np.array(value)
        if isinstance(value, (np.generic, np.ndarray)):
            value = value.item() if value.size == 1 else value.tolist()
        if isinstance(value, float):
            value = FormattedFloat(value)
        if isinstance(value, Path):
            value = str(value)
        event_dict[key] = value
    return event_dict

def configure_logging(log_dir: Path = Path("hw04/artifacts"), log_name: str = "train.jso
n"):
    log_dir.mkdir(parents=True, exist_ok=True)
    log_file = log_dir / log_name
    log_level = os.environ.get("LOG_LEVEL", "INFO").upper()

    shared_processors = [
        structlog.stdlib.add_logger_name,
        structlog.stdlib.add_log_level,
        structlog.stdlib.PositionalArgumentsFormatter(),
        structlog.processors.TimeStamper(fmt="iso"),
        structlog.processors.StackInfoRenderer(),
        structlog.processors.format_exc_info,
        structlog.processors.UnicodeDecoder(),
        custom_serializer_processor,
    ]


    structlog.configure(
        processors=shared_processors + [

            structlog.stdlib.ProcessorFormatter.wrap_for_formatter,
        ],
        logger_factory=structlog.stdlib.LoggerFactory(),
        wrapper_class=structlog.stdlib.BoundLogger,
        cache_logger_on_first_use=True,
    )

    console_renderer = structlog.dev.ConsoleRenderer(
        colors=sys.stdout.isatty(),
        exception_formatter=structlog.dev.RichTracebackFormatter(),
    )
    console_handler = logging.StreamHandler(sys.stdout)
    console_handler.setFormatter(
        structlog.stdlib.ProcessorFormatter(

            processors=[
                structlog.stdlib.ProcessorFormatter.remove_processors_meta,
                console_renderer
```



```python
            ],
        )
    )

    file_renderer = structlog.processors.JSONRenderer()
    file_handler = logging.FileHandler(log_file, mode="w")
    file_handler.setFormatter(
        structlog.stdlib.ProcessorFormatter(
            processors=[
                structlog.stdlib.ProcessorFormatter.remove_processors_meta,
                file_renderer
            ],
        )
    )

    root_logger = logging.getLogger()
    root_logger.addHandler(console_handler)
    root_logger.addHandler(file_handler)
    root_logger.setLevel(log_level)

    logging.getLogger("matplotlib").setLevel(logging.WARNING)
    logging.getLogger("PIL").setLevel(logging.WARNING)

    return log_file
```

```python
from pathlib import Path
from typing import Tuple
from pydantic import BaseModel
from pydantic_settings import BaseSettings


class ModelSettings(BaseModel):
    input_depth: int = 3
    layer_depths: list[int] = [32, 64, 128, 256]
    blocks_per_stage: list[int] = [3, 4, 6, 3]
    num_classes: int = 100


class DataSettings(BaseModel):
    batch_size: int = 64
    validation_size: int = 5000
    shuffle_buffer: int = 10000
    cifar100: bool = True


class TrainingSettings(BaseModel):
    """Settings for model training."""
    num_epochs: int = 40
    learning_rate: float = 0.001
    weight_decay: float = 1e-4                    # L2 penalty
    log_interval: int = 100
    final_test: bool = True
    checkpoint_dir: Path = Path("hw04/checkpoints")


class PlottingSettings(BaseModel):
    """Settings for logging and saving artifacts."""
    output_dir: Path = Path("hw04/artifacts")
    figsize: Tuple[int, int] = (5, 3)
    dpi: int = 200


class AppSettings(BaseSettings):
    """Main application settings."""
    debug: bool = False
    random_seed: int = 31415

    model: ModelSettings = ModelSettings()
    data: DataSettings = DataSettings()
    training: TrainingSettings = TrainingSettings()
    plotting: PlottingSettings = PlottingSettings()


def load_settings() -> AppSettings:
    """Load application settings."""
    return AppSettings()
```

