DeepTrain

Release 0.6.0

Sep 15, 2020

Contents

1 Installation **3**

2 Examples **5**

3 Table of Contents **7**

3.1 Why DeepTrain? . . . 7
3.2 Into DeepTrain . . . 8
3.3 Deeper into DeepTrain . . . 17
3.4 Recommended Usage . . . 34
3.5 Examples . . . 35
3.6 Callbacks . . . 76
3.7 Introspection . . . 111
3.8 How to . . . ? . . . 124
3.9 How does . . . work? . . . 125
3.10 API Reference . . . 127

Full knowledge and control of the train state.

CHAPTER 1

Installation

`pip install deeptrain` (without data; see how to run examples), or clone repository

CHAPTER 2

Examples

CHAPTER 3

Table of Contents

3.1 Why DeepTrain?

DeepTrain is founded on **control** and **introspection**: full knowledge and manipulation of the train state.

3.1.1 What does it do?

Abstract away boilerplate train loop and data loading code, *without* making it into a black box. Code is written intuitively and fully documented. Everything about the train state can be seen via *dedicated attributes*; which batch is being fit and when, how long until an epoch ends, intermediate metrics, etc.

DeepTrain is *not* a "wrapper" around TF; while currently only supporting TF, fitting and data logic is framework-agnostic.

3.1.2 When is it suitable (and not)?

Training *few* models *thoroughly*: closely tracking model and train attributes to debug performance and inform next steps.

DeepTrain is *not* for models that take under an hour to train, or for training hundreds of models at once.

3.1.3 Features

Train Loop

- **Control**: iteration-, batch-, epoch-level customs
- **Resumability**: interrupt-protection, can pause mid-training – ex
- **Tracking & reproducibility**: save & load model, train state, random seeds, and hyperparameter info
- **Callbacks** at any stage of training or validation – *ex*

Data Pipeline

- **AutoData**: need only path to directory, the rest is inferred (but can customize)
- **Faster SSD loading**: load larger batches to maximize read speed utility
- **Flexible batch size**: can differ from that of loaded files, will split/combine – *ex*
- **Stateful timeseries**: splits up a batch into windows, and `reset_states()` (RNNs) at end – *ex*
- **Iter-level preprocessor**: pass batch & labels through `Preprocessor()` before feeding to model – *ex*
- **Loader function**: define custom data loader for any file extension, handled by `DataLoader()`

Introspection

- **Data**: batches and labels are enumerated by "set nums"; know what's being fit and when
- **Model**: auto descriptive naming; gradients, weights, activations visuals – *ex1*, *ex2*
- **Train state**: single-image log of key attributes & hyperparameters for easy reference – ex

Utilities

- **Preprocessing**: batch-making and format conversion methods – docs
- **Calibration**: classifier prediction threshold; best batch subset selection (for e.g. ensembling) – docs
- **Algorithms**: convenience methods for object inspection & manipulation – docs
- **Callbacks**: reusable methods with other libraries supporting callbacks – docs

List not exhaustive; for application-specific features, see examples.

3.2 Into DeepTrain

DeepTrain requires only (1) a compiled model and (2) data directory to run. This example covers these and a bit more to keep truer to standard use.

[1]:
```
import os
from tensorflow.keras.layers import Input, Dense, Conv2D, MaxPooling2D
from tensorflow.keras.layers import Flatten, Activation
from tensorflow.keras.models import Model

from deeptrain import TrainGenerator, DataGenerator
```

3.2.1 Model maker

Begin by defining a model maker function. Input should specify hyperparameters, optimizer, learning rate, etc; this is the "blueprint" which is later saved.

[2]:
```
def make_model(batch_shape, optimizer, loss, metrics, num_classes,
               filters, kernel_size):
    ipt = Input(batch_shape=batch_shape)
```

(continues on next page)

(continued from previous page)

```
    x = Conv2D(filters, kernel_size, activation='relu', padding='same')(ipt)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = Flatten()(x)
    x = Dense(num_classes)(x)

    out = Activation('softmax')(x)

    model = Model(ipt, out)
    model.compile(optimizer, loss, metrics=metrics)
    return model
```

3.2.2 Model configs

Define configs dictionary to feed as `**kwargs` to `make_model`; we'll also pass it to TrainGenerator, which will save it and show in a "report" for easy reference

[3]:
```
batch_size = 128
width, height, channels = 28, 28, 1  # MNIST dims (28 x 28 pixels, greyscale)

MODEL_CFG = dict(
    batch_shape=(batch_size, width, height, channels),
    loss='categorical_crossentropy',
    metrics=['accuracy'],
    optimizer='adam',
    num_classes=10,
    filters=16,
    kernel_size=(3, 3),
)
```

3.2.3 DataGenerator (train) configs

- `data_path`: directory where image data is located
- `labels_path`: where labels file is located
- `batch_size`: number of samples to feed at once to model
- `shuffle`: whether to shuffle data at end of each epoch
- `superbatch_set_nums`: which files to load into a `superbatch`, which holds batches persisently in memory (as opposed to `batch`, which is overwritten after use). Since MNIST is small, we can load it all into RAM.

[4]:
```
datadir = os.path.join("dir", "data", "image")
DATAGEN_CFG = dict(
    data_path=os.path.join(datadir, 'train'),
    labels_path=os.path.join(datadir, 'train', 'labels.h5'),
    batch_size=batch_size,
    shuffle=True,
    superbatch_set_nums='all',
)
```

3.2.4 DataGenerator (validation) configs

[5]:
```
VAL_DATAGEN_CFG = dict(
    data_path=os.path.join(datadir, 'val'),
    labels_path=os.path.join(datadir, 'val', 'labels.h5'),
    batch_size=batch_size,
    shuffle=False,
    superbatch_set_nums='all',
)
```

3.2.5 TrainGenerator configs

- `epochs`: number of epochs to train for
- `logs_dir`: where to save TrainGenerator state, model, report, and history
- `best_models_dir`: where to save model when it achieves new best validation performance
- `model_configs`: model configurations dict to save & write to report

[6]:
```
TRAINGEN_CFG = dict(
    epochs=3,
    logs_dir=os.path.join('dir', 'logs'),
    best_models_dir=os.path.join('dir', 'models'),
    model_configs=MODEL_CFG,
)
```

3.2.6 Create training objects

[7]:
```
model       = make_model(**MODEL_CFG)
datagen     = DataGenerator(**DATAGEN_CFG)
val_datagen = DataGenerator(**VAL_DATAGEN_CFG)
traingen    = TrainGenerator(model, datagen, val_datagen, **TRAINGEN_CFG)
```

```
Discovered 48 files with matching format
Discovered dataset with matching format
48 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

Discovered 36 files with matching format
Discovered dataset with matching format
36 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

NOTE: no existing models detected in dir\logs; starting model_num from '0'
Preloading superbatch ... Discovered 48 files with matching format
... finished, w/ 6144 total samples
Train initial data prepared
Preloading superbatch ... Discovered 36 files with matching format
... finished, w/ 4608 total samples
Val initial data prepared
NOTE: no existing models detected in dir\logs; starting model_num from '0'
Logging ON; directory (new): dir\logs\M0__model-adam__min999.000
```

3.2.7 Train

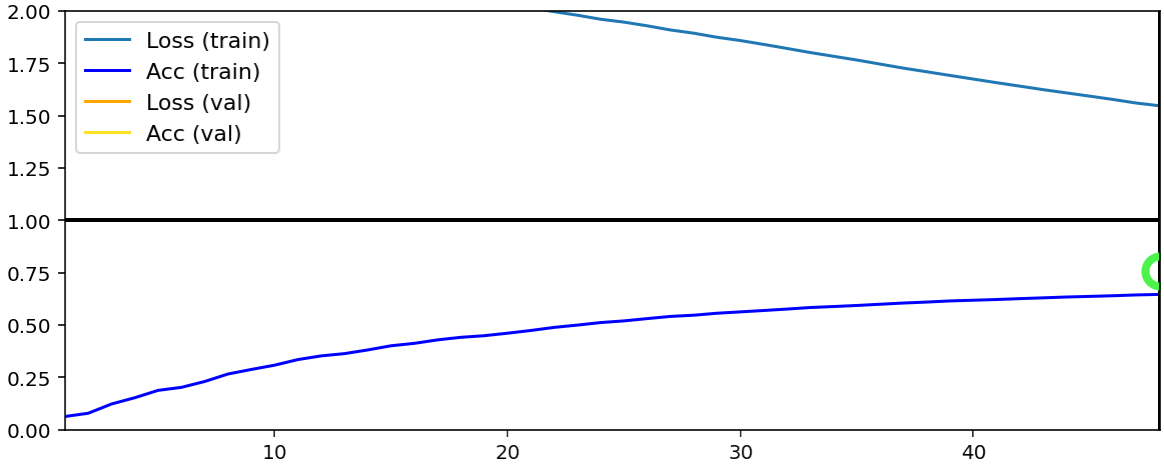
[8]:
```
traingen.train()
```

```
Fitting set 1...   (Loss, Acc) = (2.297301, 0.062500)
Fitting set 2...   (Loss, Acc) = (2.292228, 0.078125)
Fitting set 3...   (Loss, Acc) = (2.280833, 0.122396)
Fitting set 4...   (Loss, Acc) = (2.268434, 0.152344)
Fitting set 5...   (Loss, Acc) = (2.251584, 0.187500)
Fitting set 6...   (Loss, Acc) = (2.239864, 0.201823)
Fitting set 7...   (Loss, Acc) = (2.228770, 0.229911)
Fitting set 8...   (Loss, Acc) = (2.214706, 0.265625)
Fitting set 9...   (Loss, Acc) = (2.201900, 0.287326)
Fitting set 10...  (Loss, Acc) = (2.189120, 0.307812)
Fitting set 11...  (Loss, Acc) = (2.170205, 0.334517)
Fitting set 12...  (Loss, Acc) = (2.154839, 0.352214)
Fitting set 13...  (Loss, Acc) = (2.142305, 0.362981)
Fitting set 14...  (Loss, Acc) = (2.126470, 0.380580)
Fitting set 15...  (Loss, Acc) = (2.110874, 0.400521)
Fitting set 16...  (Loss, Acc) = (2.096904, 0.412109)
Fitting set 17...  (Loss, Acc) = (2.080665, 0.428768)
Fitting set 18...  (Loss, Acc) = (2.065265, 0.440972)
Fitting set 19...  (Loss, Acc) = (2.049479, 0.448602)
Fitting set 20...  (Loss, Acc) = (2.032223, 0.460547)
Fitting set 21...  (Loss, Acc) = (2.016439, 0.473586)
Fitting set 22...  (Loss, Acc) = (1.996922, 0.488281)
Fitting set 23...  (Loss, Acc) = (1.980086, 0.499321)
Fitting set 24...  (Loss, Acc) = (1.960719, 0.511393)
Fitting set 25...  (Loss, Acc) = (1.947011, 0.519375)
Fitting set 26...  (Loss, Acc) = (1.929373, 0.530349)
Fitting set 27...  (Loss, Acc) = (1.909454, 0.540799)
Fitting set 28...  (Loss, Acc) = (1.894096, 0.546596)
Fitting set 29...  (Loss, Acc) = (1.874574, 0.556034)
Fitting set 30...  (Loss, Acc) = (1.858988, 0.562500)
Fitting set 31...  (Loss, Acc) = (1.840485, 0.569052)
Fitting set 32...  (Loss, Acc) = (1.821290, 0.575928)
Fitting set 33...  (Loss, Acc) = (1.801637, 0.583333)
Fitting set 34...  (Loss, Acc) = (1.783776, 0.588006)
Fitting set 35...  (Loss, Acc) = (1.766005, 0.593080)
Fitting set 36...  (Loss, Acc) = (1.746479, 0.598958)
Fitting set 37...  (Loss, Acc) = (1.727118, 0.604730)
Fitting set 38...  (Loss, Acc) = (1.709816, 0.609375)
Fitting set 39...  (Loss, Acc) = (1.692258, 0.614784)
Fitting set 40...  (Loss, Acc) = (1.674974, 0.618164)
Fitting set 41...  (Loss, Acc) = (1.657619, 0.621570)
Fitting set 42...  (Loss, Acc) = (1.641050, 0.625744)
Fitting set 43...  (Loss, Acc) = (1.624397, 0.629542)
Fitting set 44...  (Loss, Acc) = (1.608801, 0.633523)
Fitting set 45...  (Loss, Acc) = (1.593297, 0.636458)
Fitting set 46...  (Loss, Acc) = (1.577476, 0.639606)
Fitting set 47...  (Loss, Acc) = (1.560293, 0.643451)
Fitting set 48...  (Loss, Acc) = (1.547326, 0.645996)
Data set_nums shuffled


 EPOCH 1 -- COMPLETE
```

(continues on next page)

(continued from previous page)

```
Validating...
Validating set 1...  (Loss, Acc) = (0.783646, 0.835938)
Validating set 2...  (Loss, Acc) = (0.765299, 0.851562)
Validating set 3...  (Loss, Acc) = (0.769534, 0.851562)
Validating set 4...  (Loss, Acc) = (0.764879, 0.853516)
Validating set 5...  (Loss, Acc) = (0.758425, 0.851562)
Validating set 6...  (Loss, Acc) = (0.764723, 0.843750)
Validating set 7...  (Loss, Acc) = (0.764539, 0.842634)
Validating set 8...  (Loss, Acc) = (0.767474, 0.844727)
Validating set 9...  (Loss, Acc) = (0.767794, 0.845486)
Validating set 10... (Loss, Acc) = (0.763187, 0.843750)
Validating set 11... (Loss, Acc) = (0.755877, 0.843750)
Validating set 12... (Loss, Acc) = (0.755090, 0.841797)
Validating set 13... (Loss, Acc) = (0.744053, 0.846154)
Validating set 14... (Loss, Acc) = (0.741029, 0.847098)
Validating set 15... (Loss, Acc) = (0.746417, 0.844271)
Validating set 16... (Loss, Acc) = (0.747256, 0.845215)
Validating set 17... (Loss, Acc) = (0.746406, 0.846048)
Validating set 18... (Loss, Acc) = (0.746663, 0.845486)
Validating set 19... (Loss, Acc) = (0.744822, 0.846628)
Validating set 20... (Loss, Acc) = (0.744538, 0.844922)
Validating set 21... (Loss, Acc) = (0.745785, 0.841890)
Validating set 22... (Loss, Acc) = (0.746459, 0.841619)
Validating set 23... (Loss, Acc) = (0.749001, 0.838995)
Validating set 24... (Loss, Acc) = (0.748322, 0.837565)
Validating set 25... (Loss, Acc) = (0.748221, 0.837500)
Validating set 26... (Loss, Acc) = (0.748553, 0.837740)
Validating set 27... (Loss, Acc) = (0.750275, 0.837384)
Validating set 28... (Loss, Acc) = (0.749730, 0.837612)
Validating set 29... (Loss, Acc) = (0.750421, 0.836476)
Validating set 30... (Loss, Acc) = (0.751461, 0.835677)
Validating set 31... (Loss, Acc) = (0.752642, 0.834677)
Validating set 32... (Loss, Acc) = (0.752270, 0.834473)
Validating set 33... (Loss, Acc) = (0.751130, 0.835227)
Validating set 34... (Loss, Acc) = (0.751729, 0.834559)
Validating set 35... (Loss, Acc) = (0.756417, 0.832143)
Validating set 36... (Loss, Acc) = (0.757068, 0.832031)
TrainGenerator state saved
Model report generated and saved
Best model saved to dir\models\M0__model-adam__min.757
TrainGenerator state saved
Model report generated and saved
```




```
Fitting set 21...  (Loss, Acc) = (0.765751, 0.824219)
Fitting set 27...  (Loss, Acc) = (0.727158, 0.841797)
Fitting set 44...  (Loss, Acc) = (0.746188, 0.834635)
Fitting set 3...   (Loss, Acc) = (0.730069, 0.838867)
Fitting set 23...  (Loss, Acc) = (0.728389, 0.839844)
Fitting set 31...  (Loss, Acc) = (0.720288, 0.835286)
Fitting set 45...  (Loss, Acc) = (0.720948, 0.834263)
Fitting set 22...  (Loss, Acc) = (0.704299, 0.837402)
Fitting set 16...  (Loss, Acc) = (0.698179, 0.838108)
Fitting set 38...  (Loss, Acc) = (0.691929, 0.833984)
Fitting set 34...  (Loss, Acc) = (0.692961, 0.832031)
Fitting set 1...   (Loss, Acc) = (0.680281, 0.838216)
Fitting set 4...   (Loss, Acc) = (0.676445, 0.841046)
Fitting set 37...  (Loss, Acc) = (0.663750, 0.845145)
Fitting set 46...  (Loss, Acc) = (0.658366, 0.846094)
Fitting set 12...  (Loss, Acc) = (0.649041, 0.846924)
Fitting set 10...  (Loss, Acc) = (0.645140, 0.846278)
Fitting set 32...  (Loss, Acc) = (0.640361, 0.846137)
Fitting set 8...   (Loss, Acc) = (0.633576, 0.846423)
Fitting set 25...  (Loss, Acc) = (0.634411, 0.843555)
Fitting set 19...  (Loss, Acc) = (0.633591, 0.844308)
Fitting set 36...  (Loss, Acc) = (0.627558, 0.844993)
Fitting set 2...   (Loss, Acc) = (0.619685, 0.847317)
Fitting set 18...  (Loss, Acc) = (0.615301, 0.848796)
Fitting set 42...  (Loss, Acc) = (0.613810, 0.848906)
Fitting set 26...  (Loss, Acc) = (0.606216, 0.850511)
Fitting set 6...   (Loss, Acc) = (0.602131, 0.850550)
Fitting set 48...  (Loss, Acc) = (0.603196, 0.849191)
Fitting set 15...  (Loss, Acc) = (0.598294, 0.850350)
Fitting set 35...  (Loss, Acc) = (0.595915, 0.849349)
Fitting set 39...  (Loss, Acc) = (0.590949, 0.851436)
Fitting set 40...  (Loss, Acc) = (0.587879, 0.851929)
Fitting set 41...  (Loss, Acc) = (0.583732, 0.852391)
Fitting set 29...  (Loss, Acc) = (0.578788, 0.853516)
Fitting set 17...  (Loss, Acc) = (0.576870, 0.853906)
Fitting set 5...   (Loss, Acc) = (0.571825, 0.855360)
Fitting set 11...  (Loss, Acc) = (0.565289, 0.856524)
Fitting set 33...  (Loss, Acc) = (0.563101, 0.856394)
Fitting set 30...  (Loss, Acc) = (0.560148, 0.856671)
```

(continues on next page)

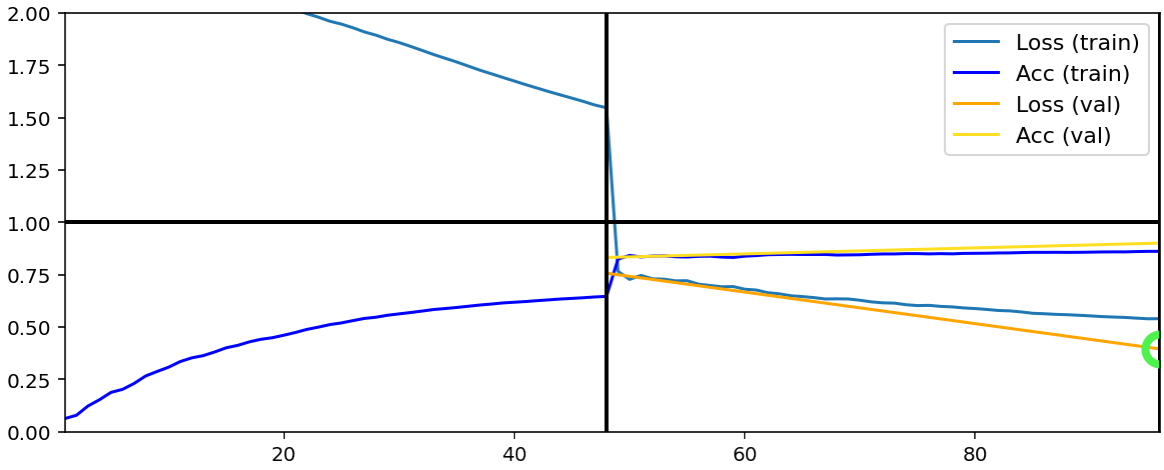
(continued from previous page)

```
Fitting set 9...   (Loss, Acc) = (0.558437, 0.856348)
Fitting set 43...  (Loss, Acc) = (0.555807, 0.856612)
Fitting set 13...  (Loss, Acc) = (0.552969, 0.857422)
Fitting set 20...  (Loss, Acc) = (0.549530, 0.858376)
Fitting set 7...   (Loss, Acc) = (0.547364, 0.858754)
Fitting set 14...  (Loss, Acc) = (0.545547, 0.858767)
Fitting set 47...  (Loss, Acc) = (0.542079, 0.860139)
Fitting set 24...  (Loss, Acc) = (0.539119, 0.861287)
Fitting set 28...  (Loss, Acc) = (0.539579, 0.861247)
Data set_nums shuffled



 EPOCH 2 -- COMPLETE



Validating...
Validating set 1...  (Loss, Acc) = (0.355274, 0.921875)
Validating set 2...  (Loss, Acc) = (0.348840, 0.921875)
Validating set 3...  (Loss, Acc) = (0.387879, 0.914062)
Validating set 4...  (Loss, Acc) = (0.394017, 0.908203)
Validating set 5...  (Loss, Acc) = (0.399065, 0.903125)
Validating set 6...  (Loss, Acc) = (0.397540, 0.903646)
Validating set 7...  (Loss, Acc) = (0.392841, 0.906250)
Validating set 8...  (Loss, Acc) = (0.396953, 0.906250)
Validating set 9...  (Loss, Acc) = (0.396412, 0.907986)
Validating set 10... (Loss, Acc) = (0.399894, 0.908594)
Validating set 11... (Loss, Acc) = (0.389734, 0.909091)
Validating set 12... (Loss, Acc) = (0.388978, 0.908203)
Validating set 13... (Loss, Acc) = (0.378990, 0.910457)
Validating set 14... (Loss, Acc) = (0.379070, 0.908482)
Validating set 15... (Loss, Acc) = (0.388130, 0.907292)
Validating set 16... (Loss, Acc) = (0.391481, 0.905762)
Validating set 17... (Loss, Acc) = (0.385940, 0.906250)
Validating set 18... (Loss, Acc) = (0.386285, 0.904948)
Validating set 19... (Loss, Acc) = (0.382503, 0.905428)
Validating set 20... (Loss, Acc) = (0.380249, 0.906641)
Validating set 21... (Loss, Acc) = (0.385483, 0.903646)
Validating set 22... (Loss, Acc) = (0.383873, 0.904474)
Validating set 23... (Loss, Acc) = (0.386388, 0.903872)
Validating set 24... (Loss, Acc) = (0.383956, 0.902995)
Validating set 25... (Loss, Acc) = (0.384522, 0.903438)
Validating set 26... (Loss, Acc) = (0.385134, 0.903546)
Validating set 27... (Loss, Acc) = (0.388839, 0.902199)
Validating set 28... (Loss, Acc) = (0.388487, 0.902065)
Validating set 29... (Loss, Acc) = (0.389055, 0.903556)
Validating set 30... (Loss, Acc) = (0.389983, 0.902865)
Validating set 31... (Loss, Acc) = (0.391534, 0.902470)
Validating set 32... (Loss, Acc) = (0.391110, 0.902588)
Validating set 33... (Loss, Acc) = (0.391102, 0.902225)
Validating set 34... (Loss, Acc) = (0.388916, 0.902803)
Validating set 35... (Loss, Acc) = (0.395877, 0.900223)
Validating set 36... (Loss, Acc) = (0.395200, 0.900608)
TrainGenerator state saved
Model report generated and saved
Best model saved to dir\models\M0__model-adam__min.395
```

(continues on next page)

(continued from previous page)

```
TrainGenerator state saved
Model report generated and saved
```




```
Fitting set 40...   (Loss, Acc) = (0.393554, 0.910156)
Fitting set 30...   (Loss, Acc) = (0.389306, 0.912109)
Fitting set 17...   (Loss, Acc) = (0.409685, 0.904948)
Fitting set 10...   (Loss, Acc) = (0.399228, 0.899414)
Fitting set 19...   (Loss, Acc) = (0.411247, 0.894531)
Fitting set 9...    (Loss, Acc) = (0.411084, 0.888672)
Fitting set 24...   (Loss, Acc) = (0.401326, 0.893415)
Fitting set 14...   (Loss, Acc) = (0.403191, 0.895020)
Fitting set 8...    (Loss, Acc) = (0.395733, 0.897135)
Fitting set 35...   (Loss, Acc) = (0.398654, 0.895703)
Fitting set 45...   (Loss, Acc) = (0.400620, 0.894531)
Fitting set 15...   (Loss, Acc) = (0.395943, 0.894857)
Fitting set 36...   (Loss, Acc) = (0.390992, 0.897536)
Fitting set 4...    (Loss, Acc) = (0.390960, 0.898717)
Fitting set 39...   (Loss, Acc) = (0.386528, 0.899740)
Fitting set 43...   (Loss, Acc) = (0.386909, 0.899170)
Fitting set 6...    (Loss, Acc) = (0.385125, 0.898667)
Fitting set 16...   (Loss, Acc) = (0.383201, 0.899523)
Fitting set 13...   (Loss, Acc) = (0.381569, 0.899877)
Fitting set 11...   (Loss, Acc) = (0.375037, 0.901367)
Fitting set 23...   (Loss, Acc) = (0.376780, 0.899740)
Fitting set 31...   (Loss, Acc) = (0.377788, 0.899680)
Fitting set 32...   (Loss, Acc) = (0.379219, 0.898607)
Fitting set 33...   (Loss, Acc) = (0.379287, 0.897949)
Fitting set 18...   (Loss, Acc) = (0.377729, 0.899844)
Fitting set 21...   (Loss, Acc) = (0.376463, 0.899790)
Fitting set 38...   (Loss, Acc) = (0.372582, 0.900608)
Fitting set 26...   (Loss, Acc) = (0.367968, 0.901088)
Fitting set 22...   (Loss, Acc) = (0.365197, 0.902613)
Fitting set 44...   (Loss, Acc) = (0.365697, 0.902734)
Fitting set 3...    (Loss, Acc) = (0.364059, 0.903604)
Fitting set 37...   (Loss, Acc) = (0.360719, 0.904419)
Fitting set 7...    (Loss, Acc) = (0.359887, 0.904711)
Fitting set 1...    (Loss, Acc) = (0.356046, 0.906135)
Fitting set 29...   (Loss, Acc) = (0.354691, 0.906585)
Fitting set 48...   (Loss, Acc) = (0.357704, 0.905707)
```

(continues on next page)

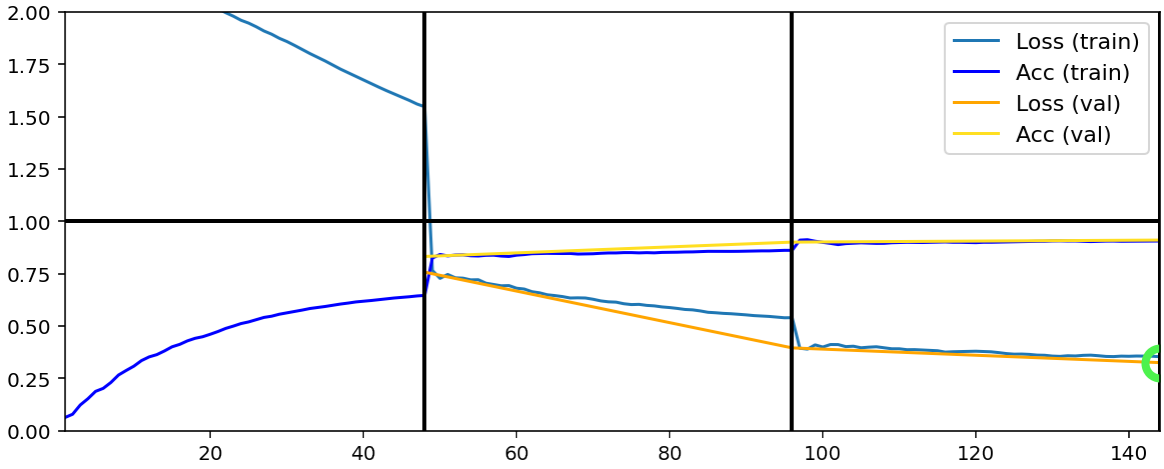
(continued from previous page)

```
Fitting set 41...  (Loss, Acc) = (0.356611, 0.905933)
Fitting set 28...  (Loss, Acc) = (0.359539, 0.904708)
Fitting set 25...  (Loss, Acc) = (0.360531, 0.903145)
Fitting set 5...   (Loss, Acc) = (0.357812, 0.904395)
Fitting set 12...  (Loss, Acc) = (0.354390, 0.905393)
Fitting set 27...  (Loss, Acc) = (0.353634, 0.905599)
Fitting set 34...  (Loss, Acc) = (0.355857, 0.904524)
Fitting set 20...  (Loss, Acc) = (0.355108, 0.904741)
Fitting set 42...  (Loss, Acc) = (0.356337, 0.904427)
Fitting set 47...  (Loss, Acc) = (0.356071, 0.904806)
Fitting set 46...  (Loss, Acc) = (0.355477, 0.905502)
Fitting set 2...   (Loss, Acc) = (0.353324, 0.905843)
Data set_nums shuffled


_____________________
 EPOCH 3 -- COMPLETE


Validating...
Validating set 1...  (Loss, Acc) = (0.248495, 0.945312)
Validating set 2...  (Loss, Acc) = (0.256513, 0.933594)
Validating set 3...  (Loss, Acc) = (0.303261, 0.924479)
Validating set 4...  (Loss, Acc) = (0.315434, 0.919922)
Validating set 5...  (Loss, Acc) = (0.323978, 0.917188)
Validating set 6...  (Loss, Acc) = (0.320035, 0.916667)
Validating set 7...  (Loss, Acc) = (0.311358, 0.917411)
Validating set 8...  (Loss, Acc) = (0.318938, 0.911133)
Validating set 9...  (Loss, Acc) = (0.318156, 0.910590)
Validating set 10... (Loss, Acc) = (0.324514, 0.910937)
Validating set 11... (Loss, Acc) = (0.314028, 0.914773)
Validating set 12... (Loss, Acc) = (0.314245, 0.913411)
Validating set 13... (Loss, Acc) = (0.305751, 0.915264)
Validating set 14... (Loss, Acc) = (0.305372, 0.915179)
Validating set 15... (Loss, Acc) = (0.316371, 0.914062)
Validating set 16... (Loss, Acc) = (0.321232, 0.911133)
Validating set 17... (Loss, Acc) = (0.314669, 0.914522)
Validating set 18... (Loss, Acc) = (0.315064, 0.912760)
Validating set 19... (Loss, Acc) = (0.310267, 0.914062)
Validating set 20... (Loss, Acc) = (0.307732, 0.914844)
Validating set 21... (Loss, Acc) = (0.316089, 0.912202)
Validating set 22... (Loss, Acc) = (0.314979, 0.912642)
Validating set 23... (Loss, Acc) = (0.317234, 0.912024)
Validating set 24... (Loss, Acc) = (0.314426, 0.912109)
Validating set 25... (Loss, Acc) = (0.315502, 0.911875)
Validating set 26... (Loss, Acc) = (0.315770, 0.911959)
Validating set 27... (Loss, Acc) = (0.319364, 0.910880)
Validating set 28... (Loss, Acc) = (0.318615, 0.910435)
Validating set 29... (Loss, Acc) = (0.319337, 0.911369)
Validating set 30... (Loss, Acc) = (0.320033, 0.910677)
Validating set 31... (Loss, Acc) = (0.321572, 0.911038)
Validating set 32... (Loss, Acc) = (0.320362, 0.910889)
Validating set 33... (Loss, Acc) = (0.320599, 0.910748)
Validating set 34... (Loss, Acc) = (0.318106, 0.911535)
Validating set 35... (Loss, Acc) = (0.325595, 0.909598)
Validating set 36... (Loss, Acc) = (0.324875, 0.910373)
```

(continues on next page)

(continued from previous page)

```
TrainGenerator state saved
Model report generated and saved
Best model saved to dir\models\M0__model-adam__min.325
TrainGenerator state saved
Model report generated and saved
```




```
Training has concluded.
```

3.2.8 Delve deeper

DeepTrain offers much beyond the minimals; it's suggested to proceed with the advanced example before exploring others.