```python
import optax
import structlog
from flax import nnx
import jax
import pathlib as Path
import orbax.checkpoint as ocp

from .config import load_settings
from .logging import configure_logging
from .model import Classifier
from .training import train, test_evaluation, load_checkpoint
from .data import load_CIFAR10, load_CIFAR100

def main() -> None:
    settings = load_settings()
    log_file = configure_logging()
    log = structlog.get_logger()
    log.info("Settings loaded", settings=settings.model_dump())
    print(f"Logs are being saved to {log_file}")

    if settings.data.cifar100:
        ds_train, ds_val, ds_test = load_CIFAR100(
        batch_size=settings.data.batch_size,
        validation_size=settings.data.validation_size,
    )
    else:
        ds_train, ds_val, ds_test = load_CIFAR10(
            batch_size=settings.data.batch_size,
            validation_size=settings.data.validation_size,
        )
    rngs = nnx.Rngs(params=settings.random_seed)
    model = Classifier(
        rngs=rngs,
        input_depth=settings.model.input_depth,
        layer_depths=settings.model.layer_depths,
        blocks_per_stage=settings.model.blocks_per_stage,
        layer_kernel_sizes=[(3,3)] * len(settings.model.layer_depths),
        num_classes=settings.model.num_classes,
    )

    params = nnx.state(model, nnx.Param)
    num_params = sum(p.size for p in jax.tree_util.tree_leaves(params))
    log.info("Model created", num_params=num_params)

    # LR schedule and optimizer
    train_size      = 50000
    steps_per_epoch = train_size // settings.data.batch_size
    decay_steps     = settings.training.num_epochs * steps_per_epoch
    lr_schedular = optax.schedules.cosine_decay_schedule(
        init_value=settings.training.learning_rate,
        decay_steps=decay_steps,
    )
    optimizer = nnx.Optimizer(
        model,
        optax.adamw(
            learning_rate=lr_schedular,
            weight_decay=settings.training.weight_decay,
        ),
        wrt=nnx.Param,
    )

    if settings.training.final_test:
        log.info("Starting final testing...")

        model_kwargs = dict(
            input_depth=settings.model.input_depth,
            layer_depths=settings.model.layer_depths,
            blocks_per_stage=settings.model.blocks_per_stage,
            layer_kernel_sizes=[(3, 3)] * len(settings.model.layer_depths),
```



```python
            num_classes=settings.model.num_classes,
        )

        model = load_checkpoint(
            Classifier,
            model_kwargs
        )

        test_loss, test_acc = test_evaluation(model, ds_test, rngs)
        log.info("Final Test Results", test_loss=test_loss, test_acc=test_acc)
    else:
        _ = train(model, optimizer, ds_train, ds_val, rngs)
```

```python
# Deleted dropout to implement checkpoint

import jax
import jax.numpy as jnp
from flax import nnx

class Conv2d(nnx.Module):
    def __init__(
        self,
        in_channels: int,
        out_channels: int,
        kernel_size,
        *,
        strides=(1, 1),
        padding="SAME",
        use_bias=True,
        rngs: nnx.Rngs,
    ):
        self.conv = nnx.Conv(
            in_channels, out_channels, kernel_size,
            strides=strides, padding=padding, use_bias=use_bias, rngs=rngs
        )


    def __call__(self, x, *, rngs: nnx.Rngs):
        x = self.conv(x)
        return x


class ResidualBlock(nnx.Module):
    """Basic residual block."""
    def __init__(
        self,
        in_channels: int,
        out_channels: int,
        *,
        stride=(1, 1),
        rngs: nnx.Rngs,
    ):
        self.norm1 = nnx.GroupNorm(num_features=in_channels, num_groups=8, rngs=
rngs)
        self.conv1 = Conv2d(
            in_channels, out_channels, (3, 3),
            strides=stride, rngs=rngs,
        )
        self.norm2 = nnx.GroupNorm(num_features=out_channels, num_groups=8, rngs
=rngs)
        self.conv2 = Conv2d(
            out_channels, out_channels, (3, 3),
            strides=(1, 1), rngs=rngs,
        )
        self.shortcut = (
            Conv2d(
                in_channels, out_channels, (1, 1),
                strides=stride, rngs=rngs,
            )
            if in_channels != out_channels or stride != (1, 1)
            else None
        )

    def __call__(self, x, *, rngs: nnx.Rngs):
        out = self.norm1(x)
        out = nnx.relu(out)
        out = self.conv1(out, rngs=rngs)

        out = self.norm2(out)
        out = nnx.relu(out)
        out = self.conv2(out, rngs=rngs)
```



```python
        identity = x if self.shortcut is None else self.shortcut(x, rngs=rngs)
        return out + identity

class ResidualStage(nnx.Module):
    def __init__(
        self,
        in_channels: int,
        out_channels: int,
        num_blocks: int,
        stride=(1, 1),
        *,
        rngs: nnx.Rngs,
    ):
        blocks = []
        blocks.append(
            ResidualBlock(
                in_channels,
                out_channels,
                stride=stride,
                rngs=rngs,
            )
        )
        # Rest keep stride = (1,1)
        for _ in range(1, num_blocks):
            blocks.append(
                ResidualBlock(
                    out_channels,
                    out_channels,
                    stride=(1, 1),
                    rngs=rngs,
                )
            )
        self.blocks = nnx.List(blocks)

    def __call__(self, x, *, rngs: nnx.Rngs):
        for block in self.blocks:
            x = block(x, rngs=rngs)
        return x

class Classifier(nnx.Module):
    """Residual network classifier for MNIST."""

    def __init__(
        self,
        input_depth: int,
        layer_depths: list[int],
        blocks_per_stage,
        layer_kernel_sizes: list[tuple[int, int]],
        num_classes: int,
        *,
        rngs: nnx.Rngs,
    ):
        assert len(layer_depths) == len(layer_kernel_sizes), \
            "layer_depths and layer_kernel_sizes must match in length"
        self.init_conv = Conv2d(input_depth, layer_depths[0],
                                kernel_size=layer_kernel_sizes[0],
                                strides=(1, 1), rngs=rngs)

        # Build stages
        stages = []
        self.blocks_per_stage = blocks_per_stage
        in_ch = layer_depths[0]
        for i, out_ch in enumerate(layer_depths):
            stride = (2, 2) if i > 0 else (1, 1)  # downsample at stage boundari
es
            num_blocks = (self.blocks_per_stage[i]
                      if self.blocks_per_stage is not None
                      else 1)
            stage = ResidualStage(
```

```
                in_channels=in_ch,
                out_channels=out_ch,
                num_blocks=num_blocks,
                stride=stride,
                rngs=rngs,
            )
            stages.append(stage)
            in_ch = out_ch

        self.stages = nnx.List(stages)

        # Final classifier head
        self.fc = nnx.Linear(in_ch, num_classes, rngs=rngs)

    def __call__(self, x, *, rngs: nnx.Rngs):
        x = self.init_conv(x, rngs=rngs)
        for stage in self.stages:
            x = stage(x, rngs=rngs)

        x = jnp.mean(x, axis=(1, 2))

        # Final logits
        return self.fc(x)
```

```python
import jax
import jax.numpy as jnp
import optax
from flax import nnx
import matplotlib.pyplot as plt
import structlog
import orbax.checkpoint as ocp
from pathlib import Path
import os

from .config import load_settings

log = structlog.get_logger()

def cross_entropy_loss(logits, labels, num_classes, smoothing = 0.1):
    one_hot = jax.nn.one_hot(labels, num_classes)
    smoothed = one_hot * (1 - smoothing) + smoothing / num_classes # Implemeted label smoothing for CIFAR-100 training.
    loss = optax.softmax_cross_entropy(logits, smoothed).mean()
    return loss


def compute_accuracy(logits, labels):
    preds = jnp.argmax(logits, axis=-1)
    return jnp.mean(preds == labels)


def _loss_fn(model, batch, rngs):
    settings = load_settings()
    images, labels = batch
    labels = labels.ravel()
    logits = model(images, rngs=rngs)
    loss = cross_entropy_loss(logits, labels, settings.model.num_classes)
    return loss, logits


@nnx.jit
def train_step(model, optimizer, batch, rngs):
    grad_fn = nnx.value_and_grad(_loss_fn, has_aux=True)
    (loss, logits), grads = grad_fn(model, batch, rngs)
    optimizer.update(model, grads)
    return loss, logits


@nnx.jit
def eval_step(model, batch, rngs):
    loss, logits = _loss_fn(model, batch, rngs)
    return loss, logits

def evaluate(model, dataset, rngs):
    total_loss, total_acc, n_batches = 0.0, 0.0, 0
    for batch in dataset.as_numpy_iterator():
        loss, logits = eval_step(model, batch, rngs)
        _, labels = batch
        labels = labels.ravel()
        acc = compute_accuracy(logits, labels)
        loss = float(loss)
        acc = float(acc)
        total_loss += loss
        total_acc += float(acc)
        n_batches += 1
    return total_loss / max(n_batches, 1), total_acc / max(n_batches, 1)

def train(model, optimizer, train_ds, val_ds, rngs):
    settings = load_settings()
    output_dir = settings.plotting.output_dir
    output_dir.mkdir(parents=True, exist_ok=True)
    checkpoint_dir = settings.training.checkpoint_dir.resolve()
    checkpoint_dir.mkdir(parents=True, exist_ok=True)
```