3.3 Deeper into DeepTrain

This example assumes you've read `basic.ipynb`, and covers:

- Multi-phase training
  - Changing loss & hyperparameters between phases
- Callback streaming images to directory
- Saving & loading
- Variable-layer model building

[1]:
```
import os
from tensorflow.keras.layers import Input, Conv2D, UpSampling2D, Dropout
from tensorflow.keras.layers import BatchNormalization, Activation
from tensorflow.keras.models import Model

from deeptrain import TrainGenerator, DataGenerator
from deeptrain.callbacks import VizAE2D
```

3.3.1 Configuration

[2]:
```
# This scheme enables variable number of layers
def make_model(batch_shape, optimizer, loss, metrics,
               filters, kernel_size, strides, activation, up_sampling_2d,
               input_dropout, preout_dropout):
    """Compressing, denoising AutoEncoder."""
    ipt = Input(batch_shape=batch_shape)
    x   = Dropout(input_dropout)(ipt)

    configs = (activation, filters, kernel_size, strides, up_sampling_2d)
    for a, f, ks, s, ups in zip(*configs):
        x = UpSampling2D(ups)(x) if ups else x
        x = Conv2D(f, ks, strides=s, padding='same')(x)
        x = BatchNormalization()(x)
        x = Activation(a)(x)

    x   = Dropout(preout_dropout)(x)
    x   = Conv2D(1, (3, 3), strides=1, padding='same', activation='sigmoid')(x)
    out = x

    model = Model(ipt, out)
    model.compile(optimizer, loss, metrics=metrics)
    return model
```

[3]:
```
batch_size = 128
width, height, channels = 28, 28, 1
# 28x compression
MODEL_CFG = dict(
    batch_shape=(batch_size, width, height, channels),
    loss='mse',
    metrics=None,
    optimizer='nadam',
    activation=['relu'] * 5,
    filters=[6, 12, 2, 6, 12],
    kernel_size=[(3, 3)] * 5,
    strides=[(2, 2), (2, 2), 1, 1, 1],
    up_sampling_2d=[None, None, None, (2, 2), (2, 2)],
    input_dropout=.5,
    preout_dropout=.4,
)
datadir = os.path.join("dir", "data", "image")
DATAGEN_CFG = dict(
    data_path=os.path.join(datadir, 'train'),
    batch_size=batch_size,
    shuffle=True,
    superbatch_set_nums='all',
)
VAL_DATAGEN_CFG = dict(
    data_path=os.path.join(datadir, 'val'),
    batch_size=batch_size,
    shuffle=False,
    superbatch_set_nums='all',
)
```

- `key_metric`: the metric that decides the "best" model
- `max_is_best`: whether greater `key_metric` is better (we seek to minimize loss)

- `input_as_labels`: y = x, or model.fit(x, x)
- `eval_fn`: function to use in validation
- `val_freq`: how often to validate (default: every epoch)
- `plot_history_freq`: how often to plot history (default: every epoch)
- `unique_checkpoint_freq`: how often to checkpoint (default: every epoch)
- `model_save_kw`: kwargs passed to `model.save()`. Exclude optimizer since we'll save its (and model's) weights separately to load later
- `model_name_configs`: set which model attributes to include in automatic name generation, and their (shortened) aliases

[4]:
```
TRAINGEN_CFG = dict(
    epochs=6,
    logs_dir=os.path.join('dir', 'logs'),
    best_models_dir=os.path.join('dir', 'models'),
    model_configs=MODEL_CFG,
    key_metric='mae',
    max_is_best=False,
    input_as_labels=True,
    eval_fn='predict',
    val_freq={'epoch': 2},
    plot_history_freq={'epoch': 2},
    unique_checkpoint_freq={'epoch': 2},
    iter_verbosity=0,  # silence per-iteration progress printing (to spare notebook␣
↪length)
    model_save_kw=dict(include_optimizer=False, save_format='h5'),
    model_name_configs=dict(input_dropout='idp', preout_dropout='pdp',
                            optimizer='', lr='', best_key_metric=None),
)
```

3.3.2 Create visualization callback

[5]:
```
TRAINGEN_CFG['callbacks'] = [VizAE2D(n_images=8, save_images=True)]
```

3.3.3 Create training objects

[6]:
```
model = make_model(**MODEL_CFG)
dg    = DataGenerator(**DATAGEN_CFG)
vdg   = DataGenerator(**VAL_DATAGEN_CFG)
tg    = TrainGenerator(model, dg, vdg, **TRAINGEN_CFG)

# save optimizer weights & attrs to load later
_ = tg.saveskip_list.pop(tg.saveskip_list.index('optimizer_state'))
```

```
WARNING: multiple file extensions found in `path`; only .npy will be used
Discovered 48 files with matching format
48 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

WARNING: multiple file extensions found in `path`; only .npy will be used
```

(continues on next page)

(continued from previous page)

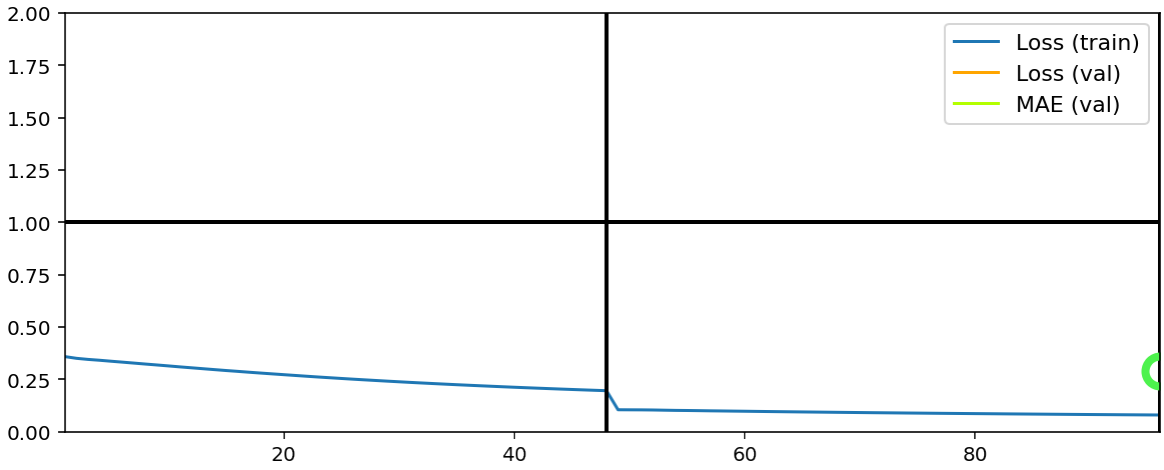
```
Discovered 36 files with matching format
36 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

NOTE: will exclude `labels` from saving when `input_as_labels=True`; to keep 'labels',
↪ add '{labels}'to `saveskip_list` instead
Preloading superbatch ... WARNING: multiple file extensions found in `path`; only .
↪npy will be used
Discovered 48 files with matching format
... finished, w/ 6144 total samples
Train initial data prepared
Preloading superbatch ... WARNING: multiple file extensions found in `path`; only .
↪npy will be used
Discovered 36 files with matching format
... finished, w/ 4608 total samples
Val initial data prepared
Logging ON; directory (new): dir\logs\M28__model-idp.5-pdp.4-nadam__min999.000
```

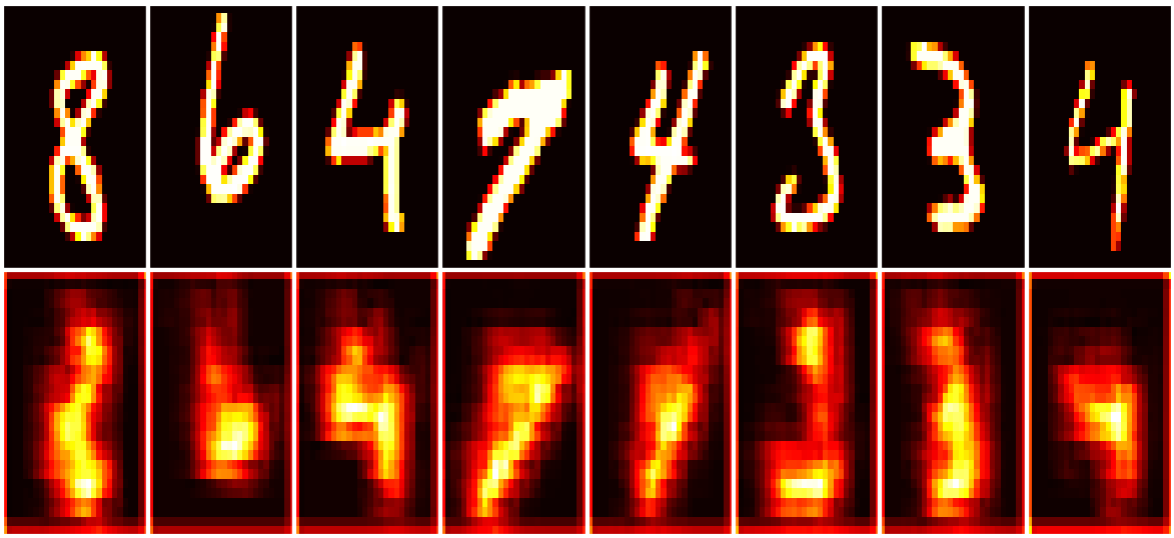
Don't mind the warnings; they're due to putting `labels.h5` in the same directory as data (and not using it for the autoencoder).

3.3.4 Train

[7]:
```
tg.train()
```

```
Data set_nums shuffled


_________________________
 EPOCH 1 -- COMPLETE
_________________________


Data set_nums shuffled


_________________________
 EPOCH 2 -- COMPLETE
_________________________



Validating...
TrainGenerator state saved
Model report generated and saved
Best model saved to dir\models\M28__model-idp.5-pdp.4-nadam__min.287
TrainGenerator state saved
Model report generated and saved
```



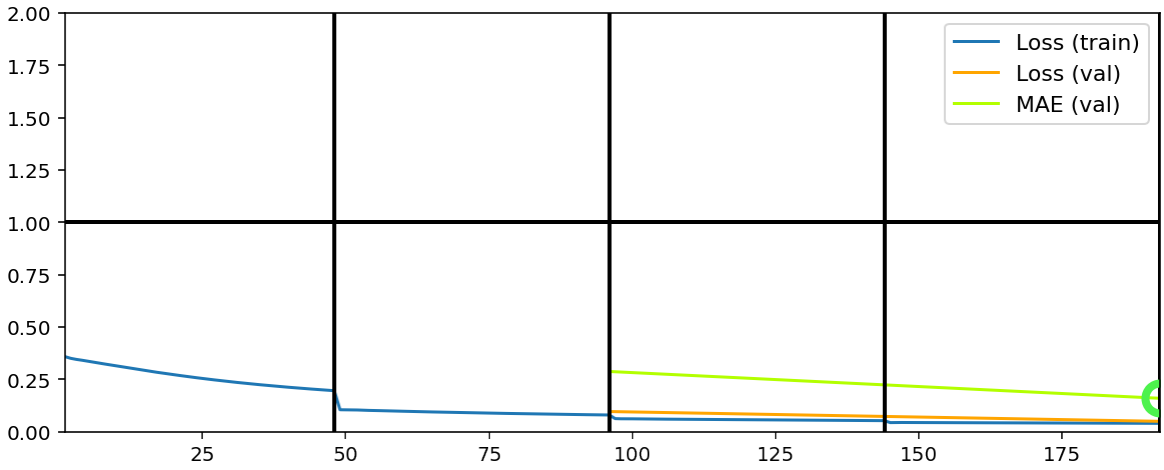


```
Data set_nums shuffled


_____________________
 EPOCH 3 -- COMPLETE


Data set_nums shuffled


_____________________
 EPOCH 4 -- COMPLETE



Validating...
TrainGenerator state saved
Model report generated and saved
Best model saved to dir\models\M28__model-idp.5-pdp.4-nadam__min.159
```

(continues on next page)

(continued from previous page)

```
TrainGenerator state saved
Model report generated and saved
```

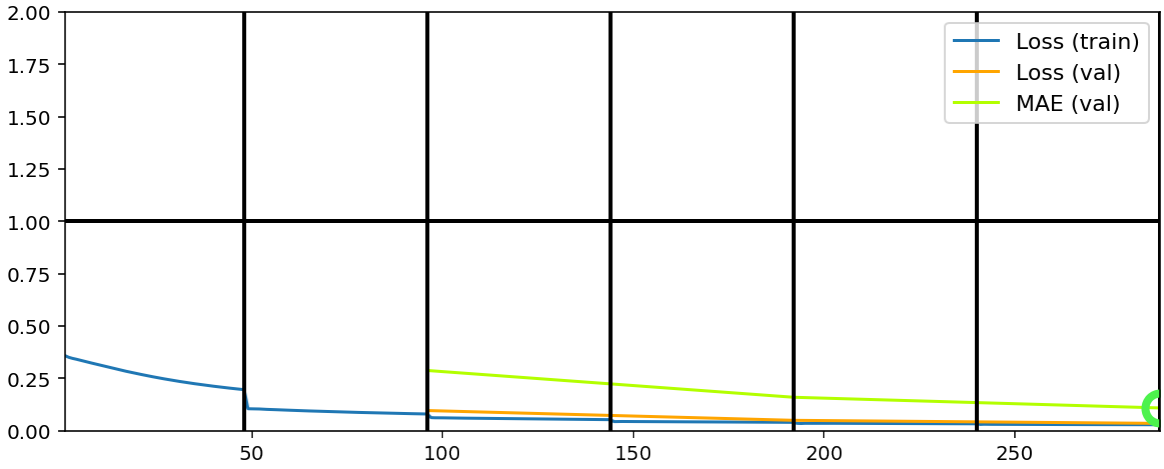


```
Data set_nums shuffled


_____________________
 EPOCH 5 -- COMPLETE


Data set_nums shuffled


_____________________
 EPOCH 6 -- COMPLETE


Validating...
```

(continues on next page)

(continued from previous page)

```
TrainGenerator state saved
Model report generated and saved
Best model saved to dir\models\M28__model-idp.5-pdp.4-nadam__min.108
TrainGenerator state saved
Model report generated and saved
```



```
Training has concluded.
```

3.3.5 Phase 2

Switch to *mean absolute error* loss; greater penalty to smaller errors forces better image resolution. Internally, TrainGenerator will append `'mae'` loss to same list as was `'mse'`.

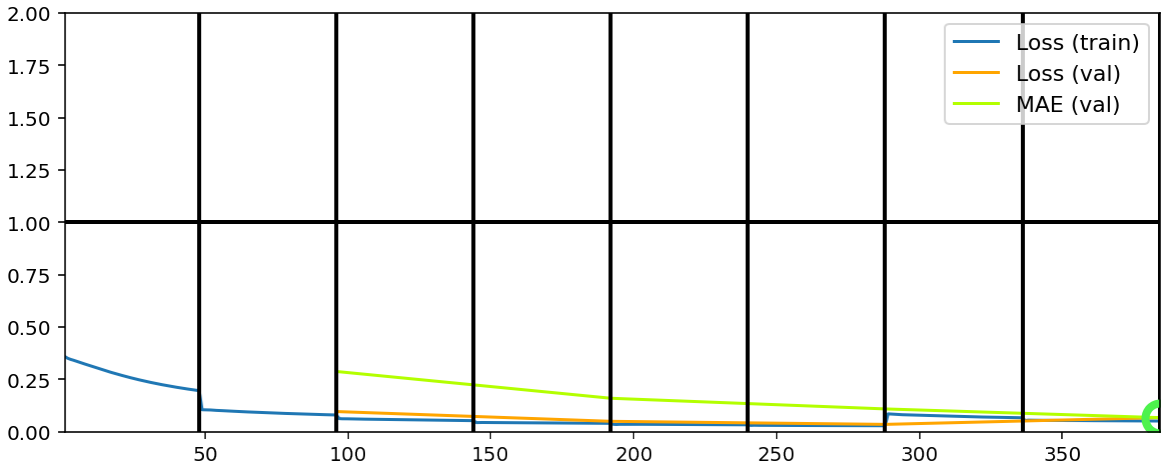
[8]:
```
tg.model.compile(MODEL_CFG['optimizer'], 'mae')
tg.epochs = 12
tg.train()
```

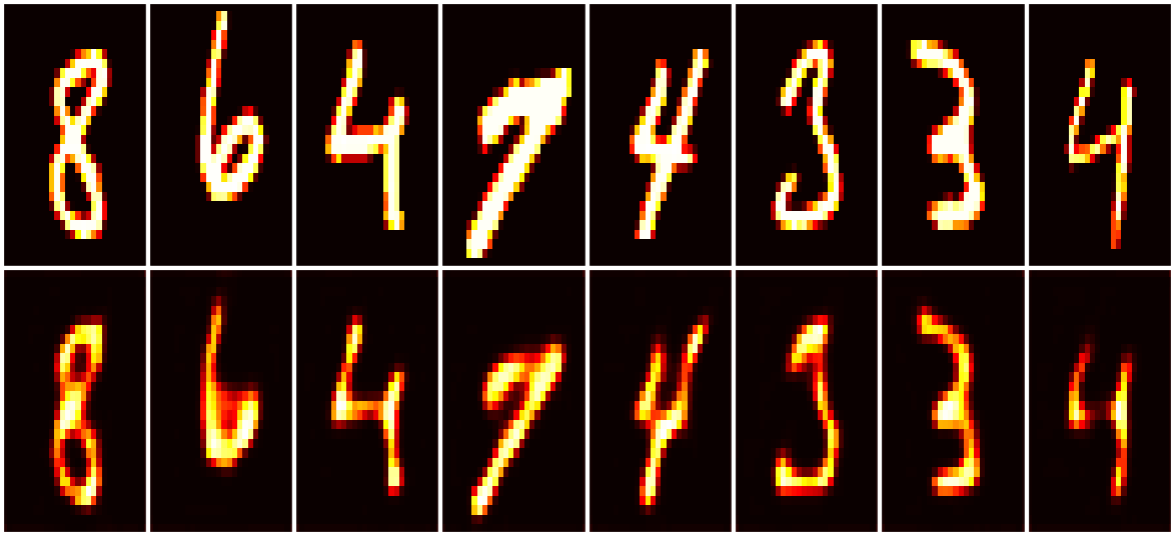
```
Data set_nums shuffled
```

(continues on next page)

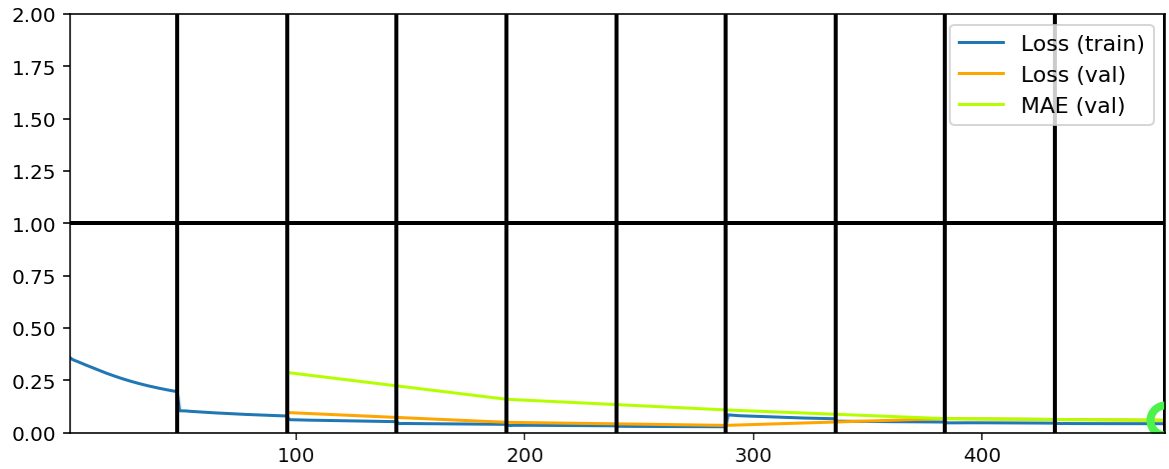
(continued from previous page)

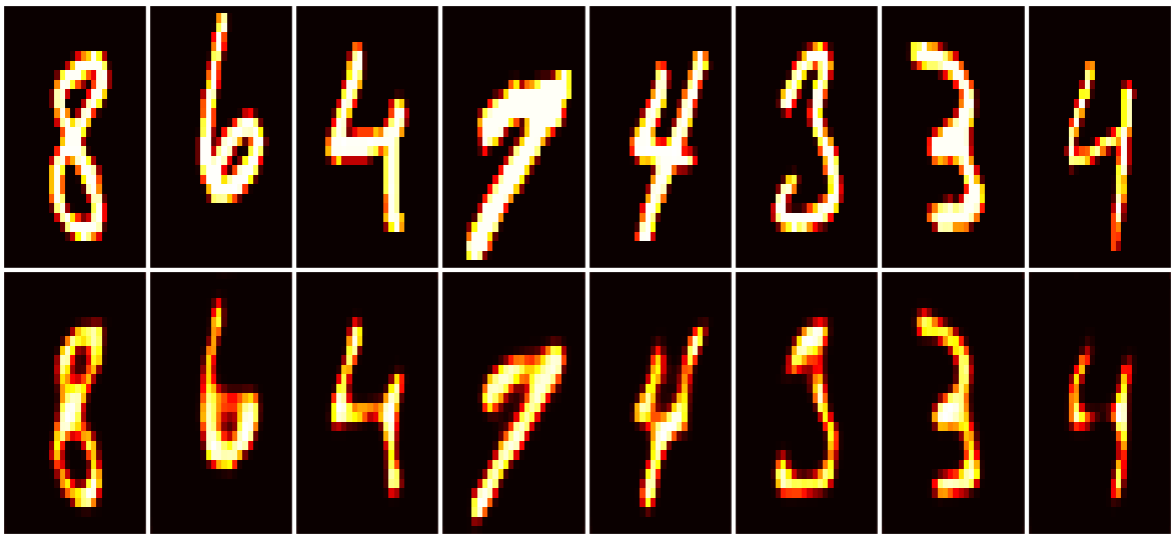
```
_____________________
 EPOCH 7 -- COMPLETE
_____________________


Data set_nums shuffled


_____________________
 EPOCH 8 -- COMPLETE
_____________________


Validating...
TrainGenerator state saved
Model report generated and saved
Best model saved to dir\models\M28__model-idp.5-pdp.4-nadam__min.067
TrainGenerator state saved
Model report generated and saved
```



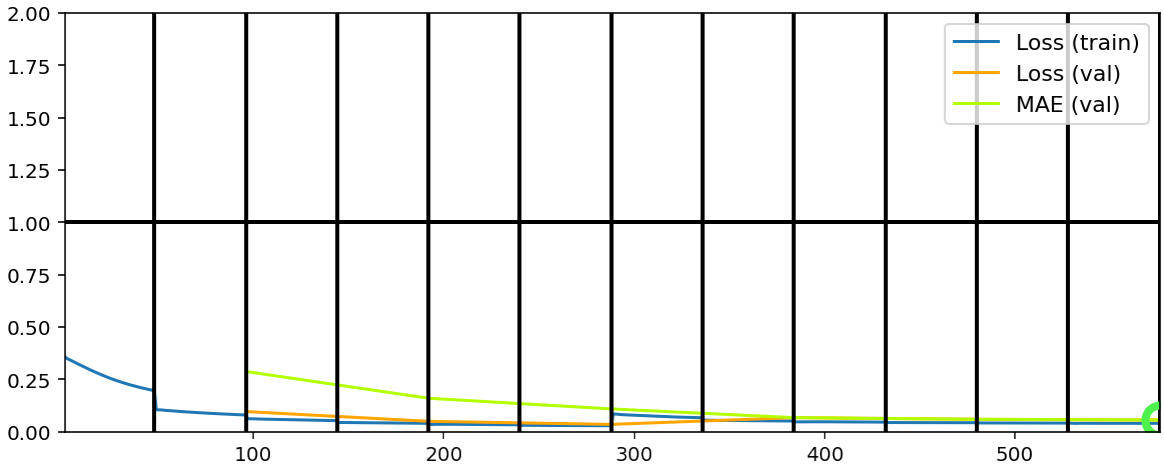


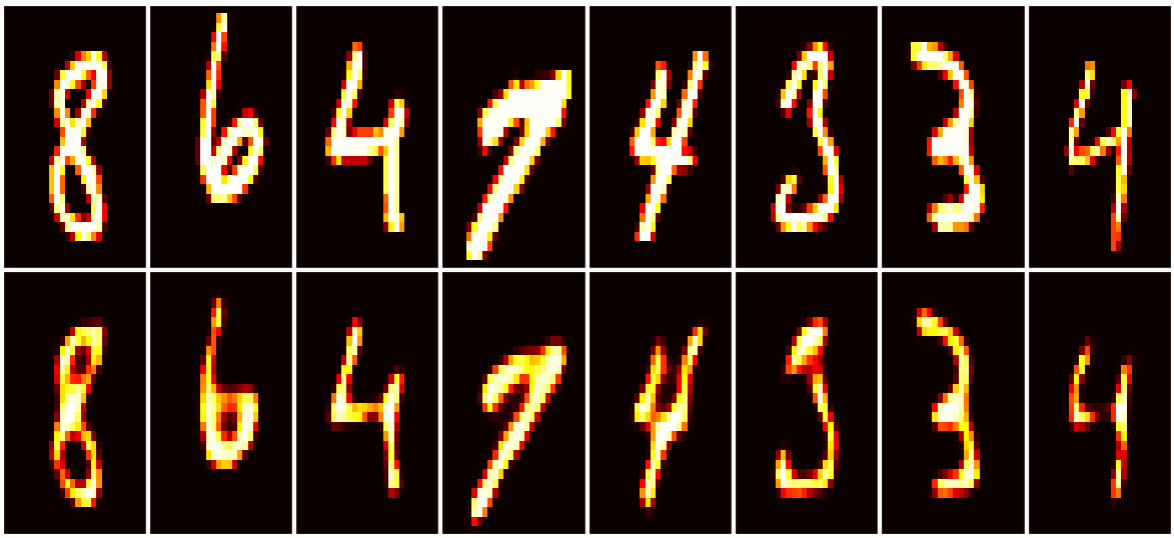
```
Data set_nums shuffled


_________________________
 EPOCH 9 -- COMPLETE
_________________________

Data set_nums shuffled


_________________________
 EPOCH 10 -- COMPLETE
_________________________


Validating...
TrainGenerator state saved
Model report generated and saved
Best model saved to dir\models\M28__model-idp.5-pdp.4-nadam__min.059
TrainGenerator state saved
Model report generated and saved
```





```
Data set_nums shuffled


_____________________
 EPOCH 11 -- COMPLETE


Data set_nums shuffled


_____________________
 EPOCH 12 -- COMPLETE


Validating...
TrainGenerator state saved
Model report generated and saved
Best model saved to dir\models\M28__model-idp.5-pdp.4-nadam__min.055
TrainGenerator state saved
Model report generated and saved
```





```
Training has concluded.
```

3.3.6 New session w/ changed hyperparams

[9]:
```
# get best save's model weights & TrainGenerator state
latest_best_weights = tg.get_last_log('weights', best=True)
latest_best_state   = tg.get_last_log('state',   best=True)

# destroy existing train objects
tg.destroy(confirm=True)
del model, dg, vdg, tg

# increase preout_dropout to strengthen regularization
MODEL_CFG['preout_dropout'] = .7
MODEL_CFG['loss'] = 'mae'
# `epochs` will load at 12, so need to increase
TRAINGEN_CFG['epochs'] = 20
TRAINGEN_CFG['loadpath'] = latest_best_state
```