```python
    checkpoint_dir = ocp.test_utils.erase_and_create_empty(checkpoint_dir)

    log_interval = settings.training.log_interval
    num_epochs = settings.training.num_epochs

    metrics_history = {
    "train_steps": [],
    "train_loss": [],
    "train_accuracy": [],
    "val_steps": [],
    "val_loss": [],
    "val_accuracy": [],
}

    global_step = 0
    best_val_acc = 0.0
    best_ckpt_path = None

    for epoch in range(1, num_epochs + 1):

        for batch in train_ds.as_numpy_iterator():
            loss, logits = train_step(model, optimizer, batch, rngs)

            if global_step % log_interval == 0:
                images, labels = batch
                labels = labels.ravel()
                acc = compute_accuracy(logits, labels)

                loss_val = float(loss)
                acc_val = float(acc)

                metrics_history["train_steps"].append(global_step)
                metrics_history["train_loss"].append(loss_val)
                metrics_history["train_accuracy"].append(acc_val)

                log.info(
                    "train_iter",
                    epoch=epoch,
                    step=global_step,
                    loss=loss_val,
                    acc=acc_val,
                )
            global_step += 1

        # Create checkpoint every 5 epoch. Compare with previous best validation accuracy and save best checkpoint.
        val_loss, val_acc = evaluate(model, val_ds, rngs)
        if epoch % 5 == 0 or epoch == num_epochs:
            checkpointer = ocp.StandardCheckpointer()
            if val_acc > best_val_acc:
                best_val_acc = val_acc
                best_ckpt_path = checkpoint_dir / f"best_state_epoch_{epoch}"
                _, state = nnx.split(model)
                checkpointer.save(best_ckpt_path, state)
            else:
                _, state = nnx.split(model)
                ckpt_path = checkpoint_dir / f"state_epoch_{epoch}"
                checkpointer.save(ckpt_path, state)

        metrics_history["val_steps"].append(global_step)
        metrics_history["val_loss"].append(val_loss)
        metrics_history["val_accuracy"].append(val_acc)

        log.info(
            "epoch_summary",
            epoch=epoch,
            step=global_step,
            val_loss=val_loss,
            val_acc=val_acc,
```

```
        )

    log.info("Training finished. Generating final plots...")
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

    ax1.set_title("Loss vs. Steps")
    ax2.set_title("Accuracy vs. Steps")

    ax1.plot(metrics_history["train_steps"], metrics_history["train_loss"], label="train"
, alpha=0.7)
    ax2.plot(metrics_history["train_steps"], metrics_history["train_accuracy"], label="t
rain", alpha=0.7)

    ax1.plot(metrics_history["val_steps"], metrics_history["val_loss"], label="val", m
arker='o', linestyle='--')
    ax2.plot(metrics_history["val_steps"], metrics_history["val_accuracy"], label="val
", marker='o', linestyle='--')

    ax1.set_xlabel("Step")
    ax1.set_ylabel("Loss")
    ax1.legend()
    ax1.grid(True, which='both', linestyle='--', linewidth=0.5)

    ax2.set_xlabel("Step")
    ax2.set_ylabel("Accuracy")
    ax2.legend()
    ax2.grid(True, which='both', linestyle='--', linewidth=0.5)

    fig.tight_layout()
    fig.savefig(output_dir / "final_training_metrics.png", dpi=settings.plotting.dpi)
    plt.close(fig)

    return metrics_history

def test_evaluation(model, test_ds, rngs):
    settings = load_settings()
    output_dir = settings.plotting.output_dir
    output_dir.mkdir(parents=True, exist_ok=True)

    test_loss, test_acc = evaluate(model, test_ds, rngs)
    log.info("test_results", test_loss=test_loss, test_acc=test_acc)