(continues on next page)

(continued from previous page)

```
# ensure model_name uses prev model_num + 1, since using new hyperparams
TRAINGEN_CFG['new_model_num'] = False
# must re-instantiate callbacks object to hold new TrainGenerator
TRAINGEN_CFG['callbacks'] = [VizAE2D(n_images=8, save_images=True)]
```

```
>>>TrainGenerator DESTROYED
```

3.3.7 Create new train objects

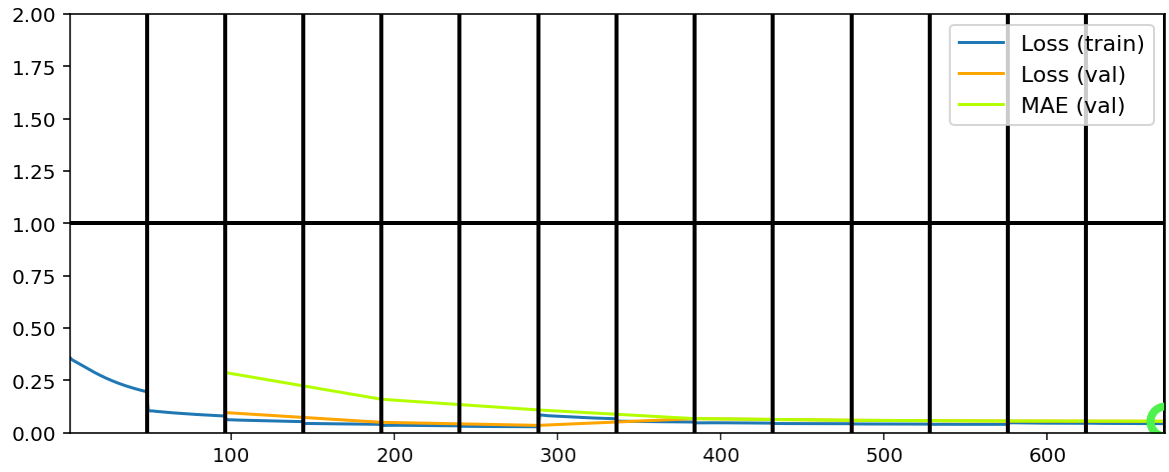
[10]:
```
model = make_model(**MODEL_CFG)
model.load_weights(latest_best_weights)

dg    = DataGenerator(**DATAGEN_CFG)
vdg   = DataGenerator(**VAL_DATAGEN_CFG)
tg    = TrainGenerator(model, dg, vdg, **TRAINGEN_CFG)
# can also load via `tg.load`, but passing in `loadpath` and starting a
# new session should work better
```

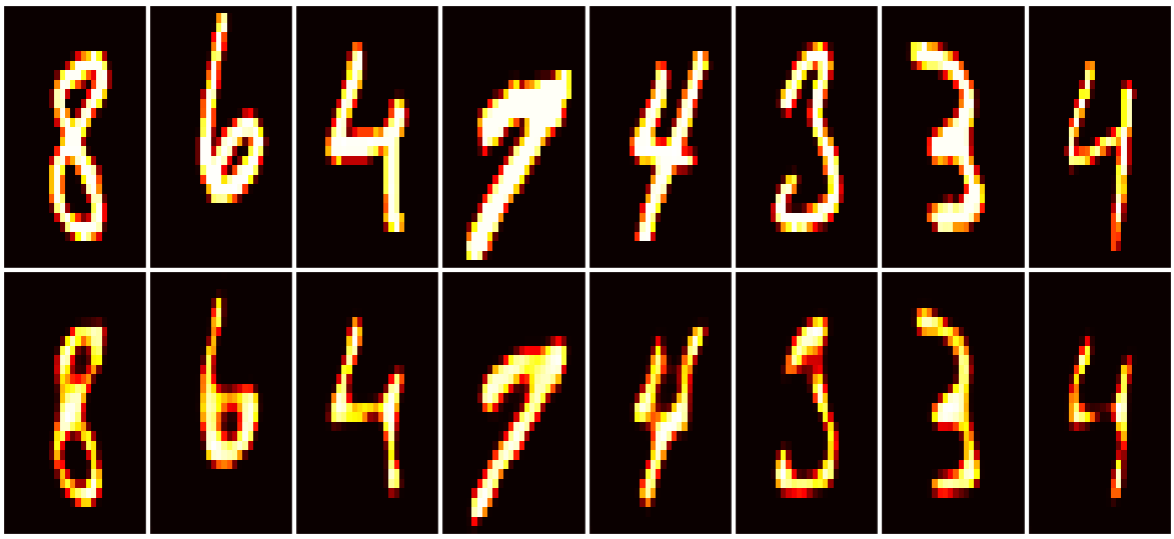
```
WARNING: multiple file extensions found in `path`; only .npy will be used
Discovered 48 files with matching format
48 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

WARNING: multiple file extensions found in `path`; only .npy will be used
Discovered 36 files with matching format
36 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

NOTE: will exclude `labels` from saving when `input_as_labels=True`; to keep 'labels',
↪ add '{labels}'to `saveskip_list` instead
Optimizer state loaded (& cleared from TrainGenerator)
TrainGenerator state loaded from dir\models\M28__model-idp.5-pdp.4-nadam__min.055__
↪state.h5
--Preloading excluded data based on datagen states ...
Preloading superbatch ... WARNING: multiple file extensions found in `path`; only .
↪npy will be used
Discovered 48 files with matching format
... finished, w/ 6144 total samples
Train initial data prepared
Preloading superbatch ... WARNING: multiple file extensions found in `path`; only .
↪npy will be used
Discovered 36 files with matching format
... finished, w/ 4608 total samples
Val initial data prepared
... finished--
Logging ON; directory (new): dir\logs\M29__model-idp.5-pdp.7-nadam__min.055
```

3.3.8 Train

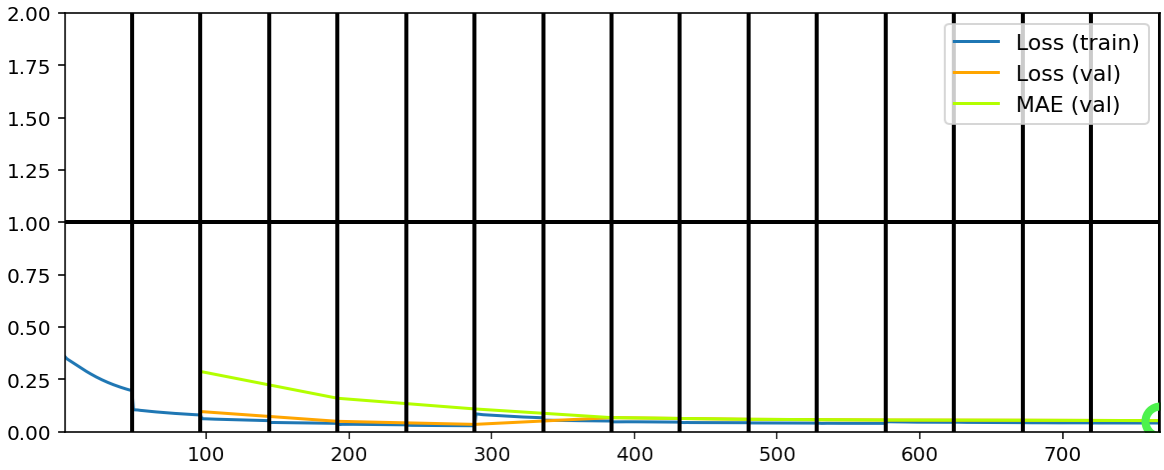
[11]:
```
tg.train()
```

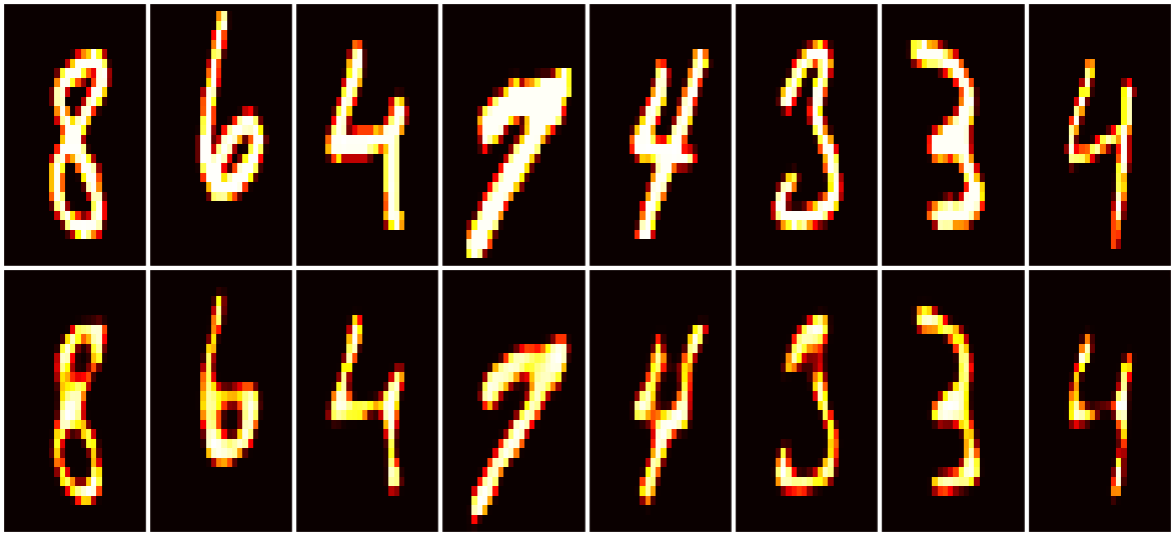
```
Data set_nums shuffled


_________________________
 EPOCH 13 -- COMPLETE
_________________________

Data set_nums shuffled


_________________________
 EPOCH 14 -- COMPLETE
_________________________


Validating...
TrainGenerator state saved
Model report generated and saved
Best model saved to dir\models\M29__model-idp.5-pdp.7-nadam__min.054
TrainGenerator state saved
Model report generated and saved
```





```
Data set_nums shuffled


_____________________
 EPOCH 15 -- COMPLETE


Data set_nums shuffled


_____________________
 EPOCH 16 -- COMPLETE


Validating...
TrainGenerator state saved
Model report generated and saved
Best model saved to dir\models\M29__model-idp.5-pdp.7-nadam__min.053
TrainGenerator state saved
Model report generated and saved
```



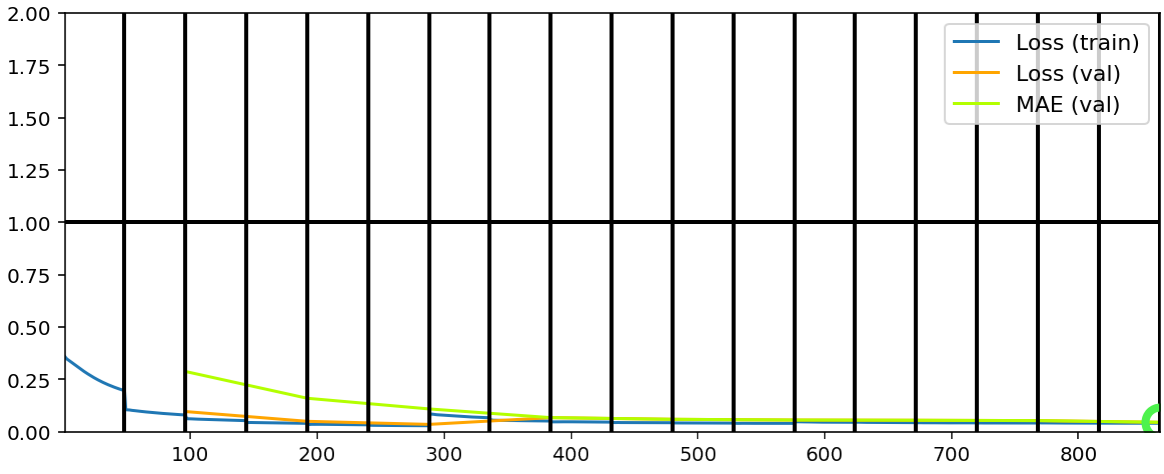


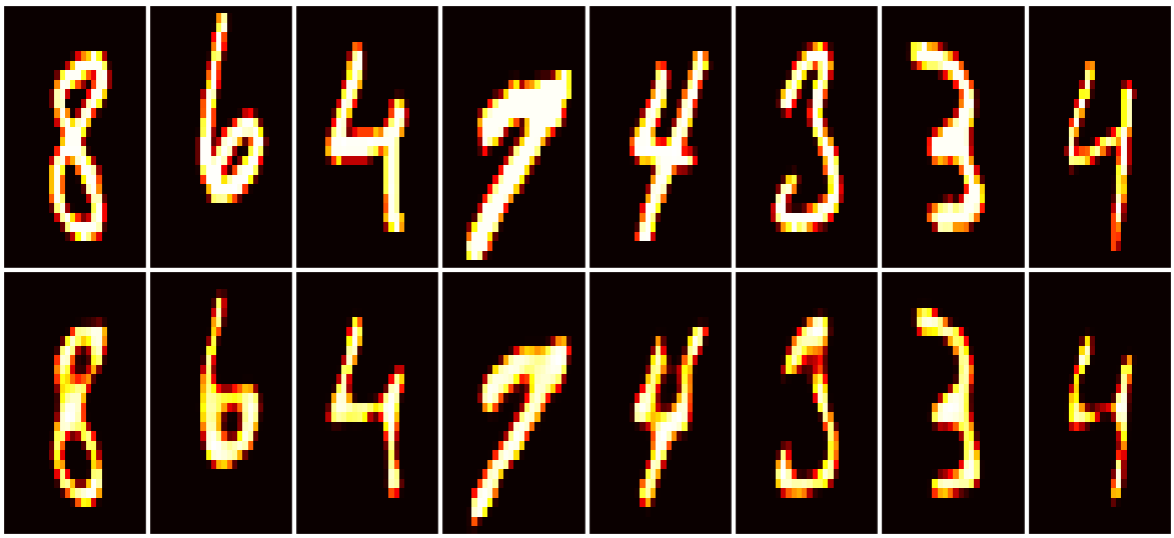
```
Data set_nums shuffled


_____________________
 EPOCH 17 -- COMPLETE


Data set_nums shuffled


_____________________
 EPOCH 18 -- COMPLETE



Validating...
TrainGenerator state saved
Model report generated and saved
Best model saved to dir\models\M29__model-idp.5-pdp.7-nadam__min.044
```

(continues on next page)

(continued from previous page)

```
TrainGenerator state saved
Model report generated and saved
```



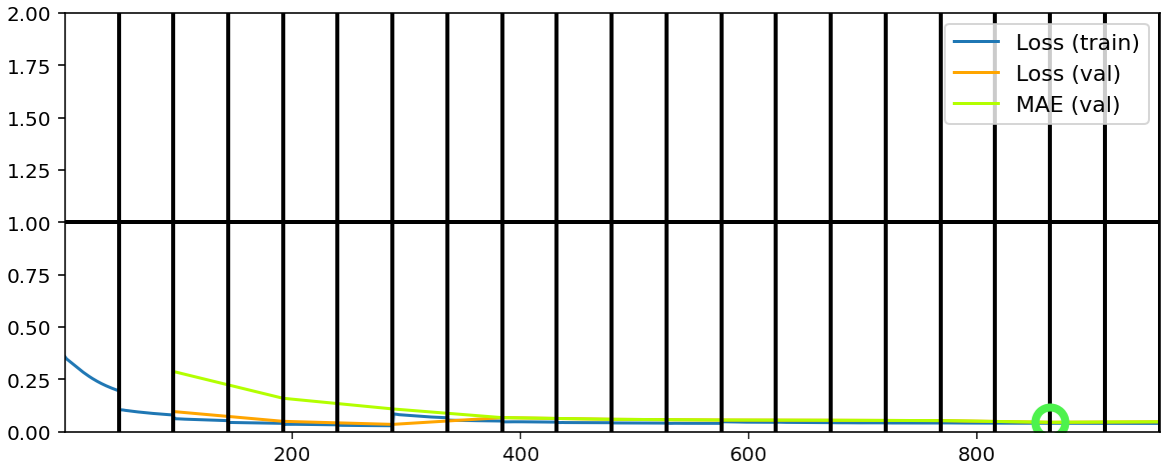


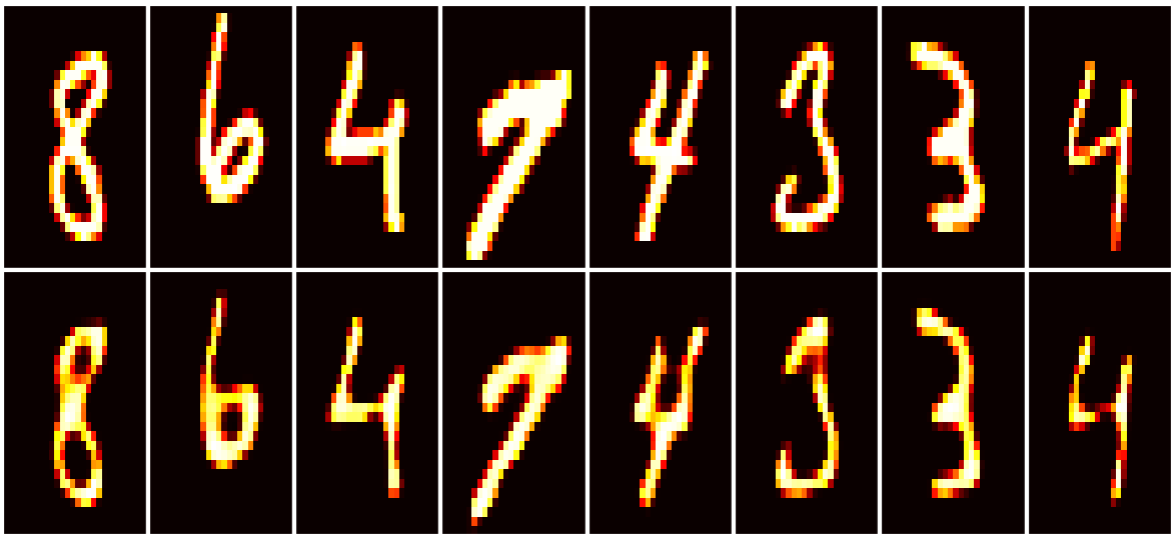
```
Data set_nums shuffled


_____________________
 EPOCH 19 -- COMPLETE

Data set_nums shuffled


_____________________
 EPOCH 20 -- COMPLETE


Validating...
```

(continues on next page)

(continued from previous page)

```
TrainGenerator state saved
Model report generated and saved
```





```
Training has concluded.
```

[12]:
```
cwd = os.getcwd()
print("Checkpoints can be found in", os.path.join(cwd, tg.logdir))
print("Best model can be found in",  os.path.join(cwd, tg.best_models_dir))
print("AE progress can be found in", os.path.join(cwd, tg.logdir, 'misc'))
```

```
Checkpoints can be found in C:\deeptrain\examples\dir\logs\M29__model-idp.5-pdp.7-
↪nadam__min.055
Best model can be found in C:\deeptrain\examples\dir\models
AE progress can be found in C:\deeptrain\examples\dir\logs\M29__model-idp.5-pdp.7-
↪nadam__min.055\misc
```

3.3.9 Inspect generated logs

Our callback is configured to write images to `tg.logdir + '/misc'`, and there's further the "report" of the train state. Open the last directory named above, also one of the *previous* model number (since we reinstantiated in Phase 2), to see the callbacks' outputs. Below we'll look at the last generated report (viewed better by opening the image file):

[13]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

img = mpimg.imread(tg.get_last_log('report'))
_, ax = plt.subplots(figsize=(20, 11))
ax.imshow(img)
ax.set_axis_off()
plt.show()
```

Report generation, including which attributes to include or exclude, is configured in `util/configs.py`, or overridden by the `report_configs` kwarg to `TrainGenerator`. We can configure it to display the most relevant info, and discard the rest. The train state should more completely be saved in the `__state.h5` file, loaded via `tg.load()`.

3.4 Recommended Usage

- Keep definitions in a separate file, init train objects and train in the "main" file
- In Jupyter, keep definitions in a collapsible cell
- The idea is to keep the workspace clean and reserve space for code that changes often or is used after training

3.5 Examples

3.5.1 Timeseries Classification

This example assumes you've read `advanced.py`, and covers:

- Timeseries binary classification on real data
- Windowed data format; sequence length 188, 4 windows -> 47 points per window
- Binary classification visuals
- Using class weights to handle imbalance

[1]:
```
import deeptrain
deeptrain.util.misc.append_examples_dir_to_sys_path()  # for `from utils import`

from utils import TS_CONFIGS as C
from utils import init_session, make_timeseries_classifier
from see_rnn import features_1D, rnn_histogram, rnn_heatmap
```

Dataset info

- PTB Diagnostic ECG Database - https://www.kaggle.com/shayanfazeli/heartbeat
- Number of samples: 14552
- Number of channels: 1
- Number of classes: 2 (binary classification)
- Sampling frequency: 125 Hz
- Datapoints per sequence: 188

Configure TrainGenerator, DataGenerators, & model

[2]:
```
batch_size = 128
window_size = 188 / 4.  # use 4 windows
assert window_size.is_integer()  # ensure it divides sequence length
window_size = int(window_size)

# Make DataGenerator divide up the (128, 188, 1)-shaped batch
# into 4 slices shaped (128, 47, 1) each, feeding one at a time to model
C['datagen'    ]['preprocessor'] = 'timeseries'
C['val_datagen']['preprocessor'] = 'timeseries'
C['datagen'    ]['preprocessor_configs'] = {'window_size': window_size}
C['val_datagen']['preprocessor_configs'] = {'window_size': window_size}

C['model']['batch_shape'] = (batch_size, window_size, 1)
```

- `eval_fn`: need `'predict'` for visuals and custom metrics
- `key_metric`: `'f1_score'` for imbalanced binary classification
- `val_metrics`: true positive rate & true negative rate are "class accuracies", i.e. class-1 acc & class-2 acc
- `plot_first_pane_max_vals`: plot only validation loss in first plot window, the rest on second, to avoid clutter and keep losses together

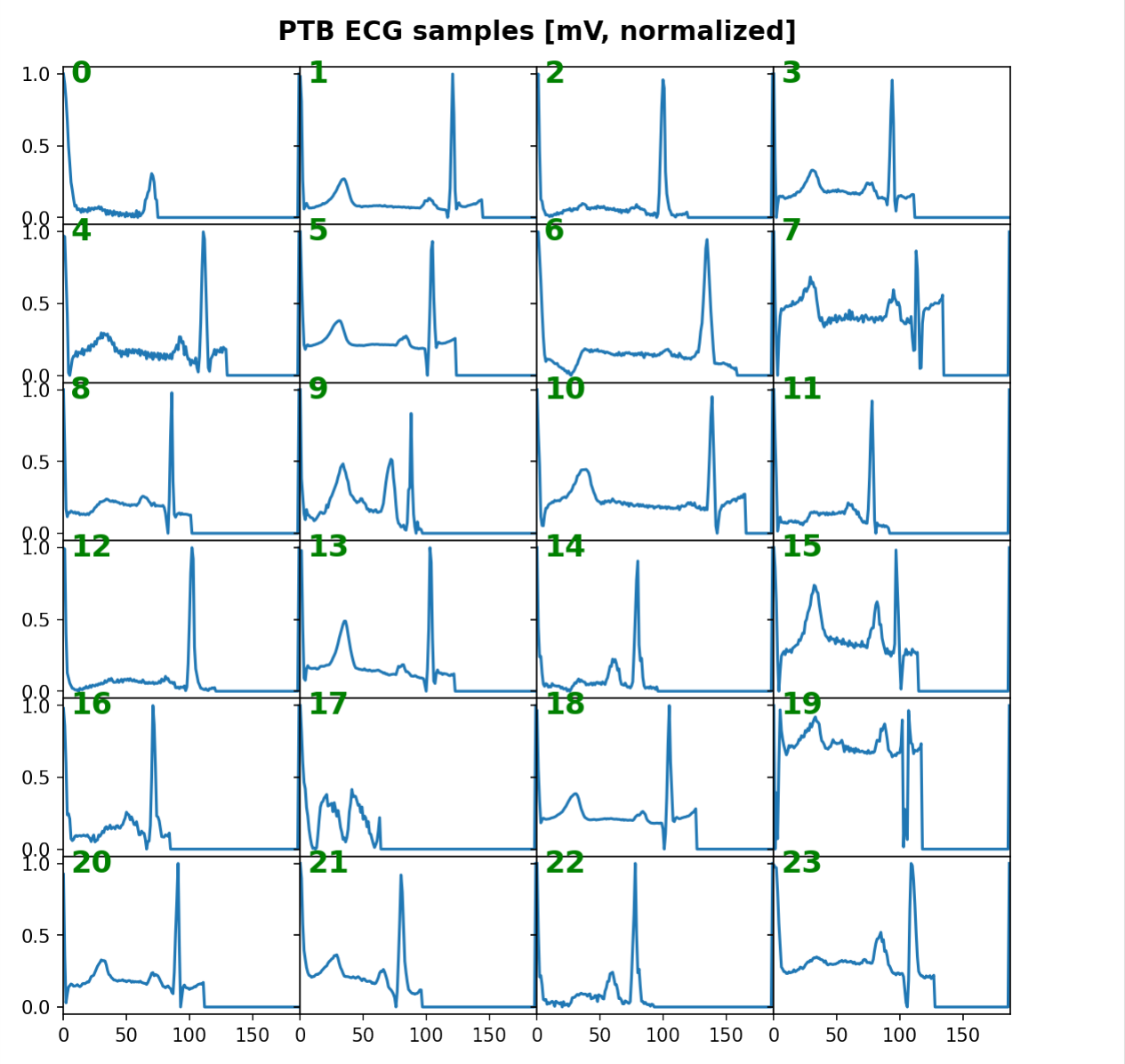
- `class_weights`: "normal" is the minority class; 3x more "abnormal" samples
- others: see utils.py

[3]:
```
C['traingen'].update(dict(
    eval_fn='predict',
    key_metric='f1_score',
    val_metrics=('loss', 'tnr', 'tpr'),
    plot_first_pane_max_vals=1,
    class_weights={0: 3, 1: 1},
    iter_verbosity=0,
    plot_configs={'fig_kw': {'figsize': (8, 5)},
                  '0': {'legend_kw': {'fontsize': 11}},
                  '1': {'legend_kw': {'fontsize': 11}}},
))
tg = init_session(C, make_timeseries_classifier)
```

```
Discovered dataset with matching format
Discovered dataset with matching format
103 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

Discovered dataset with matching format
Discovered dataset with matching format
12 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

Preloading superbatch ... Discovered dataset with matching format
... finished, w/ 13184 total samples
Train initial data prepared
Preloading superbatch ... Discovered dataset with matching format
... finished, w/ 1536 total samples
Val initial data prepared
Logging ON; directory (new): C:\deeptrain\examples\dir\logs\M2__model-adam__max.000
```

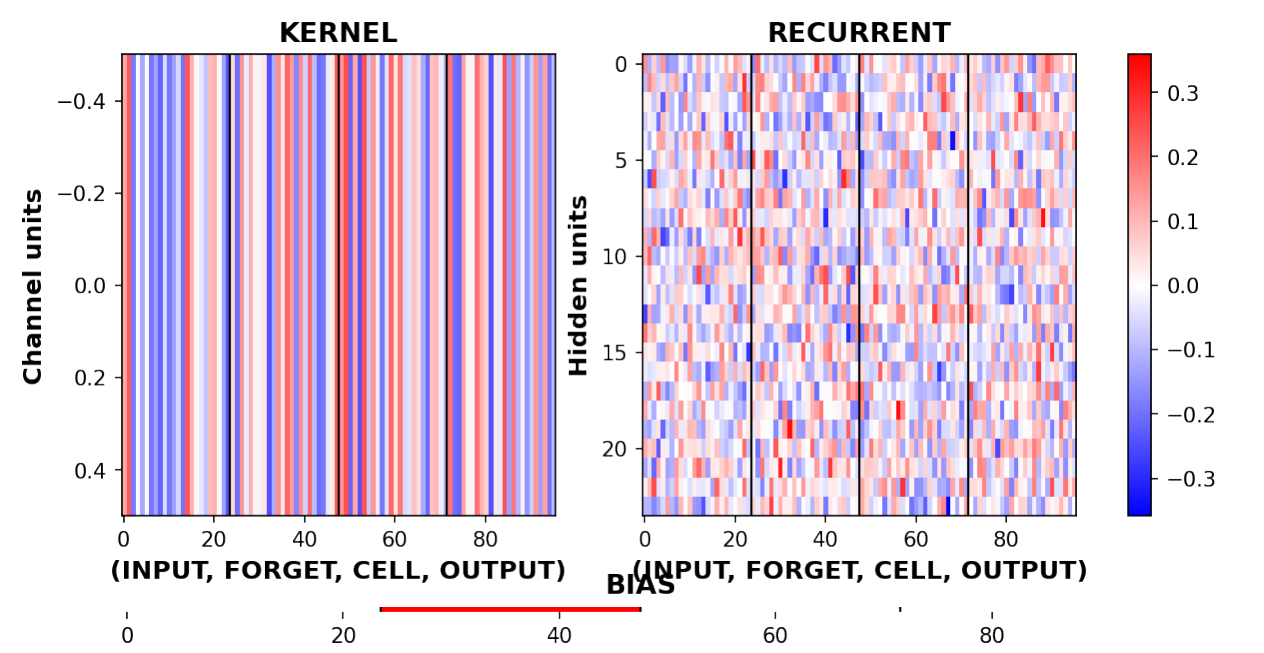
Visualize some samples

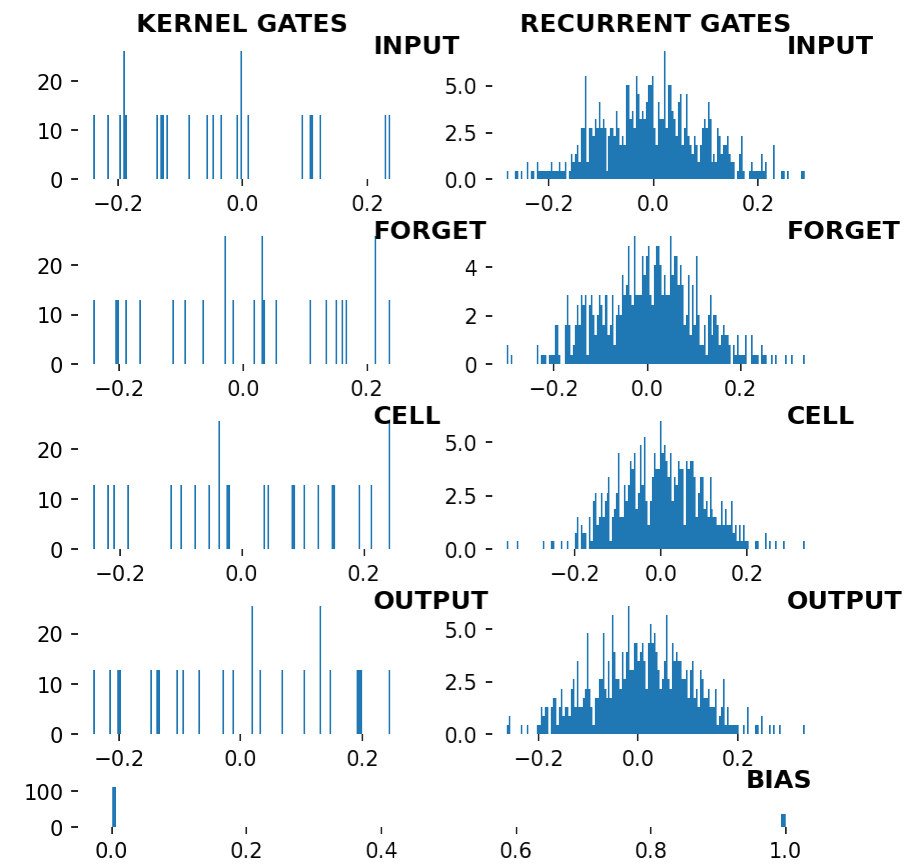
[4]:
```
data = tg.val_datagen.batch
_ = features_1D(data[:24], n_rows=6, subplot_samples=True, tight=True,
                title="PTB ECG samples [mV, normalized]")
```



Visualize LSTM weights before training

[5]:
```
_ = rnn_heatmap(  tg.model, 1, w=.9, h=.9)   # 1 == layer index
_ = rnn_histogram(tg.model, 1, w=.9, h=.9)
```

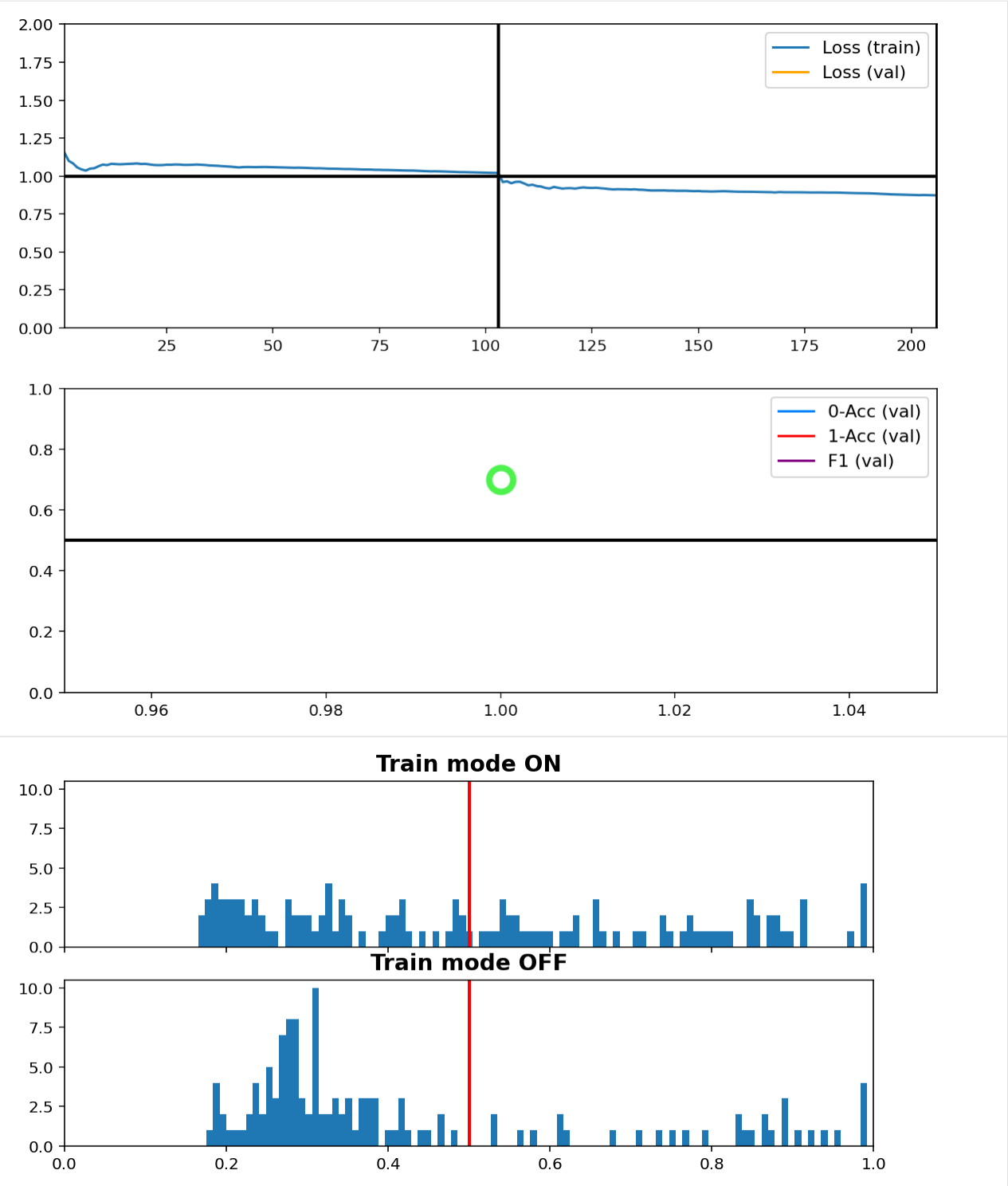

KERNEL
RECURRENT
Channel units
Hidden units
(INPUT, FORGET, CELL, OUTPUT)
(INPUT, FORGET, CELL, OUTPUT)
BIAS

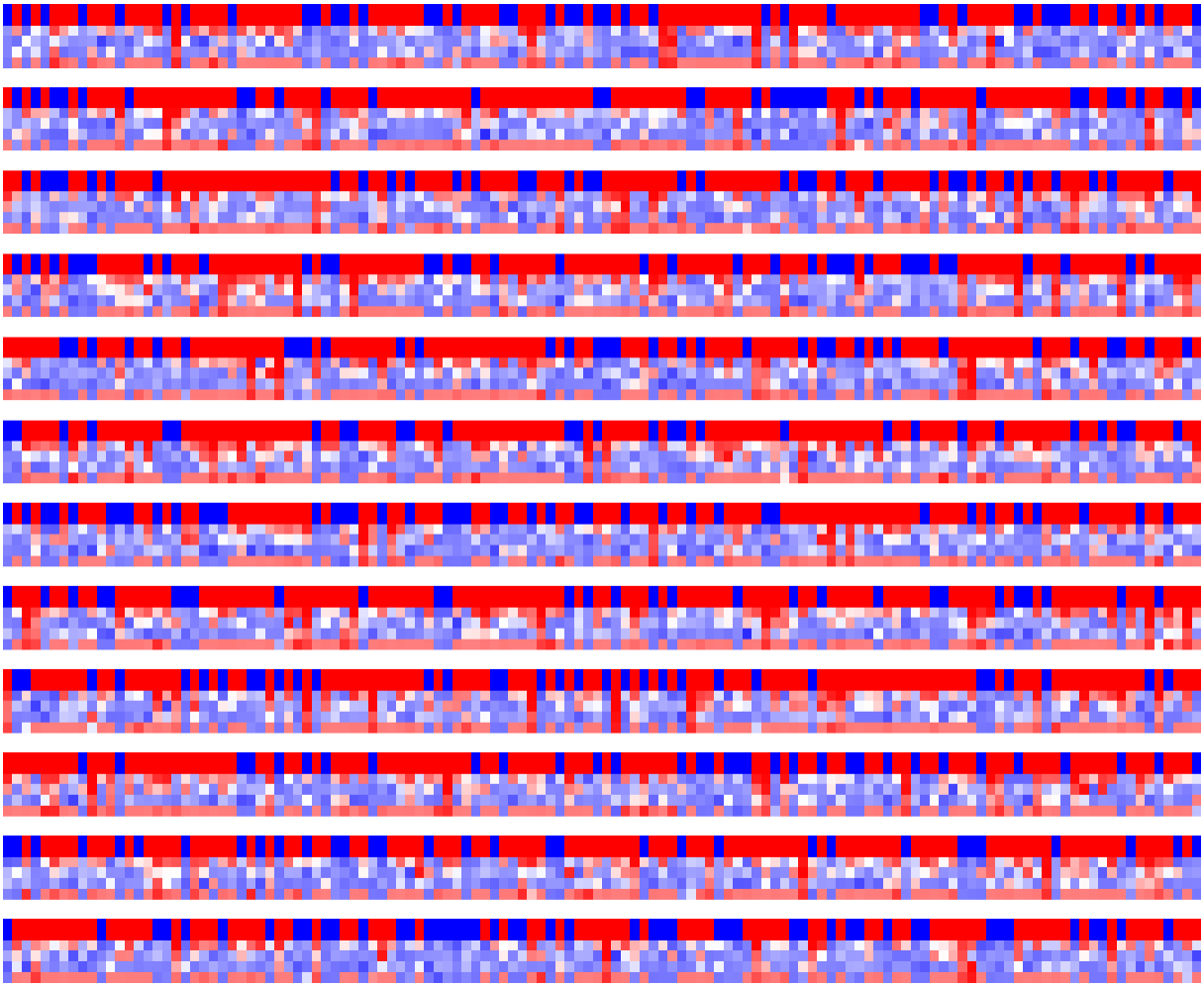

KERNEL GATES
RECURRENT GATES
INPUT
FORGET
CELL
OUTPUT
BIAS

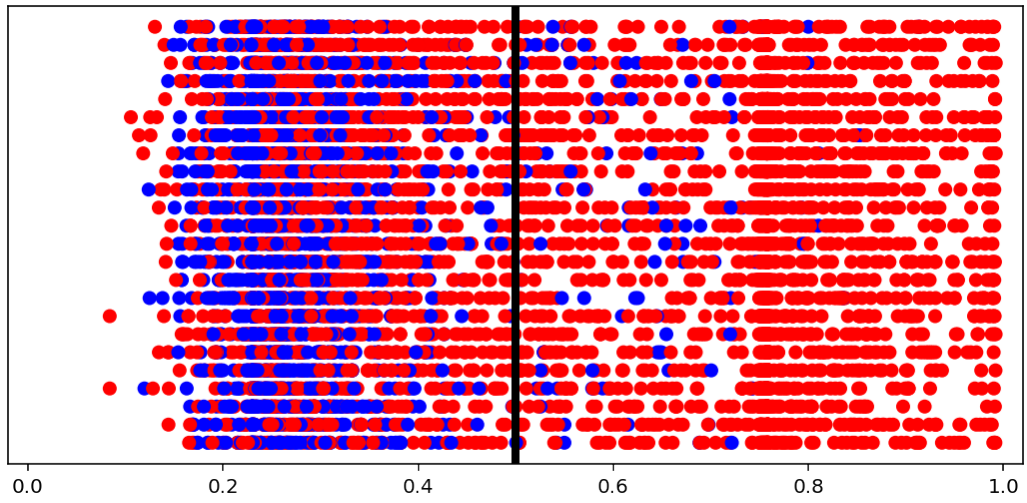
Train

[6]:
```
tg.train()
```

```
Data set_nums shuffled


_________________________
 EPOCH 1 -- COMPLETE
_________________________

Data set_nums shuffled


_________________________
 EPOCH 2 -- COMPLETE
_________________________



Validating...
TrainGenerator state saved
C:\deeptrain\deeptrain\visuals.py:577: UserWarning: Attempting to set identical left
↪== right == 1 results in singular transformations; automatically expanding.
  ax.set_xlim(xmin, xmax)
Model report generated and saved
Best model saved to C:\deeptrain\examples\dir\models\M2__model-adam__max.701
TrainGenerator state saved
Model report generated and saved
```


Loss (train)
Loss (val)
0-Acc (val)
1-Acc (val)
F1 (val)
Train mode ON
Train mode OFF



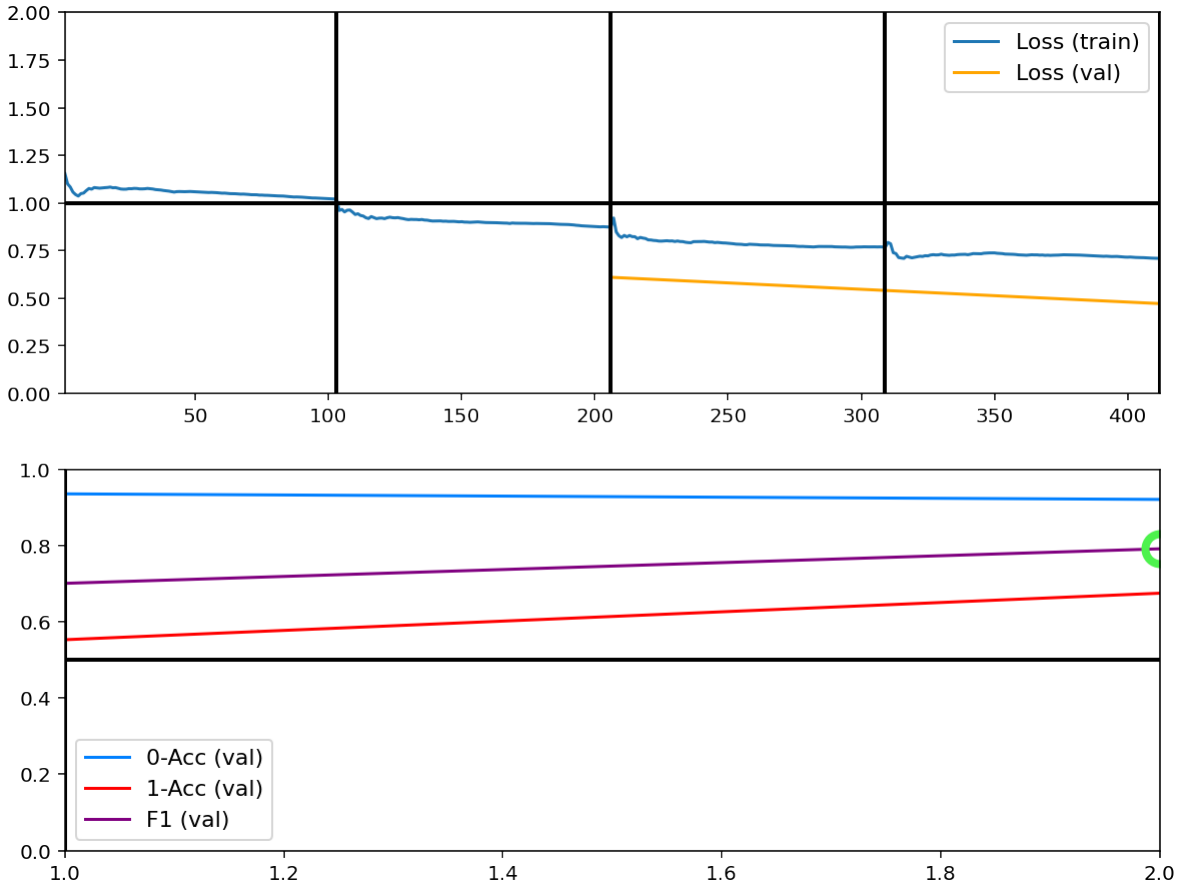


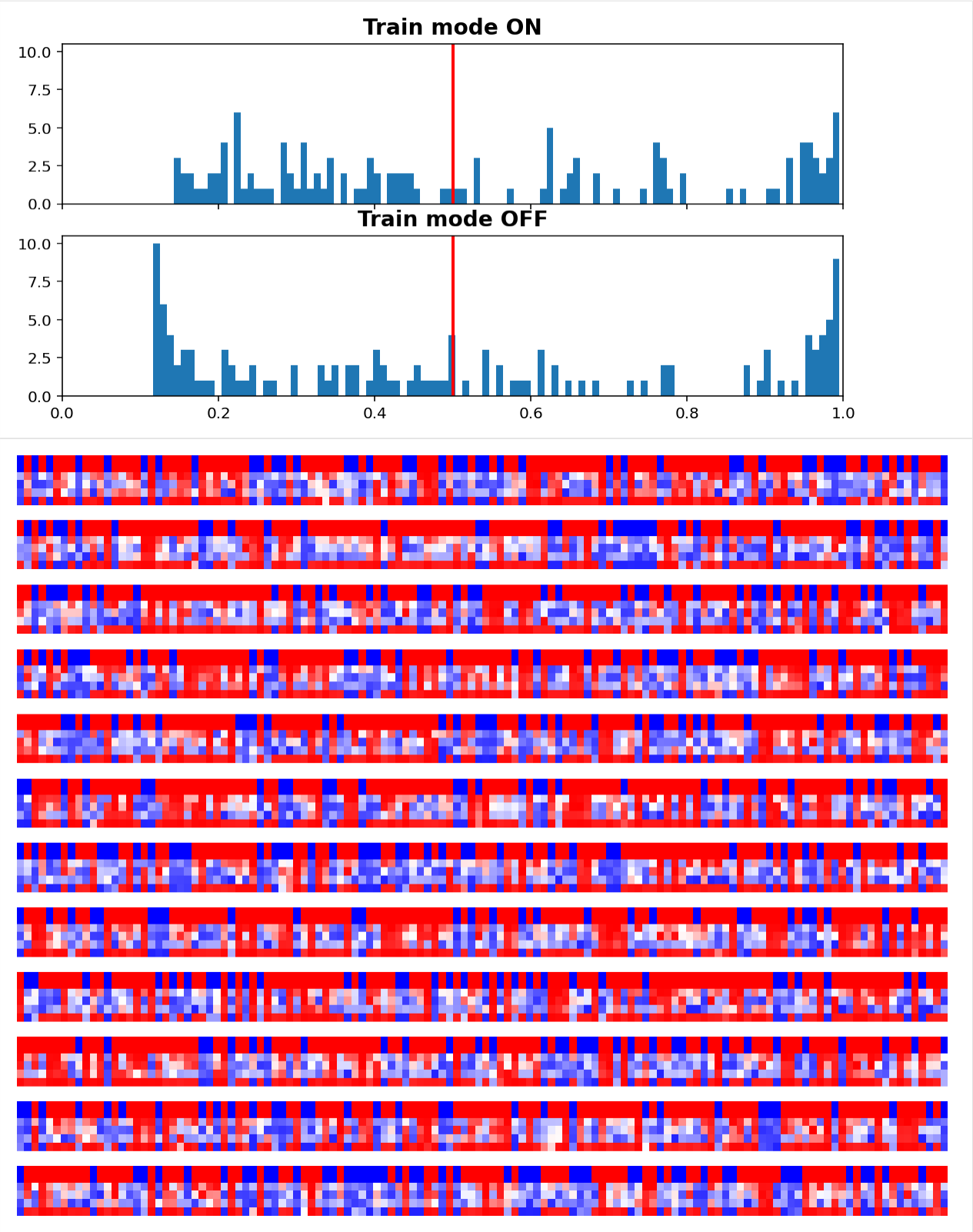

```
Data set_nums shuffled
```

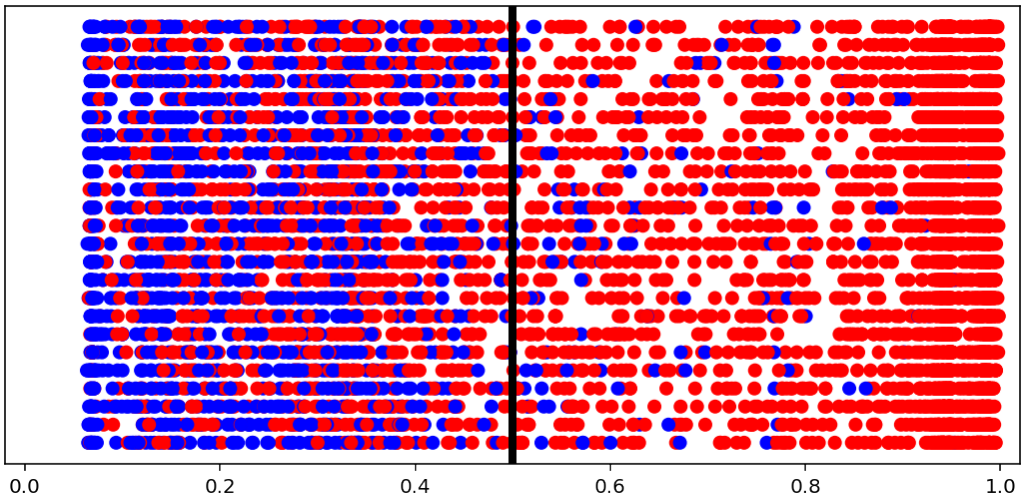
(continues on next page)

(continued from previous page)

```
_____________________
 EPOCH 3 -- COMPLETE

Data set_nums shuffled


_____________________
 EPOCH 4 -- COMPLETE


Validating...
TrainGenerator state saved
Model report generated and saved
Best model saved to C:\deeptrain\examples\dir\models\M2__model-adam__max.792
TrainGenerator state saved
Model report generated and saved
```



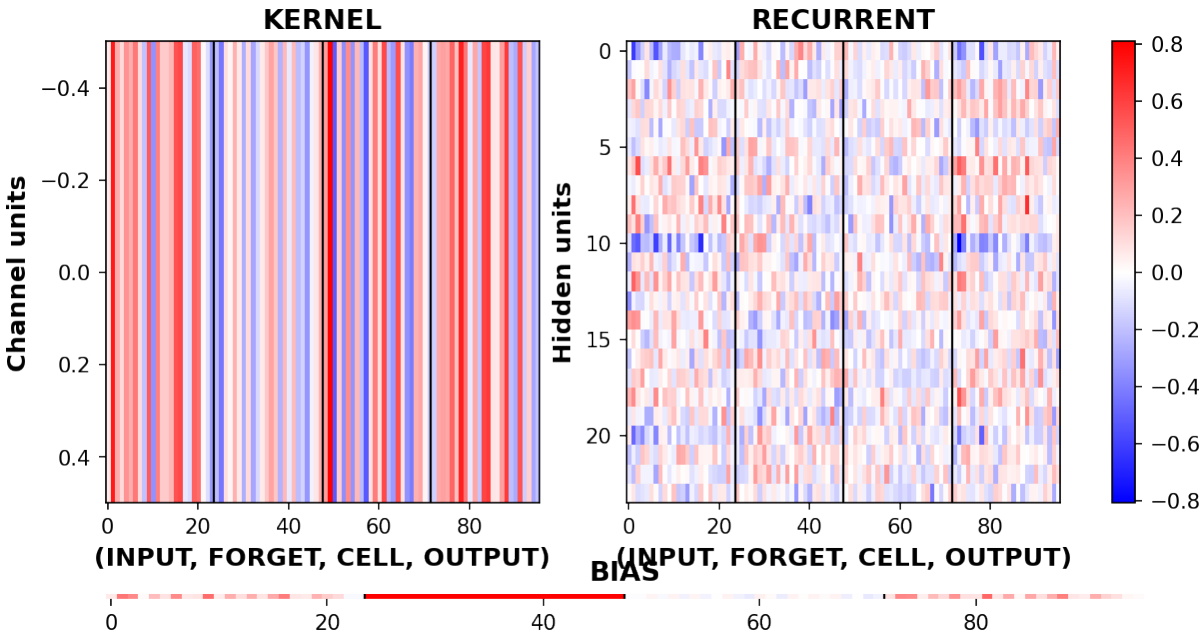

Train mode ON
10.0
7.5
5.0
2.5
0.0
Train mode OFF
10.0
7.5
5.0
2.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0

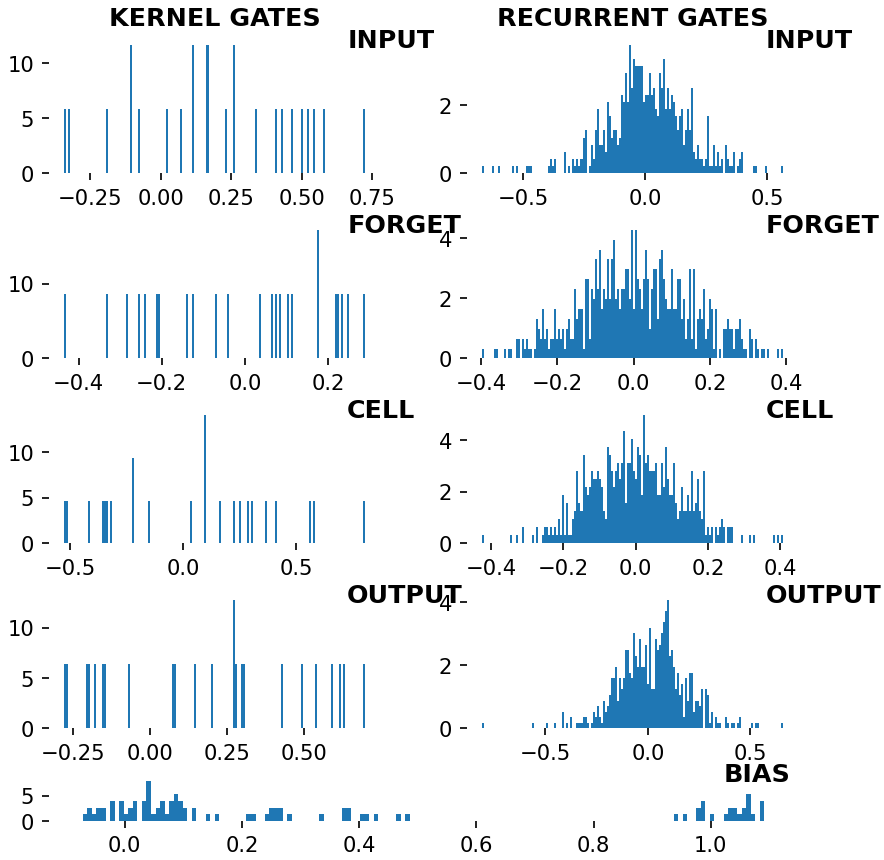



```
Training has concluded.
```

Visualize LSTM weights post-training

[7]:
```
_ = rnn_heatmap(  tg.model, 1, w=.9, h=.9)  # 1 == layer index
_ = rnn_histogram(tg.model, 1, w=.9, h=.9)
```





Differences are more pronounced when trained longer.

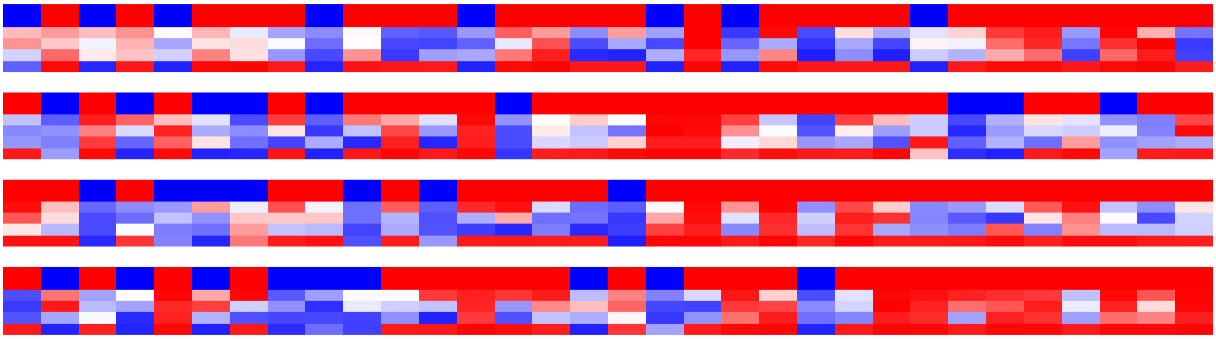
Next we inspect the callback figures; to redraw them, first we re-validate without clearing cache, to then get predictions & labels from cache for plotting.

[8]:
```
tg.validate(clear_cache=False, record_progress=False, use_callbacks=False)
```