    return test_loss, test_acc

def load_checkpoint(model_cls, model_kwargs: dict):
    settings = load_settings()
    checkpoint_dir = settings.training.checkpoint_dir.resolve()
    checkpointer = ocp.StandardCheckpointer()

    ckpts = sorted(checkpoint_dir.glob("best_state_epoch_*"), key=os.path.getmtime)
    if not ckpts:
        raise FileNotFoundError(f"No checkpoints found in {checkpoint_dir}")

    latest_ckpt = ckpts[-1]
    print(f"Loading checkpoint from: {latest_ckpt}")

    abstract_model = nnx.eval_shape(lambda: model_cls(**model_kwargs, rngs=nnx.R
ngs(0)))
    graphdef, abstract_state = nnx.split(abstract_model)

    # restore parameters
    state_restored = checkpointer.restore(latest_ckpt, abstract_state)
    model = nnx.merge(graphdef, state_restored)
    print("Checkpoint successfully restored!")
    return model
```

```
import tensorflow_datasets as tfds
import tensorflow as tf

def load_CIFAR10(batch_size=128, validation_size=5000, seed=0):
    ds_train = tfds.load("cifar10", split="train", shuffle_files=True, as_supervised
=True)
    ds_test  = tfds.load("cifar10", split="test", as_supervised=True)

    def _preprocess(image, label):
        image = tf.cast(image, tf.float32) / 255.0
        label = tf.cast(label, tf.int32)
        return image, label

    ds_train = ds_train.map(_preprocess, num_parallel_calls=tf.data.AUTOTUNE)
    ds_test  = ds_test.map(_preprocess, num_parallel_calls=tf.data.AUTOTUNE)

    ds_val = ds_train.take(validation_size)
    ds_train = ds_train.skip(validation_size)

    def augmentation(image, label):
        image = tf.image.resize_with_crop_or_pad(image, 40, 40)
        image = tf.image.random_crop(image, size=[32, 32, 3])
        image = tf.image.random_flip_left_right(image)
        image = tf.image.random_brightness(image, max_delta=0.2)
        image = tf.image.random_contrast(image, 0.8, 1.2)
        image = tf.image.random_saturation(image, 0.8, 1.2)
        return image, label

    ds_train = ds_train.map(augmentation, num_parallel_calls=tf.data.AUTOTUNE)

    ds_train = ds_train.shuffle(10 * batch_size, seed=seed).batch(batch_size).pr
efetch(tf.data.AUTOTUNE)
    ds_val   = ds_val.batch(batch_size).prefetch(tf.data.AUTOTUNE)
    ds_test  = ds_test.batch(batch_size).prefetch(tf.data.AUTOTUNE)

    return ds_train, ds_val, ds_test

def load_CIFAR100(batch_size=128, validation_size=5000, seed=0):
    ds_train = tfds.load("cifar100", split="train", shuffle_files=True, as_supervise
d=True)
    ds_test  = tfds.load("cifar100", split="test", as_supervised=True)

    def _preprocess(image, label):
        image = tf.cast(image, tf.float32) / 255.0
        label = tf.cast(label, tf.int32)
        return image, label

    ds_train = ds_train.map(_preprocess, num_parallel_calls=tf.data.AUTOTUNE)
    ds_test  = ds_test.map(_preprocess, num_parallel_calls=tf.data.AUTOTUNE)

    ds_val = ds_train.take(validation_size)
    ds_train = ds_train.skip(validation_size)

    def augmentation(image, label):
        image = tf.image.resize_with_crop_or_pad(image, 40, 40)
        image = tf.image.random_crop(image, size=[32, 32, 3])
        image = tf.image.random_flip_left_right(image)
        image = tf.image.random_brightness(image, max_delta=0.2)
        image = tf.image.random_contrast(image, 0.8, 1.2)
        image = tf.image.random_saturation(image, 0.8, 1.2)
        return image, label

    ds_train = ds_train.map(augmentation, num_parallel_calls=tf.data.AUTOTUNE)

    ds_train = ds_train.shuffle(10 * batch_size, seed=seed).batch(batch_size).pr
efetch(tf.data.AUTOTUNE)
    ds_val   = ds_val.batch(batch_size).prefetch(tf.data.AUTOTUNE)
    ds_test  = ds_test.batch(batch_size).prefetch(tf.data.AUTOTUNE)
```



```
    return ds_train, ds_val, ds_test
```