```
Validating...
```

Predictions per iteration

[9]:
```
from deeptrain import visuals
import numpy as np

lc = np.asarray(tg._labels_cache)
pc = np.asarray(tg._preds_cache)
# select subset for clearer visual
visuals.binary_preds_per_iteration(lc[:4, :, :32], pc[:4, :, :32], h=.9)
```

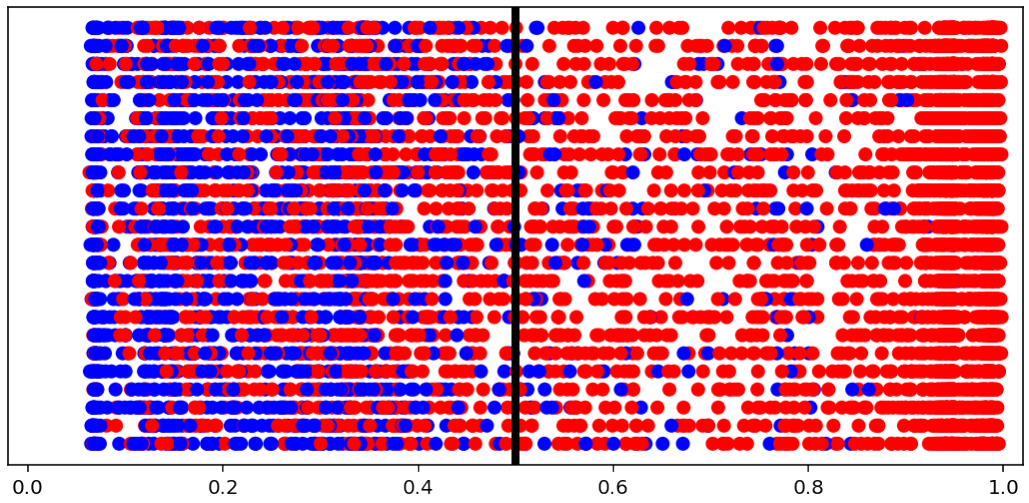


- Four "sets" for four batches
- For each set, columns = samples (32)
- Top of each set are labels, plotted with twice the thickness for clarity
- Below the labels are the predictions, heatmapped between blue (0) and red (1)
- Each row of predictions is a timeseries window ("slice")

Since the model is stateful, if predictions generaly color corretly toward bottom (later windows), it's indicative of LSTMs utilizing past windows. Here we see no such pattern, but training and tuning were limited.

Predictions distribution

[10]:
```
visuals.binary_preds_distribution(lc, pc, pred_th=tg.predict_threshold)
```

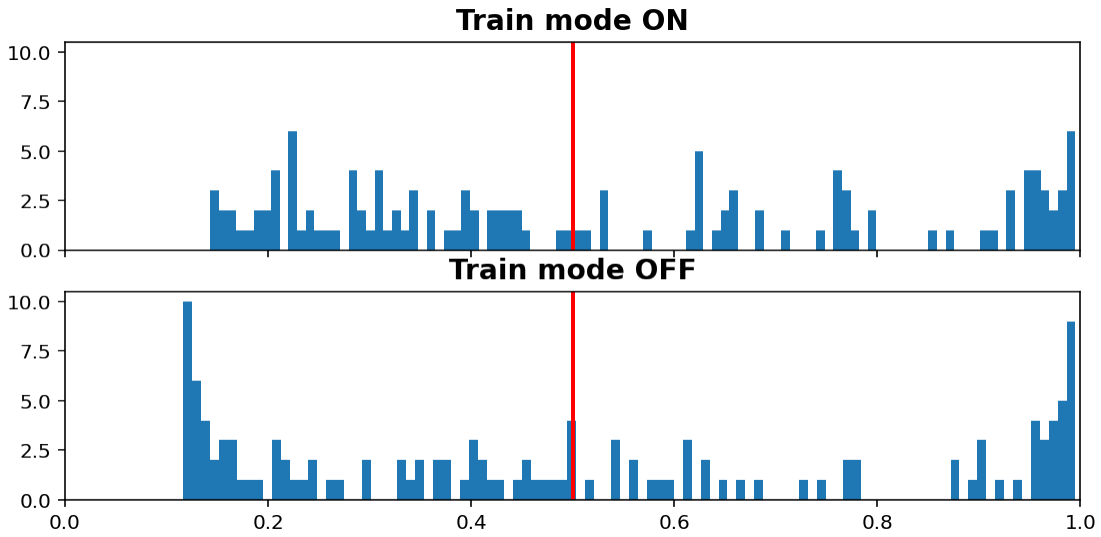


- Dots = predictions
- Red dot at 0.2 = predicted 0.2 for label 1 (red, right of the line = correct)
- Blue dot at 0.2 = predicted 0.2 for label 0 (blue, left of the line = correct)
- Vertical line = prediction threshold
- y-axis is meaningless, vertical space used only to avoid clutter

We see more red dots, as expected (0 is minority), and model being biased toward majority class. A better model will have less red on the left, and less blue on the right. The plot is a good indicator of model "calibration"; sometimes most dots will be either all the way left or all the way right, which may be undesired.

Inference vs. Train histogram

[11]:
```
visuals.infer_train_hist(tg.model, tg.val_datagen.get(skip_validation=True)[0],
                         vline=tg.predict_threshold, xlims=(0, 1))
```



- "Train mode" refers to the `learning_phase` flag
- When OFF, Dropout rate is 0, Batch Normalization uses moving average instead of batch statistics, and so on

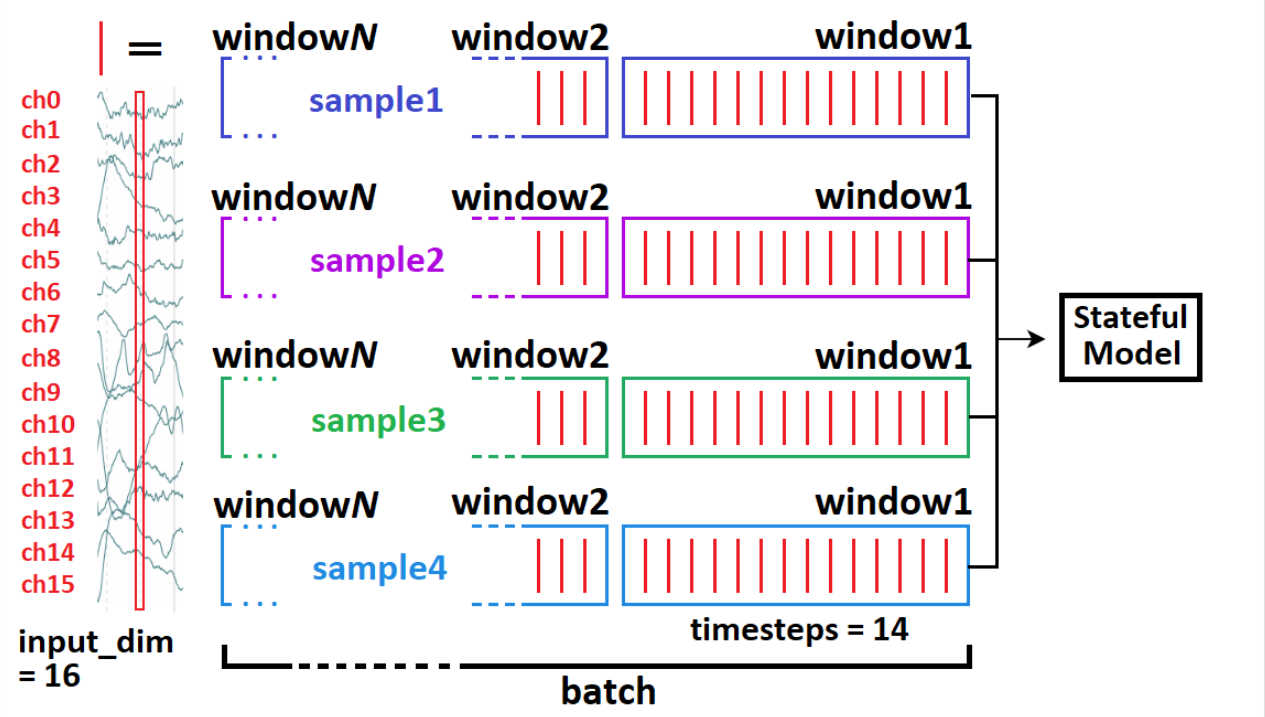
Increasing dropout rate will make ON vs OFF differences more dramatic. Note that the results are only for one validation window; using all would be expensive as we need to compute twice (train & inference), but is doable (see `introspection.py`). .....

Windowed timeseries idea

- Typical model input shape is `(batch_size, timesteps, input_dim)` (below = `(4, 14, 16)`)
- In stateful, we break up a whole batch into "windows".
- If complete sequence had 70 timesteps, then we'll feed 5 windows independently (but in right order), then call `model.reset_states()`
- Relevant: https://stackoverflow.com/a/58277760/10133797

[12]:
```
from IPython.display import Image
Image(filename='../../docs/source/_images/timeseries_windows.png', width=600)
```

[12]:



See Preprocessor example on how `TimeseriesPreprocessor` logic works.

3.5.2 Model Health Monitoring

This example assumes you've read `advanced.py`, and covers:

- Exploding & vanishing gradients monitoring
- Spotting dead weights

[1]:
```
import deeptrain
deeptrain.util.misc.append_examples_dir_to_sys_path()  # for `from utils import`

from utils import CL_CONFIGS as C
from utils import init_session, make_classifier
from utils import Adam
from see_rnn import rnn_histogram, rnn_heatmap
```

Case 1: Large weights

We train with a large learning rate to force large weights

[2]:
```
# We build a model prone to large but not exploding/vanishing gradients
C['model']['optimizer'] = Adam(6)
C['traingen']['epochs'] = 1
tg = init_session(C, make_classifier)
```

```
Discovered 48 files with matching format
Discovered dataset with matching format
48 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
```

(continues on next page)

(continued from previous page)

```
DataGenerator initiated

Discovered 36 files with matching format
Discovered dataset with matching format
36 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

Preloading superbatch ... Discovered 48 files with matching format
... finished, w/ 6144 total samples
Train initial data prepared
Preloading superbatch ... Discovered 36 files with matching format
... finished, w/ 4608 total samples
Val initial data prepared
Logging ON; directory (new): C:\deeptrain\examples\dir\logs\M2__model-Adam__min999.000
```

[3]:
```
tg.train()
```

```
Fitting set 1...   (Loss, Acc) = (2.301934, 0.164062)
Fitting set 2...   (Loss, Acc) = (61413017.150967, 0.105469)
Fitting set 3...   (Loss, Acc) = (50551152.767311, 0.101562)
Fitting set 4...   (Loss, Acc) = (39892719.075484, 0.113281)
Fitting set 5...   (Loss, Acc) = (31946056.266637, 0.115625)
Fitting set 6...   (Loss, Acc) = (26621893.911712, 0.117188)
Fitting set 7...   (Loss, Acc) = (22818785.382816, 0.119420)
Fitting set 8...   (Loss, Acc) = (19966440.713710, 0.116211)
Fitting set 9...   (Loss, Acc) = (17747949.665580, 0.114583)
Fitting set 10...  (Loss, Acc) = (15973156.238238, 0.112500)
Fitting set 11...  (Loss, Acc) = (14521056.922195, 0.113636)
Fitting set 12...  (Loss, Acc) = (13310969.448344, 0.112630)
Fitting set 13...  (Loss, Acc) = (12287049.257492, 0.109976)
Fitting set 14...  (Loss, Acc) = (11409403.477279, 0.105469)
Fitting set 15...  (Loss, Acc) = (10648777.200421, 0.105729)
Fitting set 16...  (Loss, Acc) = (9983229.190327, 0.105469)
Fitting set 17...  (Loss, Acc) = (9395981.031790, 0.105699)
Fitting set 18...  (Loss, Acc) = (8873982.566471, 0.105469)
Fitting set 19...  (Loss, Acc) = (8406931.181455, 0.105674)
Fitting set 20...  (Loss, Acc) = (7986584.963393, 0.105078)
Fitting set 21...  (Loss, Acc) = (7606271.703814, 0.104539)
Fitting set 22...  (Loss, Acc) = (7260532.407605, 0.104048)
Fitting set 23...  (Loss, Acc) = (6944857.373701, 0.101902)
Fitting set 24...  (Loss, Acc) = (6655488.531696, 0.101888)
Fitting set 25...  (Loss, Acc) = (6389269.171751, 0.101250)
Fitting set 26...  (Loss, Acc) = (6143528.244499, 0.100361)
Fitting set 27...  (Loss, Acc) = (5915990.377923, 0.100694)
Fitting set 28...  (Loss, Acc) = (5704705.207857, 0.101283)
Fitting set 29...  (Loss, Acc) = (5507991.401438, 0.101293)
Fitting set 30...  (Loss, Acc) = (5324391.880967, 0.101042)
Fitting set 31...  (Loss, Acc) = (5152637.451208, 0.100554)
Fitting set 32...  (Loss, Acc) = (4991617.706756, 0.100342)
Fitting set 33...  (Loss, Acc) = (4840356.694946, 0.099195)
Fitting set 34...  (Loss, Acc) = (4697993.408865, 0.097886)
Fitting set 35...  (Loss, Acc) = (4563765.141642, 0.098437)
Fitting set 36...  (Loss, Acc) = (4436993.987101, 0.098741)
Fitting set 37...  (Loss, Acc) = (4317075.328605, 0.097973)
Fitting set 38...  (Loss, Acc) = (4203468.178645, 0.098479)
```

(continues on next page)

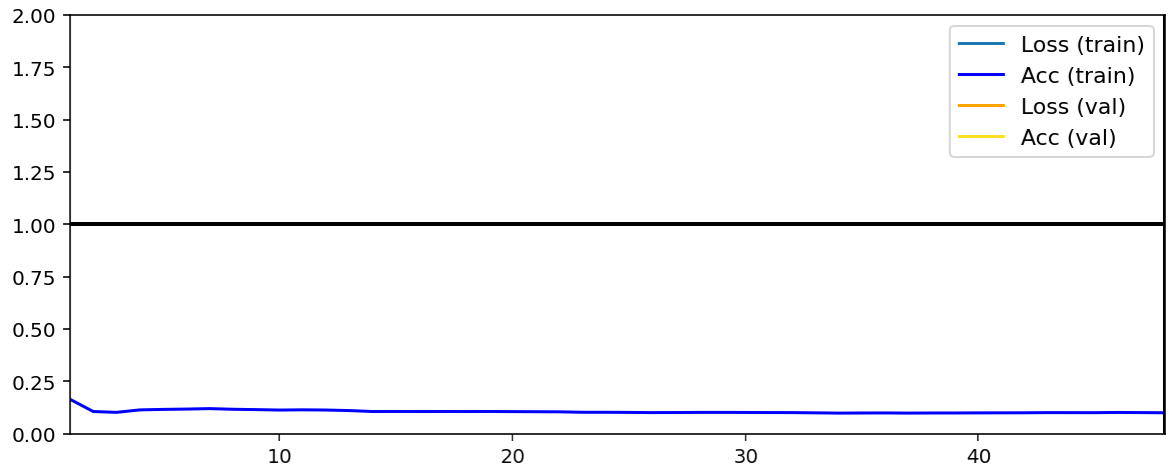
(continued from previous page)

```
Fitting set 39...  (Loss, Acc) = (4095687.036212, 0.098558)
Fitting set 40...  (Loss, Acc) = (3993294.942857, 0.099023)
Fitting set 41...  (Loss, Acc) = (3895897.572042, 0.099276)
Fitting set 42...  (Loss, Acc) = (3803138.170269, 0.099516)
Fitting set 43...  (Loss, Acc) = (3714693.161694, 0.100291)
Fitting set 44...  (Loss, Acc) = (3630268.393200, 0.100142)
Fitting set 45...  (Loss, Acc) = (3549595.830608, 0.100000)
Fitting set 46...  (Loss, Acc) = (3472430.768177, 0.101053)
Fitting set 47...  (Loss, Acc) = (3398549.320164, 0.100399)
Fitting set 48...  (Loss, Acc) = (3327746.265449, 0.099609)
Data set_nums shuffled



 EPOCH 1 -- COMPLETE



Validating...
Validating set 1...  (Loss, Acc) = (2.804169, 0.117188)
Validating set 2...  (Loss, Acc) = (2.710045, 0.109375)
Validating set 3...  (Loss, Acc) = (2.839425, 0.093750)
Validating set 4...  (Loss, Acc) = (2.752114, 0.093750)
Validating set 5...  (Loss, Acc) = (2.808827, 0.125000)
Validating set 6...  (Loss, Acc) = (2.799275, 0.046875)
Validating set 7...  (Loss, Acc) = (2.512745, 0.078125)
Validating set 8...  (Loss, Acc) = (2.578151, 0.109375)
Validating set 9...  (Loss, Acc) = (2.539990, 0.109375)
Validating set 10... (Loss, Acc) = (2.854088, 0.101562)
Validating set 11... (Loss, Acc) = (2.723956, 0.109375)
Validating set 12... (Loss, Acc) = (2.574893, 0.085938)
Validating set 13... (Loss, Acc) = (2.780102, 0.070312)
Validating set 14... (Loss, Acc) = (2.637559, 0.125000)
Validating set 15... (Loss, Acc) = (2.744786, 0.109375)
Validating set 16... (Loss, Acc) = (2.890051, 0.140625)
Validating set 17... (Loss, Acc) = (2.780263, 0.140625)
Validating set 18... (Loss, Acc) = (2.721423, 0.078125)
Validating set 19... (Loss, Acc) = (2.608651, 0.117188)
Validating set 20... (Loss, Acc) = (2.458480, 0.125000)
Validating set 21... (Loss, Acc) = (2.659738, 0.078125)
Validating set 22... (Loss, Acc) = (2.761351, 0.054688)
Validating set 23... (Loss, Acc) = (2.674613, 0.140625)
Validating set 24... (Loss, Acc) = (2.614646, 0.132812)
Validating set 25... (Loss, Acc) = (2.717063, 0.132812)
Validating set 26... (Loss, Acc) = (2.831150, 0.132812)
Validating set 27... (Loss, Acc) = (2.840220, 0.062500)
Validating set 28... (Loss, Acc) = (2.888196, 0.101562)
Validating set 29... (Loss, Acc) = (2.683285, 0.101562)
Validating set 30... (Loss, Acc) = (2.695441, 0.101562)
Validating set 31... (Loss, Acc) = (2.710979, 0.101562)
Validating set 32... (Loss, Acc) = (2.721793, 0.109375)
Validating set 33... (Loss, Acc) = (2.832050, 0.132812)
Validating set 34... (Loss, Acc) = (2.733249, 0.132812)
Validating set 35... (Loss, Acc) = (2.609143, 0.078125)
Validating set 36... (Loss, Acc) = (2.730237, 0.078125)
TrainGenerator state saved
Model report generated and saved
```

(continues on next page)

(continued from previous page)

```
Best model saved to C:\deeptrain\examples\dir\models\M2__model-Adam__min2.717
TrainGenerator state saved
Model report generated and saved
```




```
100.0% Large -- 'conv2d/kernel:0'
100.0% Large -- 'conv2d/bias:0'
100.0% Large -- 'conv2d_1/kernel:0'
100.0% Large -- 'conv2d_1/bias:0'
85.8% Large -- 'dense/kernel:0'
100.0% Large -- 'dense/bias:0'
99.7% Large -- 'dense_1/kernel:0'
100.0% Large -- 'dense_1/bias:0'
L = layer index, W = weight tensor indexTraining has concluded.
```

Case 2: Exploding/vanishing weights

We build RNNs with ReLU activations to generate extreme activations, thereby gradients and weights

[4]:
```
from utils import TS_CONFIGS as C
from utils import make_timeseries_classifier

C['model']['activation'] = 'relu'
C['model']['optimizer'] = Adam(.3)
C['traingen']['epochs'] = 1
C['traingen']['eval_fn'] = 'predict'
C['traingen']['val_freq'] = {'epoch': 1}
tg = init_session(C, make_timeseries_classifier)
```

```
WARNING:tensorflow:Layer lstm will not use cuDNN kernel since it doesn't meet the␣
↪cuDNN kernel criteria. It will use generic GPU kernel as fallback when running on␣
↪GPU
WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernel since it doesn't meet the␣
↪cuDNN kernel criteria. It will use generic GPU kernel as fallback when running on␣
↪GPU
Discovered dataset with matching format
Discovered dataset with matching format
103 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
```

(continues on next page)

(continued from previous page)

```
DataGenerator initiated

Discovered dataset with matching format
Discovered dataset with matching format
12 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

Preloading superbatch ... Discovered dataset with matching format
... finished, w/ 13184 total samples
Train initial data prepared
Preloading superbatch ... Discovered dataset with matching format
... finished, w/ 1536 total samples
Val initial data prepared
Logging ON; directory (new): C:\deeptrain\examples\dir\logs\M3__model-Adam__min999.000
```

[5]:
```
# will error due to being unable to plot nan metrics; we don't mind
try: tg.train()
except: pass
```

```
Fitting set 0...   Loss = nan RNNs reset
Fitting set 1...   Loss = nan RNNs reset
Fitting set 10...  Loss = nan RNNs reset
Fitting set 100... Loss = nan RNNs reset
Fitting set 101... Loss = nan RNNs reset
Fitting set 102... Loss = nan RNNs reset
Fitting set 11...  Loss = nan RNNs reset
Fitting set 12...  Loss = nan RNNs reset
Fitting set 13...  Loss = nan RNNs reset
Fitting set 14...  Loss = nan RNNs reset
Fitting set 15...  Loss = nan RNNs reset
Fitting set 16...  Loss = nan RNNs reset
Fitting set 17...  Loss = nan RNNs reset
Fitting set 18...  Loss = nan RNNs reset
Fitting set 19...  Loss = nan RNNs reset
Fitting set 2...   Loss = nan RNNs reset
Fitting set 20...  Loss = nan RNNs reset
Fitting set 21...  Loss = nan RNNs reset
Fitting set 22...  Loss = nan RNNs reset
Fitting set 23...  Loss = nan RNNs reset
Fitting set 24...  Loss = nan RNNs reset
Fitting set 25...  Loss = nan RNNs reset
Fitting set 26...  Loss = nan RNNs reset
Fitting set 27...  Loss = nan RNNs reset
Fitting set 28...  Loss = nan RNNs reset
Fitting set 29...  Loss = nan RNNs reset
Fitting set 3...   Loss = nan RNNs reset
Fitting set 30...  Loss = nan RNNs reset
Fitting set 31...  Loss = nan RNNs reset
Fitting set 32...  Loss = nan RNNs reset
Fitting set 33...  Loss = nan RNNs reset
Fitting set 34...  Loss = nan RNNs reset
Fitting set 35...  Loss = nan RNNs reset
Fitting set 36...  Loss = nan RNNs reset
Fitting set 37...  Loss = nan RNNs reset
Fitting set 38...  Loss = nan RNNs reset
```

(continues on next page)

(continued from previous page)

```
Fitting set 39...  Loss = nan RNNs reset
Fitting set 4...   Loss = nan RNNs reset
Fitting set 40...  Loss = nan RNNs reset
Fitting set 41...  Loss = nan RNNs reset
Fitting set 42...  Loss = nan RNNs reset
Fitting set 43...  Loss = nan RNNs reset
Fitting set 44...  Loss = nan RNNs reset
Fitting set 45...  Loss = nan RNNs reset
Fitting set 46...  Loss = nan RNNs reset
Fitting set 47...  Loss = nan RNNs reset
Fitting set 48...  Loss = nan RNNs reset
Fitting set 49...  Loss = nan RNNs reset
Fitting set 5...   Loss = nan RNNs reset
Fitting set 50...  Loss = nan RNNs reset
Fitting set 51...  Loss = nan RNNs reset
Fitting set 52...  Loss = nan RNNs reset
Fitting set 53...  Loss = nan RNNs reset
Fitting set 54...  Loss = nan RNNs reset
Fitting set 55...  Loss = nan RNNs reset
Fitting set 56...  Loss = nan RNNs reset
Fitting set 57...  Loss = nan RNNs reset
Fitting set 58...  Loss = nan RNNs reset
Fitting set 59...  Loss = nan RNNs reset
Fitting set 6...   Loss = nan RNNs reset
Fitting set 60...  Loss = nan RNNs reset
Fitting set 61...  Loss = nan RNNs reset
Fitting set 62...  Loss = nan RNNs reset
Fitting set 63...  Loss = nan RNNs reset
Fitting set 64...  Loss = nan RNNs reset
Fitting set 65...  Loss = nan RNNs reset
Fitting set 66...  Loss = nan RNNs reset
Fitting set 67...  Loss = nan RNNs reset
Fitting set 68...  Loss = nan RNNs reset
Fitting set 69...  Loss = nan RNNs reset
Fitting set 7...   Loss = nan RNNs reset
Fitting set 70...  Loss = nan RNNs reset
Fitting set 71...  Loss = nan RNNs reset
Fitting set 72...  Loss = nan RNNs reset
Fitting set 73...  Loss = nan RNNs reset
Fitting set 74...  Loss = nan RNNs reset
Fitting set 75...  Loss = nan RNNs reset
Fitting set 76...  Loss = nan RNNs reset
Fitting set 77...  Loss = nan RNNs reset
Fitting set 78...  Loss = nan RNNs reset
Fitting set 79...  Loss = nan RNNs reset
Fitting set 8...   Loss = nan RNNs reset
Fitting set 80...  Loss = nan RNNs reset
Fitting set 81...  Loss = nan RNNs reset
Fitting set 82...  Loss = nan RNNs reset
Fitting set 83...  Loss = nan RNNs reset
Fitting set 84...  Loss = nan RNNs reset
Fitting set 85...  Loss = nan RNNs reset
Fitting set 86...  Loss = nan RNNs reset
Fitting set 87...  Loss = nan RNNs reset
Fitting set 88...  Loss = nan RNNs reset
Fitting set 89...  Loss = nan RNNs reset
Fitting set 9...   Loss = nan RNNs reset
```

(continues on next page)

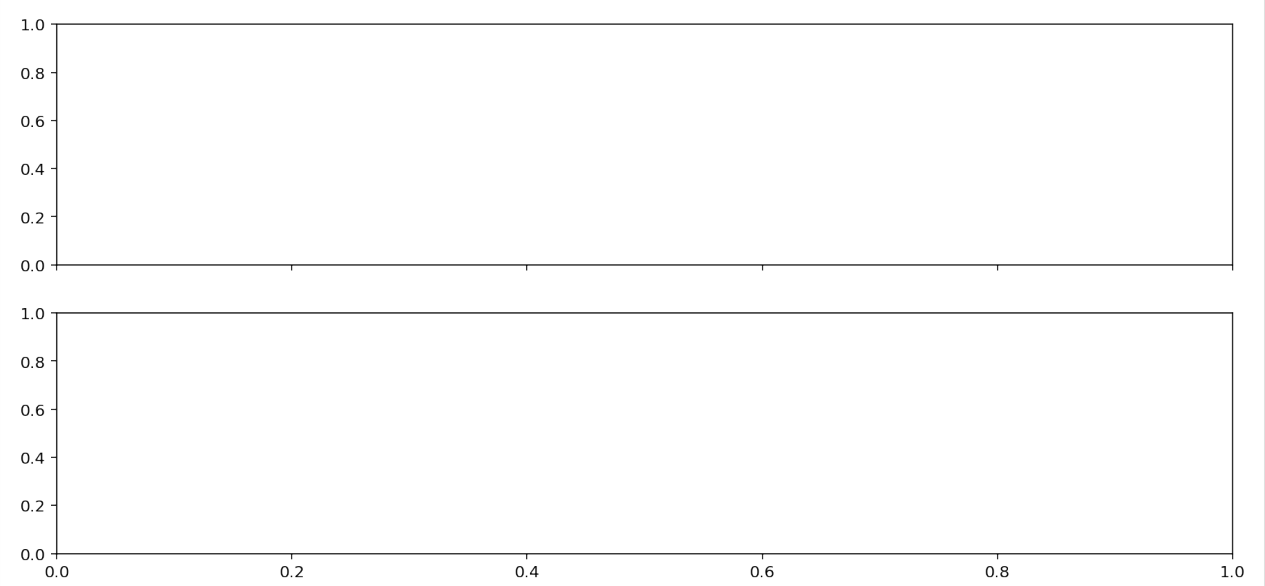
(continued from previous page)

```
Fitting set 90...  Loss = nan RNNs reset
Fitting set 91...  Loss = nan RNNs reset
Fitting set 92...  Loss = nan RNNs reset
Fitting set 93...  Loss = nan RNNs reset
Fitting set 94...  Loss = nan RNNs reset
Fitting set 95...  Loss = nan RNNs reset
Fitting set 96...  Loss = nan RNNs reset
Fitting set 97...  Loss = nan RNNs reset
Fitting set 98...  Loss = nan RNNs reset
Fitting set 99...  Loss = nan RNNs reset
Data set_nums shuffled



_____________________
 EPOCH 1 -- COMPLETE



Validating...
Validating set 0...  Loss = nan
RNNs reset Validating set 1...  Loss = nan
RNNs reset Validating set 10...
```

```
C:\deeptrain\deeptrain\metrics.py:113: RuntimeWarning: invalid value encountered in␣
↪greater_equal
  neg_abs_logits = np.where(logits >= 0, -logits, logits)
C:\deeptrain\deeptrain\metrics.py:114: RuntimeWarning: invalid value encountered in␣
↪greater_equal
  relu_logits    = np.where(logits >= 0, logits, 0)
C:\deeptrain\deeptrain\util\searching.py:70: RuntimeWarning: invalid value␣
↪encountered in greater
  new_best = (metric > best_metric if max_is_best else
```

```
 Loss = nan
RNNs reset Validating set 11... Loss = nan
RNNs reset Validating set 2...  Loss = nan
RNNs reset Validating set 3...  Loss = nan
RNNs reset Validating set 4...  Loss = nan
RNNs reset Validating set 5...  Loss = nan
RNNs reset Validating set 6...  Loss = nan
RNNs reset Validating set 7...  Loss = nan
RNNs reset Validating set 8...  Loss = nan
RNNs reset Validating set 9...  Loss = nan
RNNs reset
```

```
D:\Anaconda\envs\tf2_env\lib\site-packages\matplotlib\axes\_axes.py:6630:␣
↪RuntimeWarning: All-NaN slice encountered
  xmin = min(xmin, np.nanmin(xi))
D:\Anaconda\envs\tf2_env\lib\site-packages\matplotlib\axes\_axes.py:6630:␣
↪RuntimeWarning: invalid value encountered in less
  xmin = min(xmin, np.nanmin(xi))
D:\Anaconda\envs\tf2_env\lib\site-packages\matplotlib\axes\_axes.py:6631:␣
↪RuntimeWarning: All-NaN slice encountered
  xmax = max(xmax, np.nanmax(xi))
D:\Anaconda\envs\tf2_env\lib\site-packages\matplotlib\axes\_axes.py:6631:␣
↪RuntimeWarning: invalid value encountered in greater
  xmax = max(xmax, np.nanmax(xi))
```



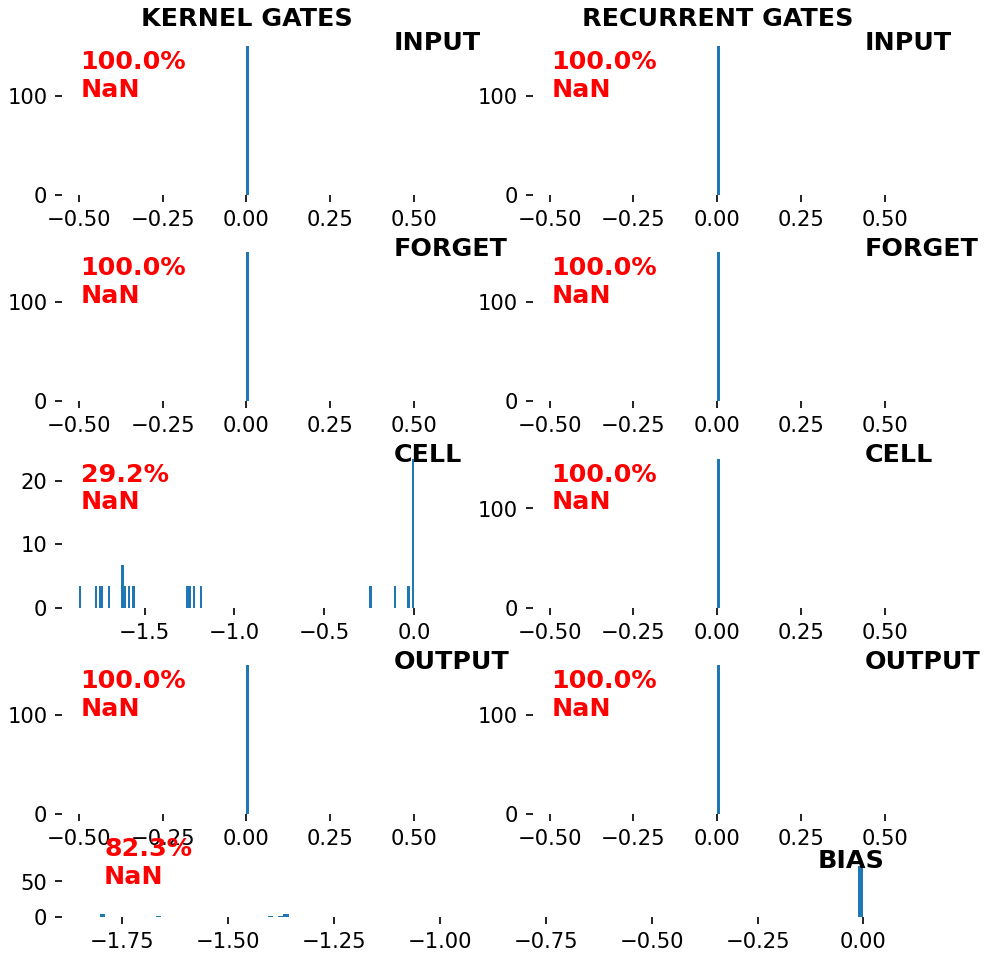
[6]:
```
tg.check_health()
```

```
3.125% dead -- 'lstm/lstm_cell/bias:0'
1.042% dead -- 'lstm_1/lstm_cell_1/bias:0'
L = layer index, W = weight tensor index

82.3% NaN -- 'lstm/lstm_cell/kernel:0'
100.0% NaN -- 'lstm/lstm_cell/recurrent_kernel:0'
82.3% NaN -- 'lstm/lstm_cell/bias:0'
100.0% NaN -- 'lstm_1/lstm_cell_1/kernel:0'
100.0% NaN -- 'lstm_1/lstm_cell_1/recurrent_kernel:0'
77.1% NaN -- 'lstm_1/lstm_cell_1/bias:0'
100.0% NaN -- 'dense_2/kernel:0'
100.0% NaN -- 'dense_2/bias:0'
L = layer index, W = weight tensor index
```

```
C:\deeptrain\deeptrain\introspection.py:405: RuntimeWarning: invalid value␣
↪encountered in less
  num_dead = np.sum(np.abs(w_value) < dead_threshold)
C:\deeptrain\deeptrain\introspection.py:490: RuntimeWarning: invalid value␣
↪encountered in greater
  num_large = np.sum(np.abs(w_value) > large_threshold) - num_nan
```

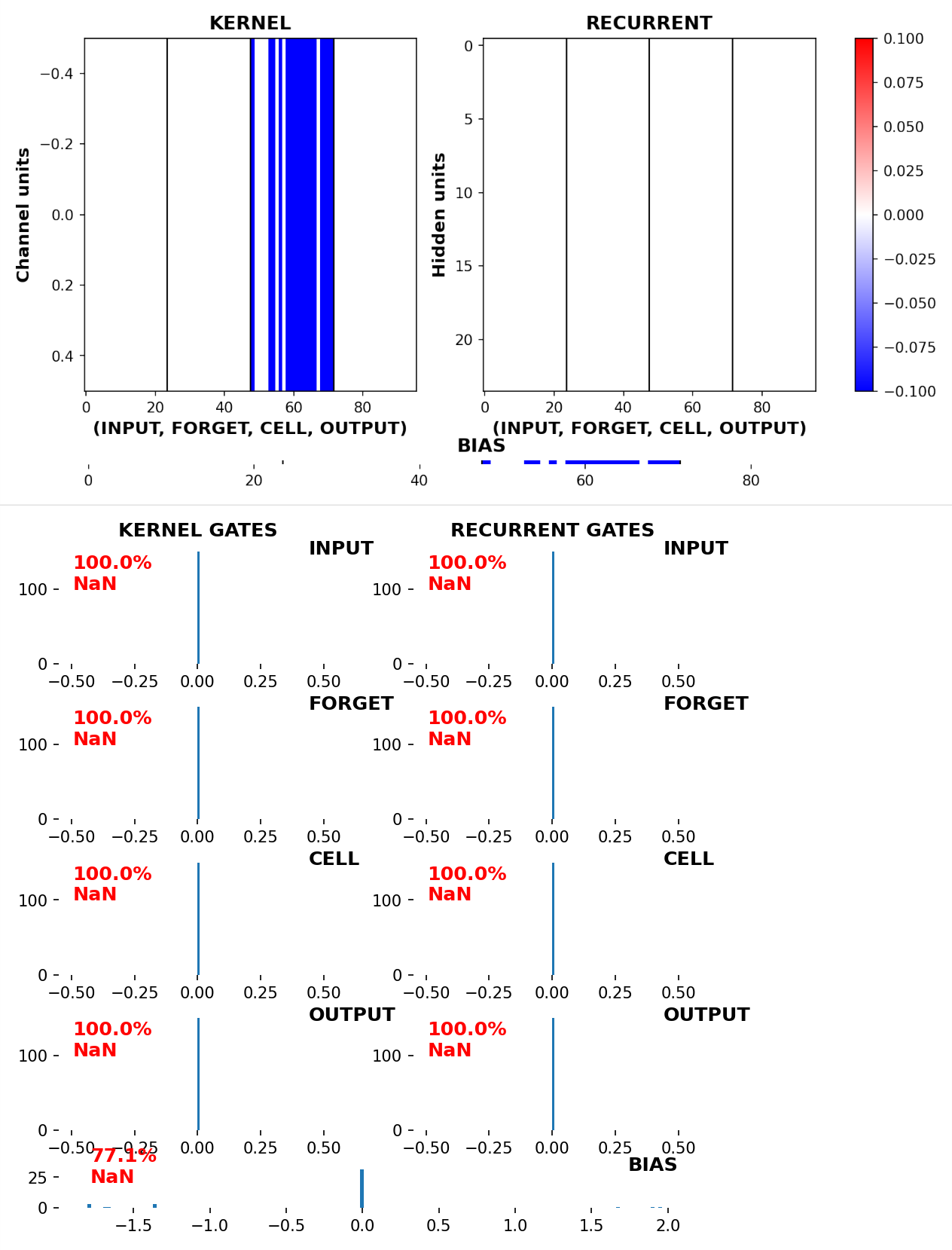
Visualize

[7]:
```
_ = rnn_histogram(tg.model, 1)
_ = rnn_heatmap(tg.model, 1)
_ = rnn_histogram(tg.model, 2)
_ = rnn_heatmap(tg.model, 2)
```

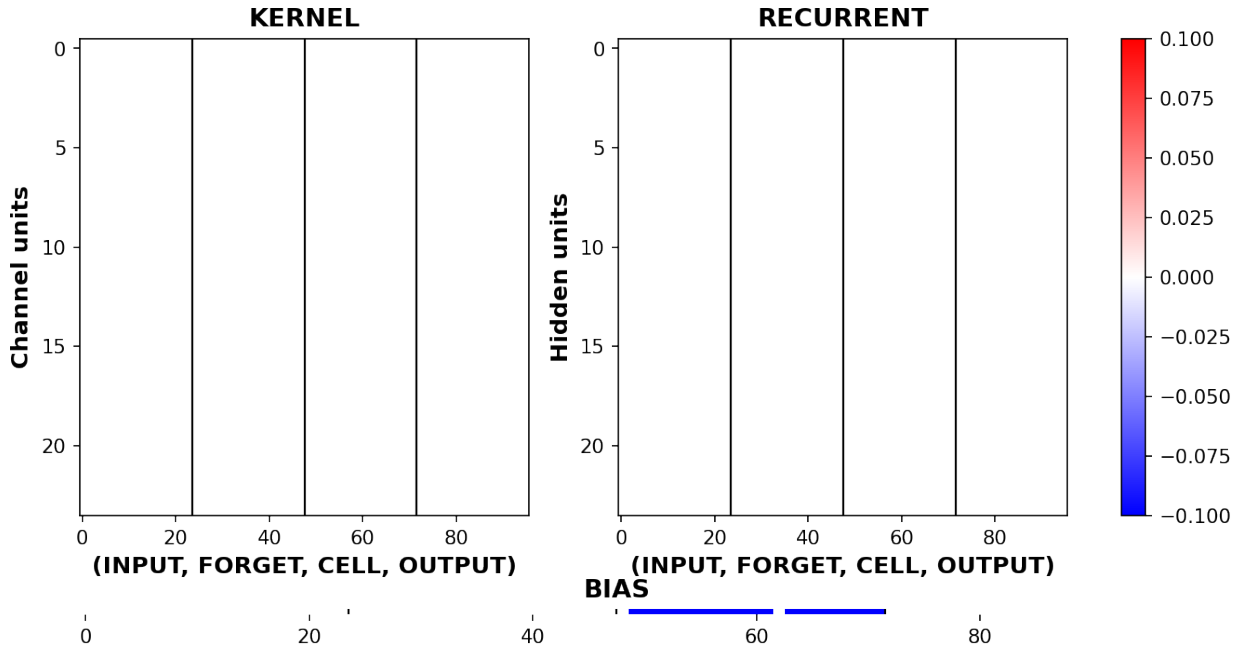



```
KERNEL:
INPUT: 100.0% NaN
FORGET: 100.0% NaN
CELL: 29.2% NaN
OUTPUT: 100.0% NaN

RECURRENT:
INPUT: 100.0% NaN
FORGET: 100.0% NaN
CELL: 100.0% NaN
OUTPUT: 100.0% NaN
```

```
D:\Anaconda\envs\tf2_env\lib\site-packages\matplotlib\colors.py:581: RuntimeWarning:␣
↪invalid value encountered in less
  xa[xa < 0] = -1
```


KERNEL
RECURRENT
Channel units
Hidden units
(INPUT, FORGET, CELL, OUTPUT)
(INPUT, FORGET, CELL, OUTPUT)
BIAS
KERNEL GATES
RECURRENT GATES
INPUT
FORGET
CELL
OUTPUT
BIAS
100.0%
NaN
77.1%
NaN

```
KERNEL:
INPUT: 100.0% NaN
FORGET: 100.0% NaN
CELL: 100.0% NaN
OUTPUT: 100.0% NaN

RECURRENT:
INPUT: 100.0% NaN
FORGET: 100.0% NaN
CELL: 100.0% NaN
OUTPUT: 100.0% NaN
```



3.5.3 Reproducibility

This example assumes you've read `callbacks/basic.ipynb`, and covers:

- Setting and restoring random seeds at arbitrary frequency for restoring from (nearly) any point in training

[1]:
```
import deeptrain
deeptrain.util.misc.append_examples_dir_to_sys_path()  # for `from utils import`

from utils import make_classifier, init_session
from utils import CL_CONFIGS as C
from deeptrain.callbacks import RandomSeedSetter
```

Random seed setter

Sets new random seeds (`random`, `numpy`, TF-graph, TF-global) every epoch, incrementing by 1 from start value (default 0).

- Since `tg.save()` is called each epoch, we specify `freq` via `'save'` instead of `'train:epoch'`.

- Setting `'load': 1` makes the setter retrieve the loaded seed values (upon `tg.load()`) and set seeds accordingly.

[2]:
```
seed_freq = {'save': 1, 'load': 1}
seed_setter = RandomSeedSetter(freq=seed_freq)
```

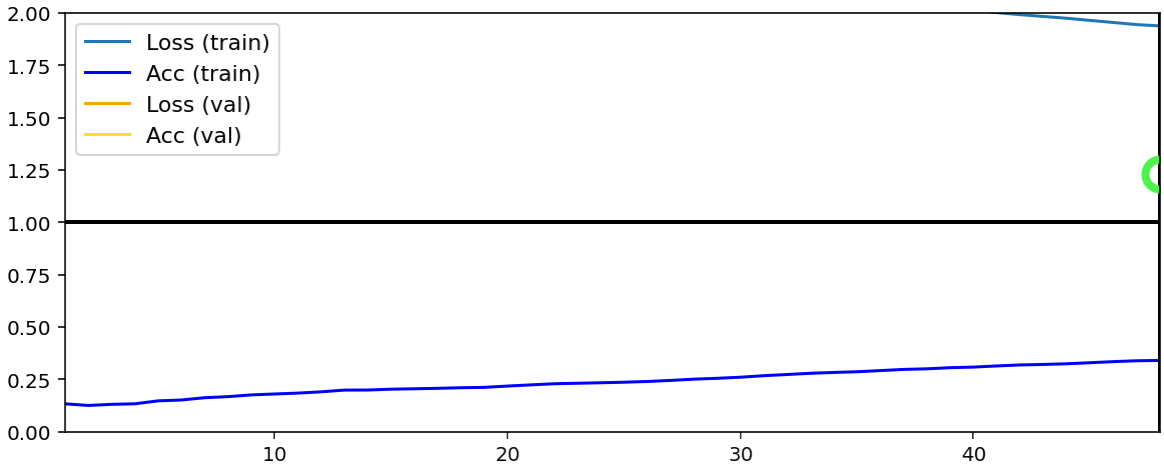
Configure & train

[3]:
```
C['traingen']['callbacks'] = [seed_setter]
C['traingen']['epochs'] = 3
C['traingen']['iter_verbosity'] = 0
tg = init_session(C, make_classifier)
```

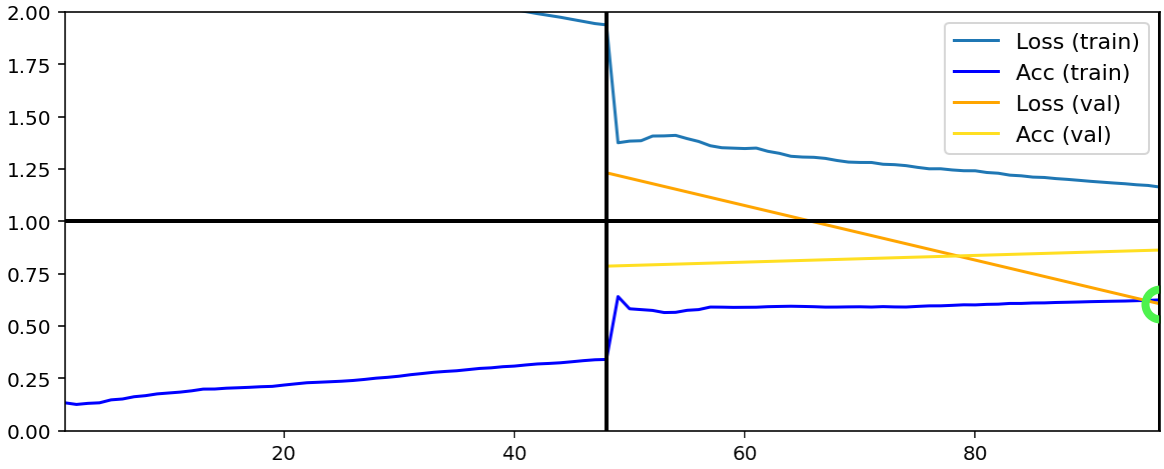
```
Discovered 48 files with matching format
Discovered dataset with matching format
48 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

Discovered 36 files with matching format
Discovered dataset with matching format
36 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

Preloading superbatch ... Discovered 48 files with matching format
... finished, w/ 6144 total samples
Train initial data prepared
Preloading superbatch ... Discovered 36 files with matching format
... finished, w/ 4608 total samples
Val initial data prepared
Logging ON; directory (new): C:\deeptrain\examples\dir\logs\M5__model-Adam__min999.000
```

[4]:
```
tg.train()
```

```
Data set_nums shuffled


_________________________
 EPOCH 1 -- COMPLETE


Validating...
RANDOM SEEDS RESET (random: 1, numpy: 1, tf-graph: 1, tf-global: 1)
TrainGenerator state saved
Model report generated and saved
Best model saved to C:\deeptrain\examples\dir\models\M5__model-Adam__min1.232
RANDOM SEEDS RESET (random: 2, numpy: 2, tf-graph: 2, tf-global: 2)
TrainGenerator state saved
Model report generated and saved
```




```
Data set_nums shuffled


 EPOCH 2 -- COMPLETE


Validating...
RANDOM SEEDS RESET (random: 3, numpy: 3, tf-graph: 3, tf-global: 3)
TrainGenerator state saved
Model report generated and saved
Best model saved to C:\deeptrain\examples\dir\models\M5__model-Adam__min.606
RANDOM SEEDS RESET (random: 4, numpy: 4, tf-graph: 4, tf-global: 4)
TrainGenerator state saved
Model report generated and saved
```

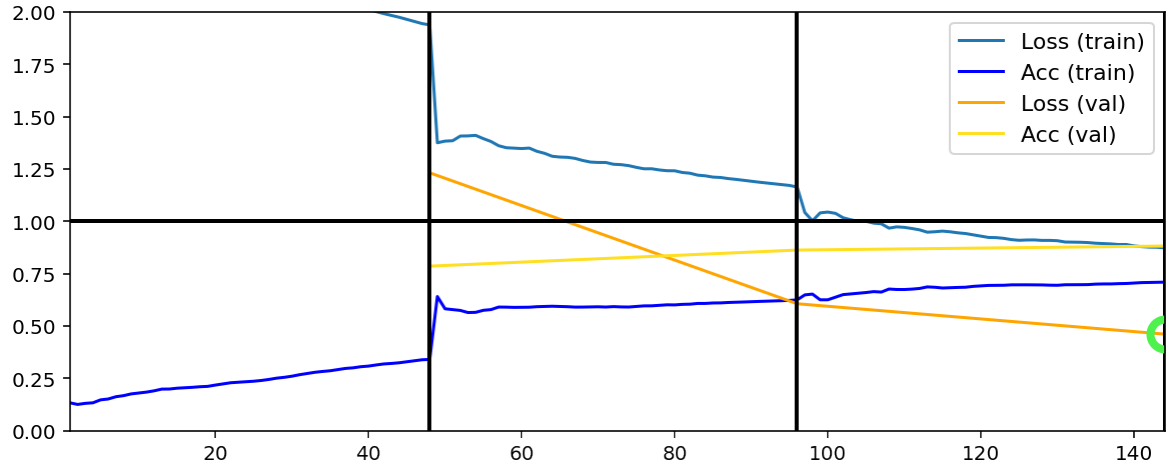



```
Data set_nums shuffled


```

(continues on next page)

(continued from previous page)

```
 EPOCH 3 -- COMPLETE


Validating...
RANDOM SEEDS RESET (random: 5, numpy: 5, tf-graph: 5, tf-global: 5)
TrainGenerator state saved
Model report generated and saved
Best model saved to C:\deeptrain\examples\dir\models\M5__model-Adam__min.461
RANDOM SEEDS RESET (random: 6, numpy: 6, tf-graph: 6, tf-global: 6)
TrainGenerator state saved
Model report generated and saved
```




```
Training has concluded.
```

- Text printed after epoch shows the values each of the four random seedwere set to, which by default start at 0 and increment by 1.
- Double incrementing is due to `tg.save()` being called within `.checkpoint()` *and* `._save_best_model()`.
- Note that TensorFlow lacks a global random state for later recovery (though it's possible to achieve with meticulous model & graph definition).
- Setting the seed at a point, and then loading the point and setting it again (which is what we'll do), however, works.

Clear current session

[5]:
```
# Retrieve last saved logfile to then load
loadpath = tg.get_last_log('state')
tg.destroy(confirm=True)
del tg, seed_setter  # seed_setter has internal reference to `tg`; destroy it
```

```
>>>TrainGenerator DESTROYED
```

Start new session, load savefile

[6]:
```
C['traingen']['loadpath'] = loadpath
C['traingen']['callbacks'] = [RandomSeedSetter(freq=seed_freq)]
tg = init_session(C, make_classifier)
```

```
Discovered 48 files with matching format
Discovered dataset with matching format
48 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

Discovered 36 files with matching format
Discovered dataset with matching format
36 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

TrainGenerator state loaded from C:\deeptrain\examples\dir\logs\M5__model-Adam__
↪min999.000\M5__model-Adam__min.461_3vals__state.h5
--Preloading excluded data based on datagen states ...
Preloading superbatch ... Discovered 48 files with matching format
... finished, w/ 6144 total samples
Train initial data prepared
Preloading superbatch ... Discovered 36 files with matching format
... finished, w/ 4608 total samples
Val initial data prepared
... finished--
RANDOM SEEDS RESET (random: 6, numpy: 6, tf-graph: 6, tf-global: 6)
Logging ON; directory (new): C:\deeptrain\examples\dir\logs\M6__model-Adam__min.461
```

Last random seed loaded and set; same would apply if we loaded from an earlier epoch.

3.5.4 Preprocessor & batching logic

This example assumes you've read `advanced.ipynb`, and covers:

- Creating custom `Preprocessor`
- How train & val loop and `DataGenerator` logic can be changed via `Preprocessor`

[1]:
```
import deeptrain
deeptrain.util.misc.append_examples_dir_to_sys_path()

from utils import make_autoencoder, init_session, AE_CONFIGS as C
from deeptrain.util.preprocessors import Preprocessor
import numpy as np
```

Preprocessor communicates with DataGenerator twofold: - .process() is called in DataGenerator.get() - DataGenerator sets and gets following attributes *through* Preprocessor: - `batch_exhausted`, `batch_loaded`, `slices_per_batch`, `slice_idx` - Thus, Preprocessor can dictate train & validation loop logic by specifying when a batch ends (setting `batch_exhausted`) in `.process()`, when some condition holds

RandCropPreprocessor

Below preprocessor randomly crops images to a predefined width & height, as an example of `.process()` in action. A better example of Preprocessor communicating with DataGenerator is the builtin `deeptrain.util.`

`preprocessors.TimeseriesPreprocessor`, demonstrated in `examples/misc/timeseries`.

[2]:
```
class RandCropPreprocessor(Preprocessor):
    """2D random crop. MNIST is 28x28, we try 25x25 crops,
    e.g. batch[2:27, 3:28]."""
    def __init__(self, size, crop_batch=True, crop_labels=False,
                 crop_same=False):
        # length          -> (length, length)
        # (width, height) -> (width, height)
        assert isinstance(size, (tuple, int))
        self.size = size if isinstance(size, tuple) else (size, size)

        self.crop_batch = crop_batch
        self.crop_labels = crop_labels
        self.crop_same = crop_same

    def process(self, batch, labels):
        if self.crop_batch:
            (x_start, x_end), (y_start, y_end) = self._make_crop_mask(batch)
            batch = batch[:, x_start:x_end, y_start:y_end]
        if self.crop_labels:
            if not self.crop_same or not self.crop_batch:
                (x_start, x_end), (y_start, y_end
                                   ) = self._make_crop_mask(labels)
            labels = labels[:, x_start:x_end, y_start:y_end]
        return batch, labels

    def _make_crop_mask(self, data):
        _, w, h, *_ = data.shape  # (samples, width, height, channels)
        x_offset = np.random.randint(0, w - self.size[0])
        y_offset = np.random.randint(0, h - self.size[1])
        x_start, x_end = x_offset, x_offset + self.size[0]
        y_start, y_end = y_offset, y_offset + self.size[1]
        return (x_start, x_end), (y_start, y_end)
```

[3]:
```
C['datagen'    ]['preprocessor'] = RandCropPreprocessor(size=24)
C['val_datagen']['preprocessor'] = RandCropPreprocessor(size=24)
C['datagen'    ]['batch_size'] = 128
C['val_datagen']['batch_size'] = 128
C['model']['batch_shape'] = (128, 24, 24, 1)
C['traingen']['iter_verbosity'] = 0
C['traingen']['epochs'] = 1
```

[4]:
```
tg = init_session(C, make_autoencoder)
```

```
WARNING: multiple file extensions found in `path`; only .npy will be used
Discovered 48 files with matching format
48 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

WARNING: multiple file extensions found in `path`; only .npy will be used
Discovered 36 files with matching format
36 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

NOTE: will exclude `labels` from saving when `input_as_labels=True`; to keep 'labels',
↪ add '{labels}'to `saveskip_list` instead
```

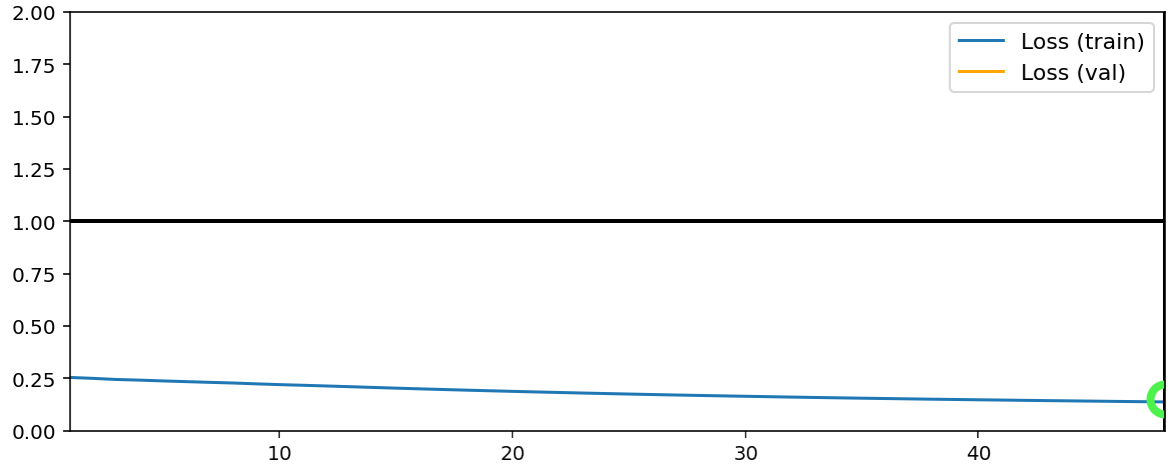
(continues on next page)

(continued from previous page)

```
Preloading superbatch ... WARNING: multiple file extensions found in `path`; only .
↪npy will be used
Discovered 48 files with matching format
... finished, w/ 6144 total samples
Train initial data prepared
Preloading superbatch ... WARNING: multiple file extensions found in `path`; only .
↪npy will be used
Discovered 36 files with matching format
... finished, w/ 4608 total samples
Val initial data prepared
Logging ON; directory (new): C:\deeptrain\examples\dir\logs\M4__model-nadam__min999.
↪000
```

[5]:
```
tg.train()
```

```
Data set_nums shuffled


_____________________
 EPOCH 1 -- COMPLETE


Validating...
TrainGenerator state saved
Model report generated and saved
Best model saved to C:\deeptrain\examples\dir\models\M4__model-nadam__min.153
TrainGenerator state saved
Model report generated and saved
```




```
Training has concluded.
```

TimeseriesPreprocessor

A better example of `Preprocessor` communicating with `DataGenerator` is the builtin `deeptrain.util.preprocessors.TimeseriesPreprocessor`, demonstrated in `examples/misc/timeseries`. Its main logic methods are worth inspecting.

`.process()` checks if we're at the first slice (window), and sets the window sequence length and number of windows per batch accordingly. This enables having variable windows per batch.

[6]:
```
from deeptrain.util.preprocessors import TimeseriesPreprocessor
from inspect import getsource
print(getsource(TimeseriesPreprocessor.process))
```

```
    def process(self, batch, labels):
        """Return next `batch` window, and unchanged `labels`."""
        if self.slice_idx == 0:
            # ensure number of windows accurate for every new batch
            self._batch_timesteps = batch.shape[1]
            self._set_slices_per_batch()
        return self._next_window(batch), labels
```

`._next_window()` fetches next window in the sequence according to `slice_idx`, `window_size`, and two other attrs (see docs)

[7]:
```
print(getsource(TimeseriesPreprocessor._next_window))
```

```
    def _next_window(self, batch):
        """Fetches temporal slice according to `window_size`, `slide_size`,
        `start_increment`, and `slice_idx`;
        See :class:`TimeseriesPreprocessor` for examples."""
        start = self.slice_idx * self.slide_size + self.start_increment
        end   = start + self.window_size
        return batch[:, start:end]
```

Lastly, it tells `DataGenerator` that batch ends when the last window was processed:

[8]:
```
print(getsource(TimeseriesPreprocessor.update_state))  # called within DataGenerator.
↪update_state
```

```
    def update_state(self):
        """Increment `slice_idx` by 1; if `slice_idx == slices_per_batch`,
        set `batch_exhausted = True`, `batch_loaded = False`.
        """
        self.slice_idx += 1
        if self.slice_idx == self.slices_per_batch:
            self.batch_exhausted = True
            self.batch_loaded = False
```

3.5.5 Flexible batch_size & Faster SSD Loading

This example assumes you've read `advanced.ipynb`, and covers:

- How batch_size can be a multiple of batch_size on file
- Faster SSD loading via flexible batch size

[1]:
```
import deeptrain
deeptrain.util.misc.append_examples_dir_to_sys_path()

from utils import make_autoencoder, init_session, AE_CONFIGS as C
```

DeepTrain can use `batch_size` an integral multiple of one on file, by splitting up into smaller batches or combining into larger.

If a file stores 128 samples, we can split it to x2 64-sample batches, or combine two files into x1 256-sample batch.

[2]:
```
C['traingen']['epochs'] = 1
```

User batch_size=64, file batch_size=128

[3]:
```
C['datagen'    ]['batch_size'] = 64
C['val_datagen']['batch_size'] = 64
C['model']['batch_shape'] = (64, 28, 28, 1)
```

[4]:
```
tg = init_session(C, make_autoencoder)
```

```
WARNING: multiple file extensions found in `path`; only .npy will be used
Discovered 48 files with matching format
48 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

WARNING: multiple file extensions found in `path`; only .npy will be used
Discovered 36 files with matching format
36 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

NOTE: will exclude `labels` from saving when `input_as_labels=True`; to keep 'labels',
↪ add '{labels}'to `saveskip_list` instead
Preloading superbatch ... WARNING: multiple file extensions found in `path`; only .
↪npy will be used
Discovered 48 files with matching format
... finished, w/ 6144 total samples
Train initial data prepared
Preloading superbatch ... WARNING: multiple file extensions found in `path`; only .
↪npy will be used
Discovered 36 files with matching format
... finished, w/ 4608 total samples
Val initial data prepared
Logging ON; directory (new): C:\deeptrain\examples\dir\logs\M3__model-nadam__min999.
↪000
```

[5]:
```
tg.train()
```

```
Fitting set 1-a... Loss = 0.258925
Fitting set 1-b... Loss = 0.253132
Fitting set 2-a... Loss = 0.248714
Fitting set 2-b... Loss = 0.244794
Fitting set 3-a... Loss = 0.240528
Fitting set 3-b... Loss = 0.236792
Fitting set 4-a... Loss = 0.233093
Fitting set 4-b... Loss = 0.229193
Fitting set 5-a... Loss = 0.225168
Fitting set 5-b... Loss = 0.221404
Fitting set 6-a... Loss = 0.217777
Fitting set 6-b... Loss = 0.214355
```

(continues on next page)

(continued from previous page)

```
Fitting set 7-a... Loss = 0.210822
Fitting set 7-b... Loss = 0.207650
Fitting set 8-a... Loss = 0.204442
Fitting set 8-b... Loss = 0.201394
Fitting set 9-a... Loss = 0.198212
Fitting set 9-b... Loss = 0.195283
Fitting set 10-a... Loss = 0.192382
Fitting set 10-b... Loss = 0.189485
Fitting set 11-a... Loss = 0.186594
Fitting set 11-b... Loss = 0.183790
Fitting set 12-a... Loss = 0.181016
Fitting set 12-b... Loss = 0.178265
Fitting set 13-a... Loss = 0.175732
Fitting set 13-b... Loss = 0.173159
Fitting set 14-a... Loss = 0.170662
Fitting set 14-b... Loss = 0.168275
Fitting set 15-a... Loss = 0.165887
Fitting set 15-b... Loss = 0.163576
Fitting set 16-a... Loss = 0.161341
Fitting set 16-b... Loss = 0.159190
Fitting set 17-a... Loss = 0.157046
Fitting set 17-b... Loss = 0.155032
Fitting set 18-a... Loss = 0.153119
Fitting set 18-b... Loss = 0.151234
Fitting set 19-a... Loss = 0.149326
Fitting set 19-b... Loss = 0.147504
Fitting set 20-a... Loss = 0.145753
Fitting set 20-b... Loss = 0.144026
Fitting set 21-a... Loss = 0.142380
Fitting set 21-b... Loss = 0.140762
Fitting set 22-a... Loss = 0.139213
Fitting set 22-b... Loss = 0.137701
Fitting set 23-a... Loss = 0.136231
Fitting set 23-b... Loss = 0.134775
Fitting set 24-a... Loss = 0.133367
Fitting set 24-b... Loss = 0.131993
Fitting set 25-a... Loss = 0.130666
Fitting set 25-b... Loss = 0.129385
Fitting set 26-a... Loss = 0.128131
Fitting set 26-b... Loss = 0.126897
Fitting set 27-a... Loss = 0.125704
Fitting set 27-b... Loss = 0.124558
Fitting set 28-a... Loss = 0.123391
Fitting set 28-b... Loss = 0.122320
Fitting set 29-a... Loss = 0.121234
Fitting set 29-b... Loss = 0.120158
Fitting set 30-a... Loss = 0.119135
Fitting set 30-b... Loss = 0.118139
Fitting set 31-a... Loss = 0.117173
Fitting set 31-b... Loss = 0.116213
Fitting set 32-a... Loss = 0.115311
Fitting set 32-b... Loss = 0.114430
Fitting set 33-a... Loss = 0.113554
Fitting set 33-b... Loss = 0.112672
Fitting set 34-a... Loss = 0.111832
Fitting set 34-b... Loss = 0.110993
Fitting set 35-a... Loss = 0.110203
```

(continues on next page)

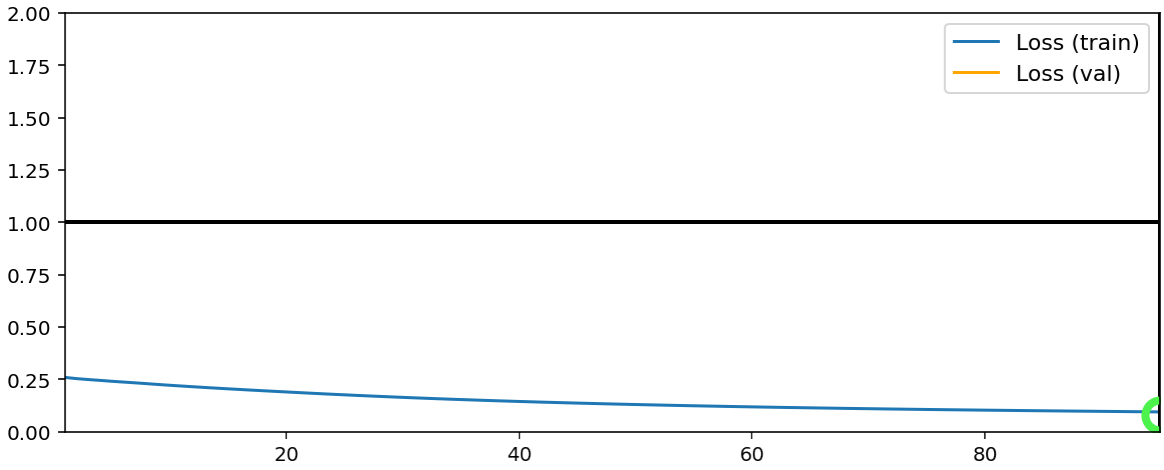
(continued from previous page)

```
Fitting set 35-b... Loss = 0.109420
Fitting set 36-a... Loss = 0.108674
Fitting set 36-b... Loss = 0.107918
Fitting set 37-a... Loss = 0.107198
Fitting set 37-b... Loss = 0.106458
Fitting set 38-a... Loss = 0.105749
Fitting set 38-b... Loss = 0.105067
Fitting set 39-a... Loss = 0.104410
Fitting set 39-b... Loss = 0.103764
Fitting set 40-a... Loss = 0.103141
Fitting set 40-b... Loss = 0.102499
Fitting set 41-a... Loss = 0.101859
Fitting set 41-b... Loss = 0.101279
Fitting set 42-a... Loss = 0.100686
Fitting set 42-b... Loss = 0.100075
Fitting set 43-a... Loss = 0.099509
Fitting set 43-b... Loss = 0.098956
Fitting set 44-a... Loss = 0.098396
Fitting set 44-b... Loss = 0.097852
Fitting set 45-a... Loss = 0.097302
Fitting set 45-b... Loss = 0.096763
Fitting set 46-a... Loss = 0.096223
Fitting set 46-b... Loss = 0.095674
Fitting set 47-a... Loss = 0.095153
Fitting set 47-b... Loss = 0.094651
Fitting set 48-a... Loss = 0.094162
Data set_nums shuffled


_________________________
 EPOCH 1 -- COMPLETE


Validating...
Validating set 1-a... Loss = 0.078783
Validating set 1-b... Loss = 0.076317
Validating set 2-a... Loss = 0.075710
Validating set 2-b... Loss = 0.079148
Validating set 3-a... Loss = 0.081881
Validating set 3-b... Loss = 0.080691
Validating set 4-a... Loss = 0.077261
Validating set 4-b... Loss = 0.081151
Validating set 5-a... Loss = 0.079396
Validating set 5-b... Loss = 0.077392
Validating set 6-a... Loss = 0.078506
Validating set 6-b... Loss = 0.080747
Validating set 7-a... Loss = 0.078852
Validating set 7-b... Loss = 0.076874
Validating set 8-a... Loss = 0.079984
Validating set 8-b... Loss = 0.077159
Validating set 9-a... Loss = 0.077525
Validating set 9-b... Loss = 0.076241
Validating set 10-a... Loss = 0.078210
Validating set 10-b... Loss = 0.078955
Validating set 11-a... Loss = 0.080569
Validating set 11-b... Loss = 0.077822
```

(continues on next page)

(continued from previous page)

```
Validating set 12-a... Loss = 0.079494
Validating set 12-b... Loss = 0.075832
Validating set 13-a... Loss = 0.079898
Validating set 13-b... Loss = 0.083832
Validating set 14-a... Loss = 0.077217
Validating set 14-b... Loss = 0.075610
Validating set 15-a... Loss = 0.082518
Validating set 15-b... Loss = 0.077477
Validating set 16-a... Loss = 0.081708
Validating set 16-b... Loss = 0.081968
Validating set 17-a... Loss = 0.080029
Validating set 17-b... Loss = 0.076852
Validating set 18-a... Loss = 0.076589
Validating set 18-b... Loss = 0.077327
Validating set 19-a... Loss = 0.079059
Validating set 19-b... Loss = 0.080847
Validating set 20-a... Loss = 0.075375
Validating set 20-b... Loss = 0.077266
Validating set 21-a... Loss = 0.082452
Validating set 21-b... Loss = 0.076946
Validating set 22-a... Loss = 0.078602
Validating set 22-b... Loss = 0.080538
Validating set 23-a... Loss = 0.077607
Validating set 23-b... Loss = 0.077118
Validating set 24-a... Loss = 0.078705
Validating set 24-b... Loss = 0.076103
Validating set 25-a... Loss = 0.077949
Validating set 25-b... Loss = 0.079300
Validating set 26-a... Loss = 0.076988
Validating set 26-b... Loss = 0.080871
Validating set 27-a... Loss = 0.083130
Validating set 27-b... Loss = 0.078603
Validating set 28-a... Loss = 0.077575
Validating set 28-b... Loss = 0.083491
Validating set 29-a... Loss = 0.078386
Validating set 29-b... Loss = 0.077624
Validating set 30-a... Loss = 0.077397
Validating set 30-b... Loss = 0.079600
Validating set 31-a... Loss = 0.079063
Validating set 31-b... Loss = 0.081749
Validating set 32-a... Loss = 0.078600
Validating set 32-b... Loss = 0.078188
Validating set 33-a... Loss = 0.081100
Validating set 33-b... Loss = 0.082994
Validating set 34-a... Loss = 0.079658
Validating set 34-b... Loss = 0.080738
Validating set 35-a... Loss = 0.079397
Validating set 35-b... Loss = 0.076358
Validating set 36-a... Loss = 0.079430
TrainGenerator state saved
Model report generated and saved
Best model saved to C:\deeptrain\examples\dir\models\M3__model-nadam__min.079
TrainGenerator state saved
Model report generated and saved
```




```
Training has concluded.
```

User batch_size=256, file batch_size=128

[6]:
```
C['datagen'    ]['batch_size'] = 256
C['val_datagen']['batch_size'] = 256
C['model']['batch_shape'] = (256, 28, 28, 1)
```

[7]:
```
tg = init_session(C, make_autoencoder)
```

```
WARNING: multiple file extensions found in `path`; only .npy will be used
Discovered 48 files with matching format
48 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

WARNING: multiple file extensions found in `path`; only .npy will be used
Discovered 36 files with matching format
36 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

NOTE: will exclude `labels` from saving when `input_as_labels=True`; to keep 'labels',
↪ add '{labels}'to `saveskip_list` instead
Preloading superbatch ... WARNING: multiple file extensions found in `path`; only .
↪npy will be used
Discovered 48 files with matching format
... finished, w/ 6144 total samples
Train initial data prepared
Preloading superbatch ... WARNING: multiple file extensions found in `path`; only .
↪npy will be used
Discovered 36 files with matching format
... finished, w/ 4608 total samples
Val initial data prepared
Logging ON; directory (new): C:\deeptrain\examples\dir\logs\M4__model-nadam__min999.
↪000
```

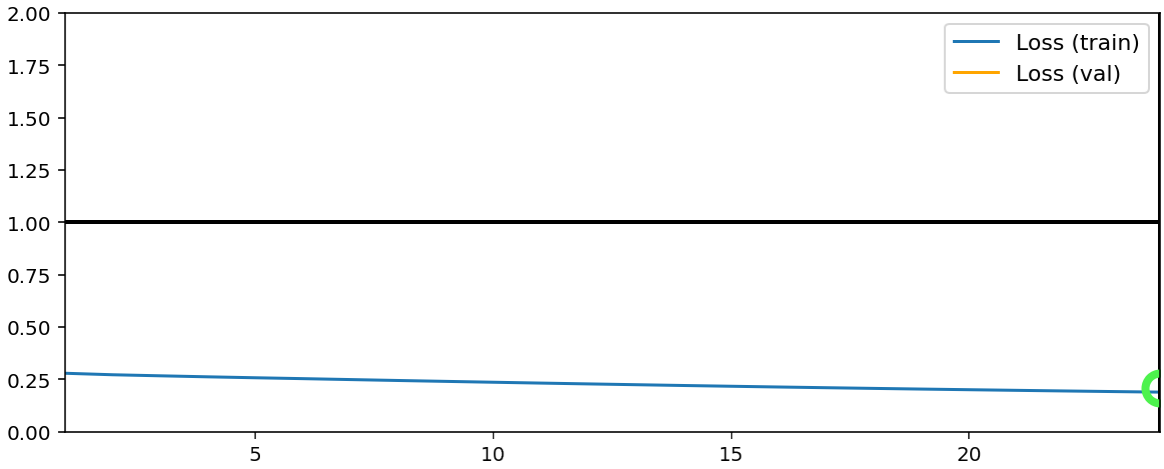
[8]:
```
tg.train()
```

```
Fitting set 1+2... Loss = 0.278780
Fitting set 3+4... Loss = 0.271617
Fitting set 5+6... Loss = 0.266600
Fitting set 7+8... Loss = 0.261748
Fitting set 9+10... Loss = 0.257241
Fitting set 11+12... Loss = 0.253015
Fitting set 13+14... Loss = 0.248581
Fitting set 15+16... Loss = 0.244207
Fitting set 17+18... Loss = 0.239981
Fitting set 19+20... Loss = 0.235842
Fitting set 21+22... Loss = 0.231791
Fitting set 23+24... Loss = 0.227870
Fitting set 25+26... Loss = 0.224052
Fitting set 27+28... Loss = 0.220263
Fitting set 29+30... Loss = 0.216646
Fitting set 31+32... Loss = 0.213080
Fitting set 33+34... Loss = 0.209693
Fitting set 35+36... Loss = 0.206401
Fitting set 37+38... Loss = 0.203193
Fitting set 39+40... Loss = 0.200143
Fitting set 41+42... Loss = 0.197141
Fitting set 43+44... Loss = 0.194222
Fitting set 45+46... Loss = 0.191372
Fitting set 47+48... Loss = 0.188653
Data set_nums shuffled



_________________________
 EPOCH 1 -- COMPLETE 



Validating...
Validating set 1+2... Loss = 0.208901
Validating set 3+4... Loss = 0.208795
Validating set 5+6... Loss = 0.208823
Validating set 7+8... Loss = 0.208704
Validating set 9+10... Loss = 0.208591
Validating set 11+12... Loss = 0.208970
Validating set 13+14... Loss = 0.208594
Validating set 15+16... Loss = 0.208772
Validating set 17+18... Loss = 0.209147
Validating set 19+20... Loss = 0.208776
Validating set 21+22... Loss = 0.208982
Validating set 23+24... Loss = 0.208814
Validating set 25+26... Loss = 0.208499
Validating set 27+28... Loss = 0.208881
Validating set 29+30... Loss = 0.208739
Validating set 31+32... Loss = 0.208822
Validating set 33+34... Loss = 0.208793
Validating set 35+36... Loss = 0.208779
TrainGenerator state saved
Model report generated and saved
Best model saved to C:\deeptrain\examples\dir\models\M4__model-nadam__min.209
TrainGenerator state saved
```

(continues on next page)

(continued from previous page)

```
Model report generated and saved
```




```
Training has concluded.
```

We can see the difference in the two settings through sets logging:

- `batch_size=64`: a `set_num` is split into `'a'` and `'b'`
- `batch_size=256`: `set_num1 + set_num2`, combining two files

Faster SSD Loading

- Save larger `batch_size` on disk (e.g. 512) than is used (e.g. 32).
- Larger files much better utilize an SSD's read speed via parallelism.
- `batch_size` on file can be as large as RAM permits.

3.5.6 Model Auto-naming

This example assumes you've read `advanced.py`, and covers:

- How to configure automatic model naming

[1]:
```
import deeptrain
deeptrain.util.misc.append_examples_dir_to_sys_path()

from utils import make_autoencoder, init_session, AE_CONFIGS as C
```

DeepTrain auto-names model based on `model_name_configs`, a dict.

- Keys denote either TrainGenerator attributes, its object's attributes (via `.`), or `model_configs` keys.
  - `'best_key_metric'` reflects the actual value, if `TrainGenerator` checkpointed since last change.
- Values denote attribute aliases; if blank or None, will use attrs as given.

[2]:
```
name_cfg = {'datagen.batch_size': 'BS',
            'filters': 'filt',
            'optimizer': '',
```

(continues on next page)

(continued from previous page)

```
            'lr': '',
            'best_key_metric': '__max'}
C['traingen'].update({'epochs': 1,
                      'model_base_name': "AE",
                      'model_name_configs': name_cfg})
C['model']['optimizer'] = 'Adam'
C['model']['lr'] = 1e-4
```

[3]:
```
tg = init_session(C, make_autoencoder)
```

```
WARNING: multiple file extensions found in `path`; only .npy will be used
Discovered 48 files with matching format
48 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

WARNING: multiple file extensions found in `path`; only .npy will be used
Discovered 36 files with matching format
36 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

NOTE: will exclude `labels` from saving when `input_as_labels=True`; to keep 'labels',
↪ add '{labels}'to `saveskip_list` instead
Preloading superbatch ... WARNING: multiple file extensions found in `path`; only .
↪npy will be used
Discovered 48 files with matching format
... finished, w/ 6144 total samples
Train initial data prepared
Preloading superbatch ... WARNING: multiple file extensions found in `path`; only .
↪npy will be used
Discovered 36 files with matching format
... finished, w/ 4608 total samples
Val initial data prepared
Logging ON; directory (new): C:\deeptrain\examples\dir\logs\M9__AE-filt6_12_2_6_12-
↪Adam-1e-4__max999.000
```

[4]:
```
tg.train()
```

```
Fitting set 1...   Loss = 0.303295
Fitting set 2...   Loss = 0.301919
Fitting set 3...   Loss = 0.301043
Fitting set 4...   Loss = 0.300576
Fitting set 5...   Loss = 0.300247
Fitting set 6...   Loss = 0.300119
Fitting set 7...   Loss = 0.299533
Fitting set 8...   Loss = 0.299117
Fitting set 9...   Loss = 0.298494
Fitting set 10...  Loss = 0.297995
Fitting set 11...  Loss = 0.297355
Fitting set 12...  Loss = 0.296885
Fitting set 13...  Loss = 0.296406
Fitting set 14...  Loss = 0.295988
Fitting set 15...  Loss = 0.295545
Fitting set 16...  Loss = 0.295146
Fitting set 17...  Loss = 0.294726
```

(continues on next page)

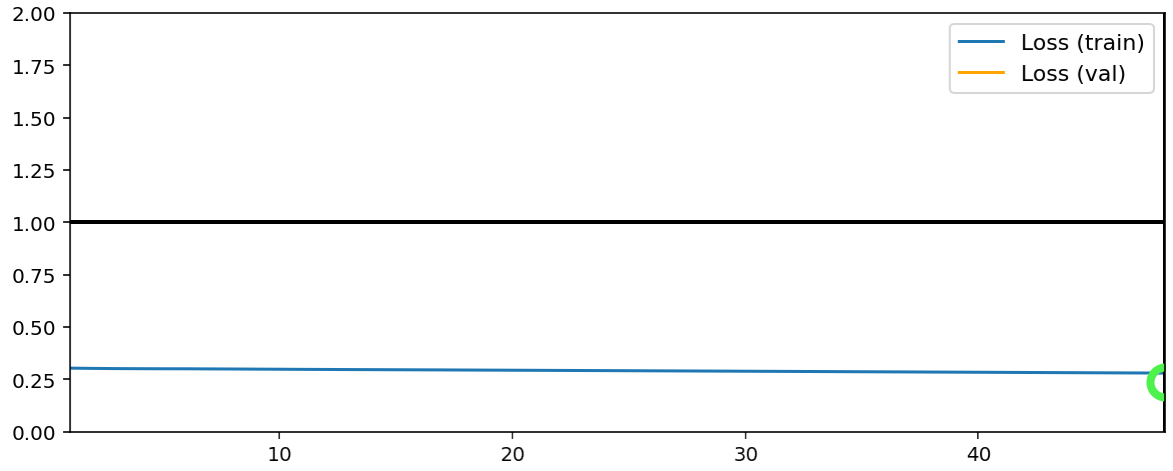
(continued from previous page)

```
Fitting set 18...  Loss = 0.294205
Fitting set 19...  Loss = 0.293743
Fitting set 20...  Loss = 0.293248
Fitting set 21...  Loss = 0.292782
Fitting set 22...  Loss = 0.292293
Fitting set 23...  Loss = 0.291838
Fitting set 24...  Loss = 0.291419
Fitting set 25...  Loss = 0.290918
Fitting set 26...  Loss = 0.290412
Fitting set 27...  Loss = 0.289942
Fitting set 28...  Loss = 0.289465
Fitting set 29...  Loss = 0.288982
Fitting set 30...  Loss = 0.288463
Fitting set 31...  Loss = 0.287996
Fitting set 32...  Loss = 0.287530
Fitting set 33...  Loss = 0.287014
Fitting set 34...  Loss = 0.286521
Fitting set 35...  Loss = 0.286011
Fitting set 36...  Loss = 0.285539
Fitting set 37...  Loss = 0.285035
Fitting set 38...  Loss = 0.284538
Fitting set 39...  Loss = 0.284004
Fitting set 40...  Loss = 0.283527
Fitting set 41...  Loss = 0.283023
Fitting set 42...  Loss = 0.282512
Fitting set 43...  Loss = 0.282022
Fitting set 44...  Loss = 0.281532
Fitting set 45...  Loss = 0.281049
Fitting set 46...  Loss = 0.280587
Fitting set 47...  Loss = 0.280085
Fitting set 48...  Loss = 0.279605
Data set_nums shuffled


_________________________
 EPOCH 1 -- COMPLETE



Validating...
Validating set 1...  Loss = 0.238562
Validating set 2...  Loss = 0.238878
Validating set 3...  Loss = 0.237879
Validating set 4...  Loss = 0.238345
Validating set 5...  Loss = 0.238605
Validating set 6...  Loss = 0.237900
Validating set 7...  Loss = 0.238646
Validating set 8...  Loss = 0.238011
Validating set 9...  Loss = 0.238837
Validating set 10... Loss = 0.238321
Validating set 11... Loss = 0.238181
Validating set 12... Loss = 0.238668
Validating set 13... Loss = 0.237183
Validating set 14... Loss = 0.239100
Validating set 15... Loss = 0.238098
Validating set 16... Loss = 0.237906
Validating set 17... Loss = 0.238677
```

(continues on next page)

(continued from previous page)

```
Validating set 18... Loss = 0.239686
Validating set 19... Loss = 0.238267
Validating set 20... Loss = 0.238836
Validating set 21... Loss = 0.238081
Validating set 22... Loss = 0.238481
Validating set 23... Loss = 0.238832
Validating set 24... Loss = 0.238768
Validating set 25... Loss = 0.238193
Validating set 26... Loss = 0.238238
Validating set 27... Loss = 0.237997
Validating set 28... Loss = 0.238209
Validating set 29... Loss = 0.238841
Validating set 30... Loss = 0.238128
Validating set 31... Loss = 0.238263
Validating set 32... Loss = 0.238241
Validating set 33... Loss = 0.238068
Validating set 34... Loss = 0.237933
Validating set 35... Loss = 0.238575
Validating set 36... Loss = 0.238656
TrainGenerator state saved
Model report generated and saved
Best model saved to C:\deeptrain\examples\dir\models\M9__AE-filt6_12_2_6_12-Adam-1e-4_
↪_max.238
TrainGenerator state saved
Model report generated and saved
```




```
Training has concluded.
```

[5]:
```
print(tg.model_name)
```

```
M9__AE-filt6_12_2_6_12-Adam-1e-4__max.238
```

Note that `logdir` and best model saves are also named with `model_name`; it, together with `model_num`, enables scalable reference to hundreds of trained models: sort through models by reading off key hyperparameters.

[6]:
```
print(tg.logdir)
print(tg.get_last_log('state', best=True))
```

```
C:\deeptrain\examples\dir\logs\M9__AE-filt6_12_2_6_12-Adam-1e-4__max999.000
C:\deeptrain\examples\dir\models\M9__AE-filt6_12_2_6_12-Adam-1e-4__max.238__state.h5
```

3.6 Callbacks

3.6.1 Basic callbacks

This example assumes you've read `advanced.ipynb`, and covers:

- Creating custom callbacks

[1]:
```
import deeptrain
deeptrain.util.misc.append_examples_dir_to_sys_path()  # for `from utils import`

from utils import make_classifier, init_session, img_labels_paths
from utils import CL_CONFIGS as C
from deeptrain.callbacks import TraingenCallback

import matplotlib.pyplot as plt
```

We can use two types of callbacks: objects (instances of `TraingenCallback`), or functions.

Callback function

Function callback takes `TrainGenerator` instance as the only argument. Below will print the total number of batches fit so far.

[2]:
```
def print_batches_fit(tg):
    print("\nBATCHES FIT: %s\n" % tg._batches_fit)
```

The next step is to specify *when* the callback is called. Callbacks are called at several stages throughout training:

- `'train:iter'`, `'train:batch'`, `'train:epoch'`
- `'val:iter'`, `'val:batch'`, `'val:epoch'`
- `'val_end'`, `'save'`, `'load'`

E.g. `'train:batch'` corresponds to `_on_batch_end` within `_train_postiter_processing` (`TrainGenerator` methods).

[3]:
```
pbf = {'train:epoch': print_batches_fit}  # print on every epoch
```

Callback object

Callback objects subclass TraingenCallback, which defines methods to override as ways to specify the *when* instead of dict keys. See `deeptrain.callbacks.TraingenCallback`.

[4]:
```
class VizWeights(TraingenCallback):
    """Show histogram of first layer's kernel weights at end of each validation."""
    def on_val_end(self, stage=None):
        # method will be called within TrainGenerator._on_val_end
        W = self.tg.model.layers[1].get_weights()[0]
        plt.hist(W.ravel(), bins=200)
        plt.show()

vizw = VizWeights()
```

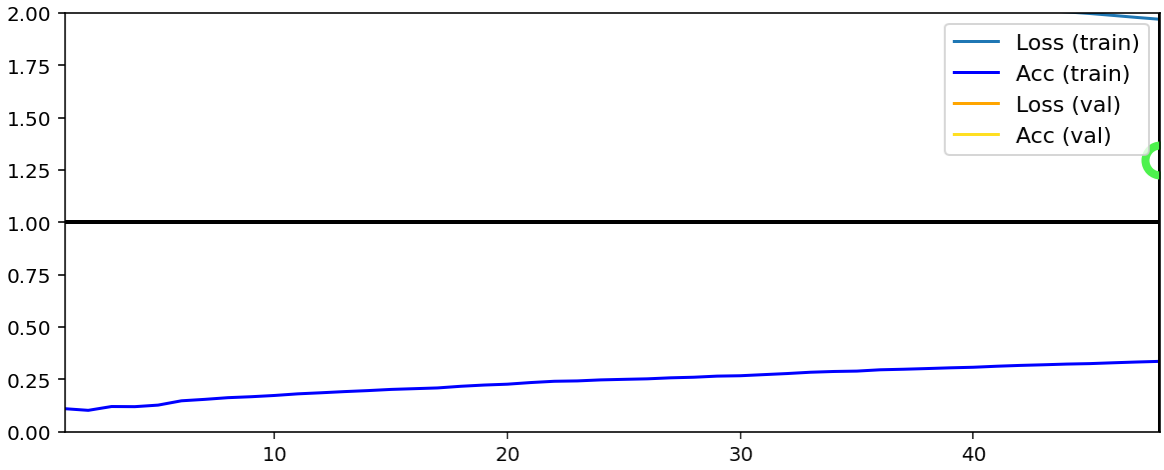
Init & train

[5]:
```
C['traingen']['epochs'] = 4
C['traingen']['callbacks'] = [pbf, vizw]
C['traingen']['iter_verbosity'] = 0
C['traingen']['plot_configs'] = {'0': {'legend_kw': {'fontsize': 11}}}
C['datagen']['labels_path']     = img_labels_paths[0]
C['val_datagen']['labels_path'] = img_labels_paths[1]
tg = init_session(C, make_classifier)
```

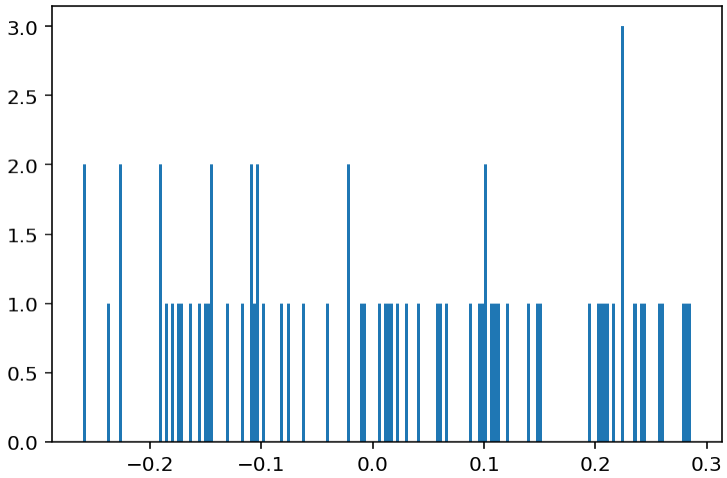
```
Discovered 48 files with matching format
Discovered dataset with matching format
48 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

Discovered 36 files with matching format
Discovered dataset with matching format
36 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

Preloading superbatch ... Discovered 48 files with matching format
... finished, w/ 6144 total samples
Train initial data prepared
Preloading superbatch ... Discovered 36 files with matching format
... finished, w/ 4608 total samples
Val initial data prepared
Logging ON; directory (new): C:\deeptrain\examples\dir\logs\M6__model-Adam__min999.000
```

[6]:
```
tg.train()
```

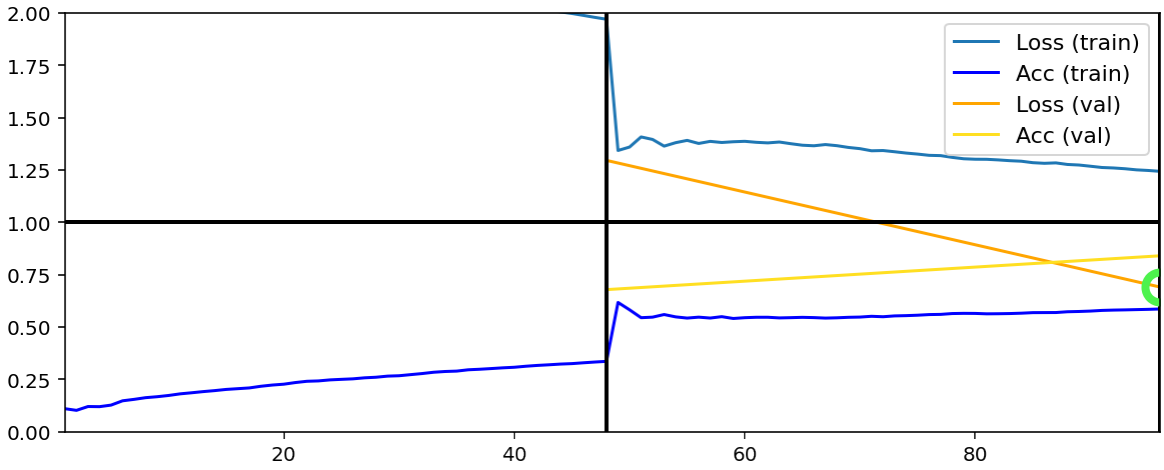
```
Data set_nums shuffled


_____________________
 EPOCH 1 -- COMPLETE


BATCHES FIT: 48



Validating...
TrainGenerator state saved
Model report generated and saved
Best model saved to C:\deeptrain\examples\dir\models\M6__model-Adam__min1.296
TrainGenerator state saved
Model report generated and saved
```



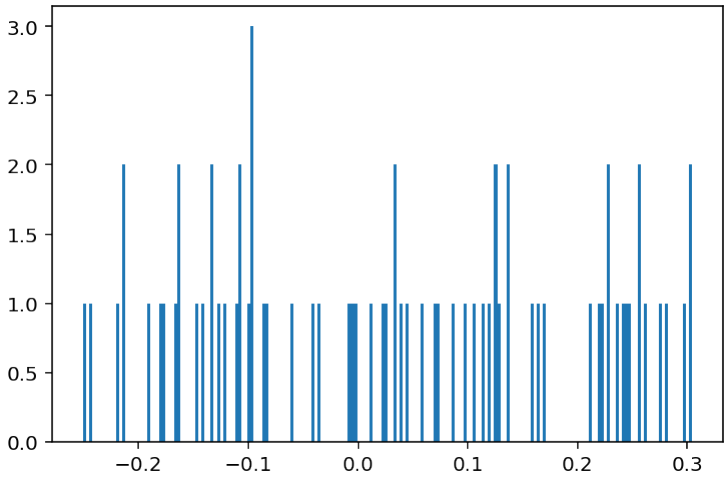

```
BATCHES FIT: 48
```



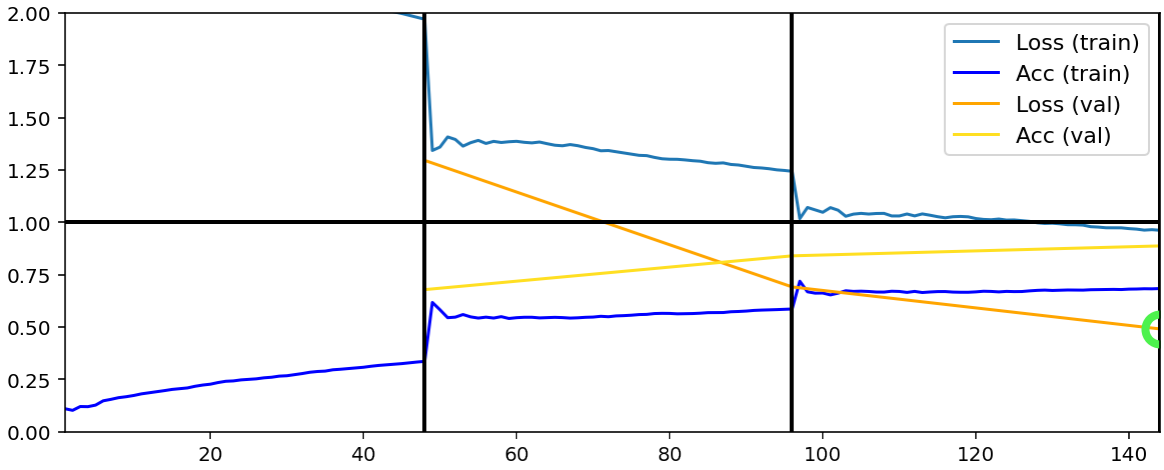

```
Data set_nums shuffled


 EPOCH 2 -- COMPLETE


BATCHES FIT: 96



Validating...
TrainGenerator state saved
Model report generated and saved
Best model saved to C:\deeptrain\examples\dir\models\M6__model-Adam__min.691
TrainGenerator state saved
Model report generated and saved
```



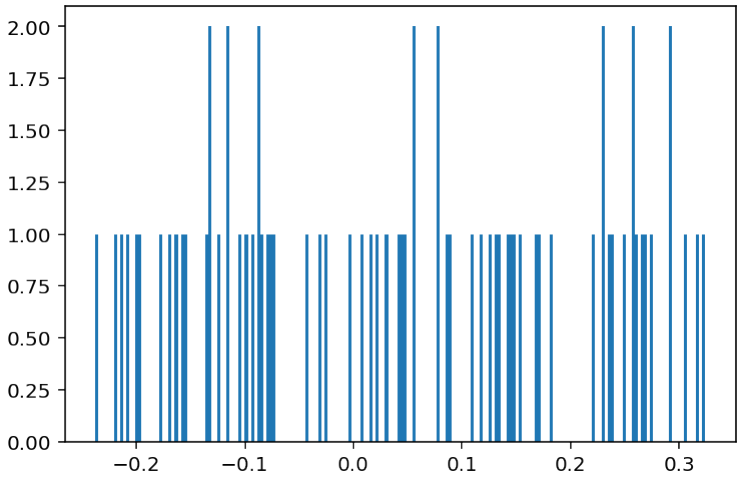

```
BATCHES FIT: 96
```



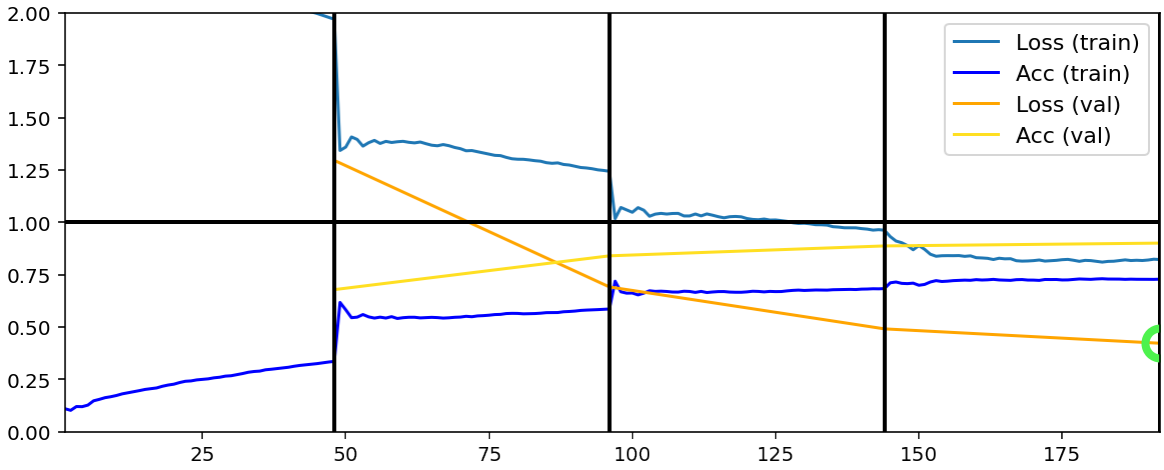

```
Data set_nums shuffled


 EPOCH 3 -- COMPLETE


BATCHES FIT: 144



Validating...
TrainGenerator state saved
Model report generated and saved
Best model saved to C:\deeptrain\examples\dir\models\M6__model-Adam__min.491
TrainGenerator state saved
Model report generated and saved
```



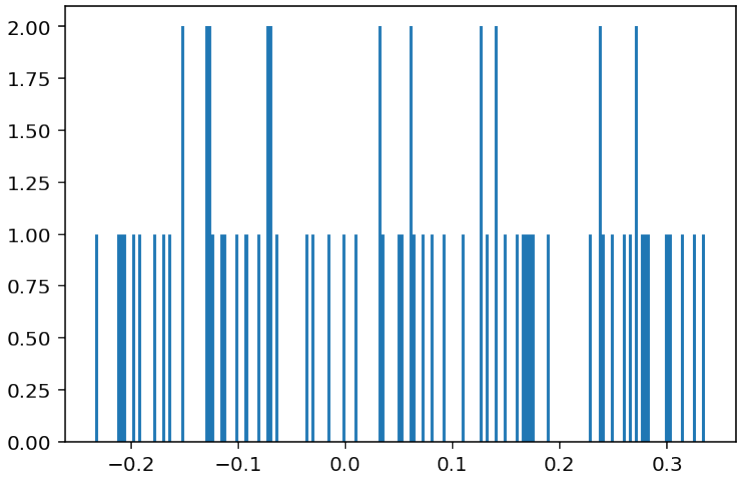

```
BATCHES FIT: 144
```




```
Data set_nums shuffled


 EPOCH 4 -- COMPLETE


BATCHES FIT: 192



Validating...
TrainGenerator state saved
Model report generated and saved
Best model saved to C:\deeptrain\examples\dir\models\M6__model-Adam__min.422
TrainGenerator state saved
Model report generated and saved
```




```
BATCHES FIT: 192
```




```
Training has concluded.
```

3.6.2 MNIST callbacks

This example assumes you've read `callbacks/basic.ipynb`, and covers:

- Creating advanced custom callbacks
- Using and modifying builtin callbacks
- Visualization and data gathering callbacks

[1]:
```
import os
import sys
import deeptrain
deeptrain.util.misc.append_examples_dir_to_sys_path()  # for `from utils import`
logger_savedir = os.path.join(sys.path[0], "logger")
```

(continues on next page)

(continued from previous page)

```
from utils import make_classifier, init_session, img_labels_paths
from utils import Adam
from utils import CL_CONFIGS as C
from see_rnn import features_2D
import numpy as np

from deeptrain.callbacks import TraingenCallback, TraingenLogger
from deeptrain.callbacks import make_layer_hists_cb
```

Data Logger

- Gathers data throughout training: weights, outputs, and gradients of model layers.
- We inherit the base class and override methods where we wish actions to occur: on save, load, and end of train epoch.

[2]:
```
class TraingenLoggerCB(TraingenLogger):
    def __init__(self, savedir, configs, **kwargs):
        super().__init__(savedir, configs, **kwargs)

    def on_save(self, stage=None):
        self.save(_id=self.tg.epoch)  # `tg` will be set inside TrainGenerator

    def on_load(self, stage=None):
        self.clear()
        self.load()

    def on_train_epoch_end(self, stage=None):
        self.log()

log_configs = {
    'weights': ['conv2d'],
    'outputs': 'conv2d',
    'gradients': ('conv2d',),
    'outputs-kw': dict(learning_phase=0),
    'gradients-kw': dict(learning_phase=0),
}
tg_logger = TraingenLoggerCB(logger_savedir, log_configs)
```

Outputs visuals

- Plots weights of the second Conv2D layer at end of each epoch.
- Weights are reshaped such that subplot 'boxes' are output channels.
- Each box plots flattened spatial dims vertically and input features horizontally.

[3]:
```
class ConvWeightsHeatmap(TraingenCallback):
    def on_val_end(self, stage=None):
        if stage == ('val_end', 'train:epoch'):
            self.viz()

    def viz(self):
        w = self.tg.model.layers[2].get_weights()[0]
        w = w.reshape(-1, *w.shape[2:])  # flatten along spatial dims
```

(continues on next page)

(continued from previous page)

```
        w = w.transpose(2, 0, 1)  # (out_features, spatial dims x in_features)

        if not hasattr(self, 'init_norm'):
            # maintain same norm throughout plots for consistency
            mx = np.max(np.abs(w))
            self.init_norm = (-mx, mx)

        features_2D(w, tight=True, w=.4, h=.4, title=None, show_xy_ticks=0,
                    norm=self.init_norm)

cwh = ConvWeightsHeatmap()
```

Callbacks can also be configured as str-function dict pairs, where str is name of a callback "stage" (see `tg._cb_alias after tg.train()`).

[4]:
```
kw = {'configs': {'annot': {'fontsize': 11}}}
grad_hists = {'train:epoch': [make_layer_hists_cb(mode='gradients:outputs', **kw),
                              make_layer_hists_cb(mode='gradients:weights', **kw)]}
weight_hists = {('val_end', 'train:epoch'): make_layer_hists_cb(mode='weights', **kw)}

configs = {'title': dict(fontsize=13), 'plot': dict(annot_kw=None), **kw['configs']}
layer_outputs_hists = {'val_end':
                       make_layer_hists_cb(mode='outputs', configs=configs)}
```

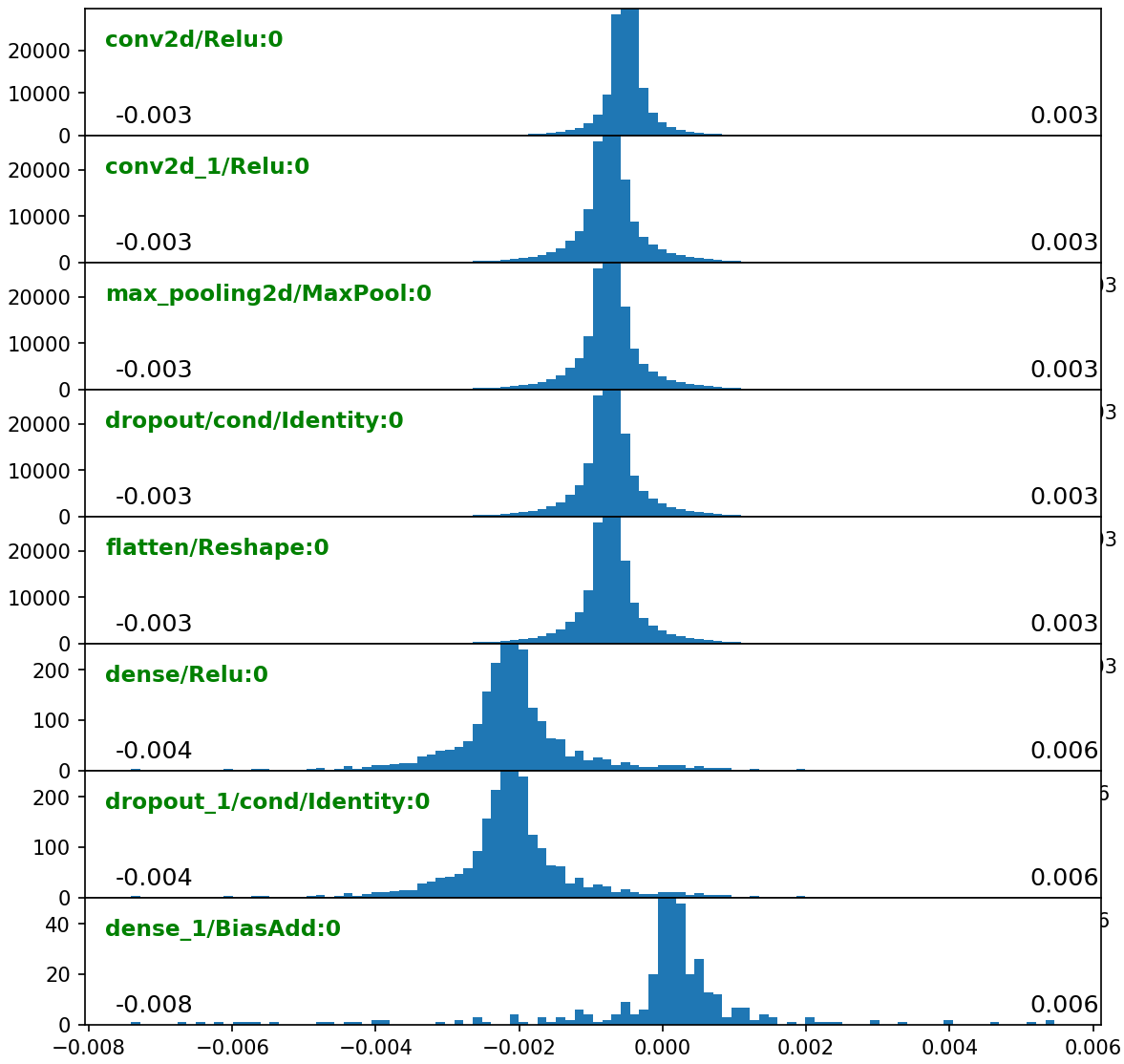
Init & train

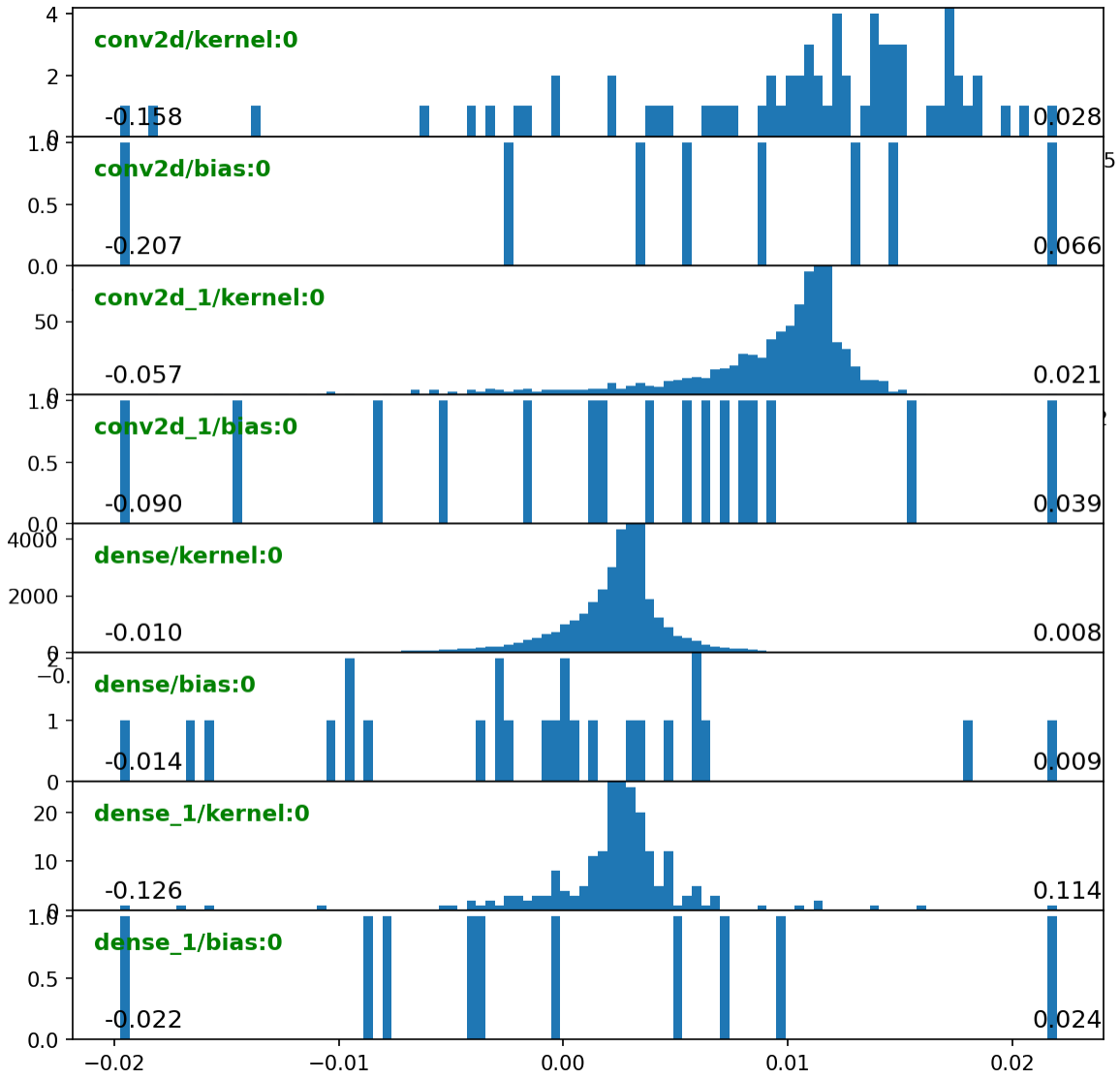
[5]:
```
C['traingen']['callbacks'] = [tg_logger, cwh, grad_hists,
                              weight_hists, layer_outputs_hists]
C['traingen']['epochs'] = 4
C['traingen']['iter_verbosity'] = 0
C['traingen']['plot_configs'] = {'0': {'legend_kw': {'fontsize': 11}}}
C['datagen']['labels_path']     = img_labels_paths[0]
C['val_datagen']['labels_path'] = img_labels_paths[1]
C['model']['optimizer'] = Adam(1e-2)
tg = init_session(C, make_classifier)
```

```
Discovered 48 files with matching format
Discovered dataset with matching format
48 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

Discovered 36 files with matching format
Discovered dataset with matching format
36 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

Preloading superbatch ... Discovered 48 files with matching format
... finished, w/ 6144 total samples
Train initial data prepared
Preloading superbatch ... Discovered 36 files with matching format
... finished, w/ 4608 total samples
Val initial data prepared
Logging ON; directory (new): C:\deeptrain\examples\dir\logs\M2__model-Adam__min999.000
```

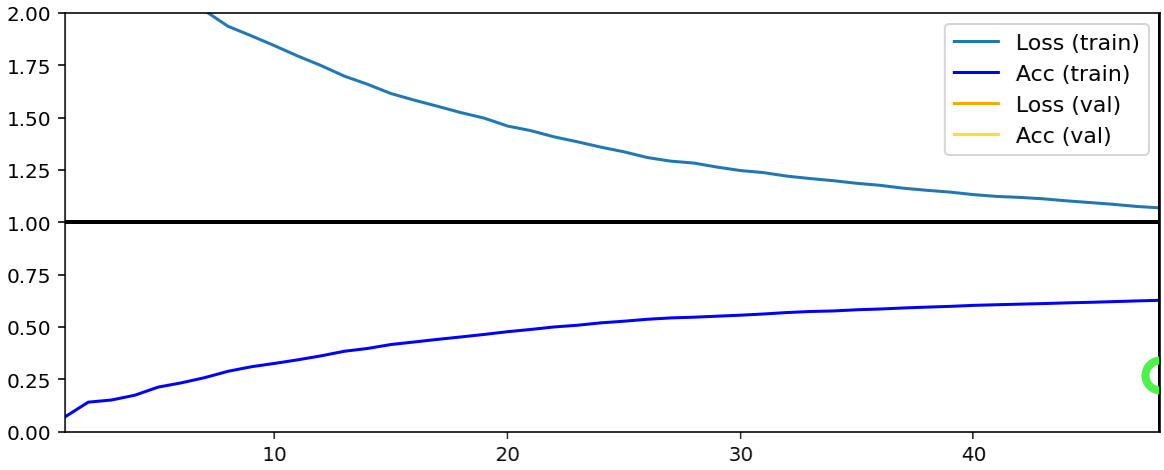
[6]:
```
tg.train()
```

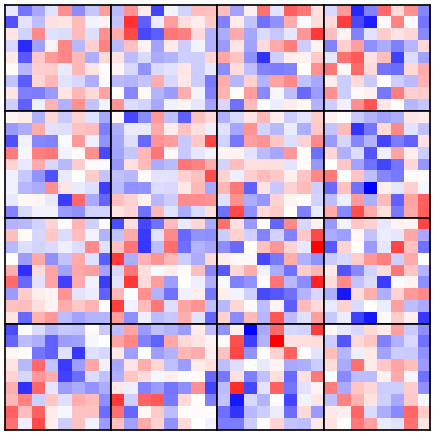
```
Data set_nums shuffled


_____________________
 EPOCH 1 -- COMPLETE
```

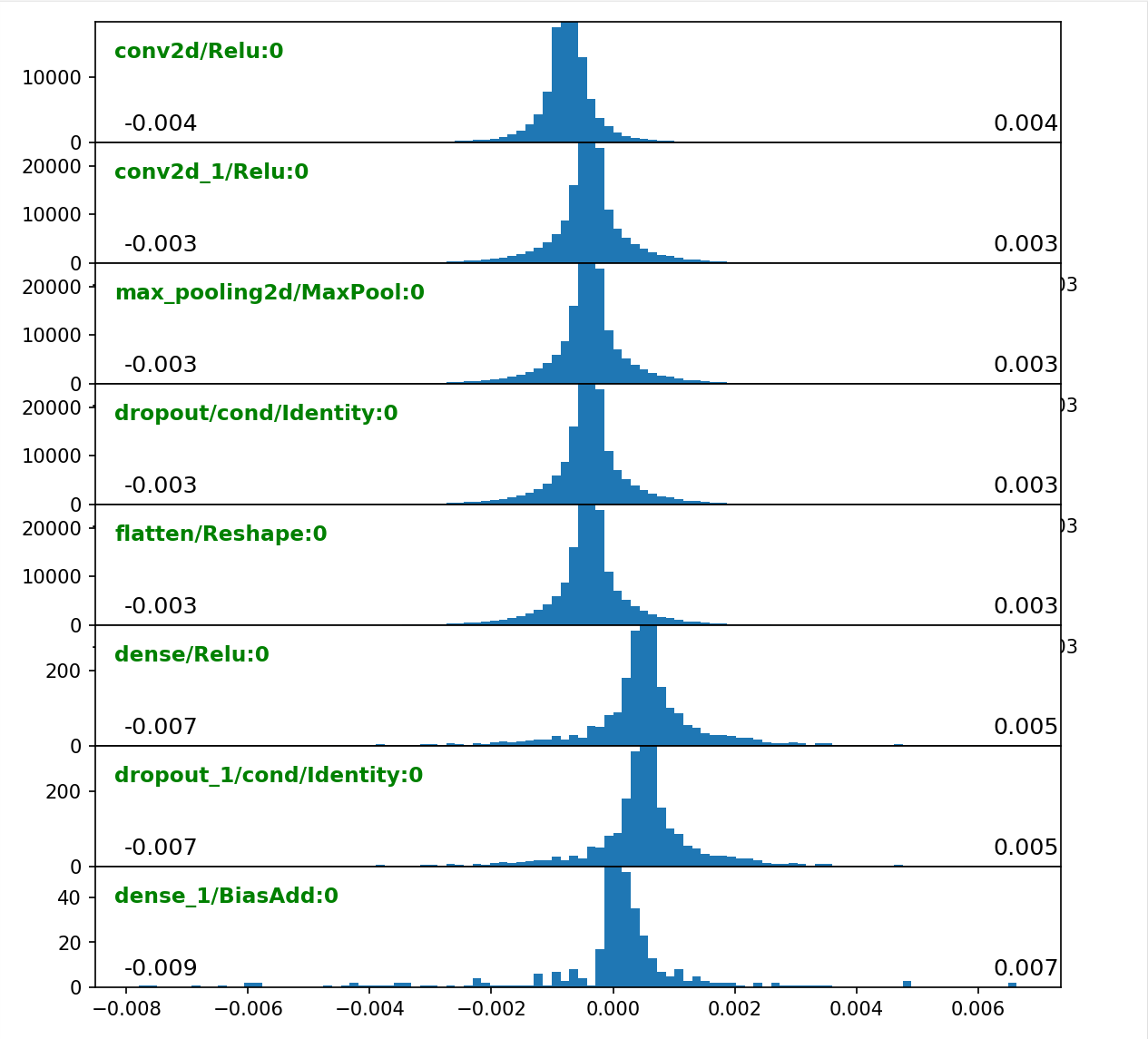


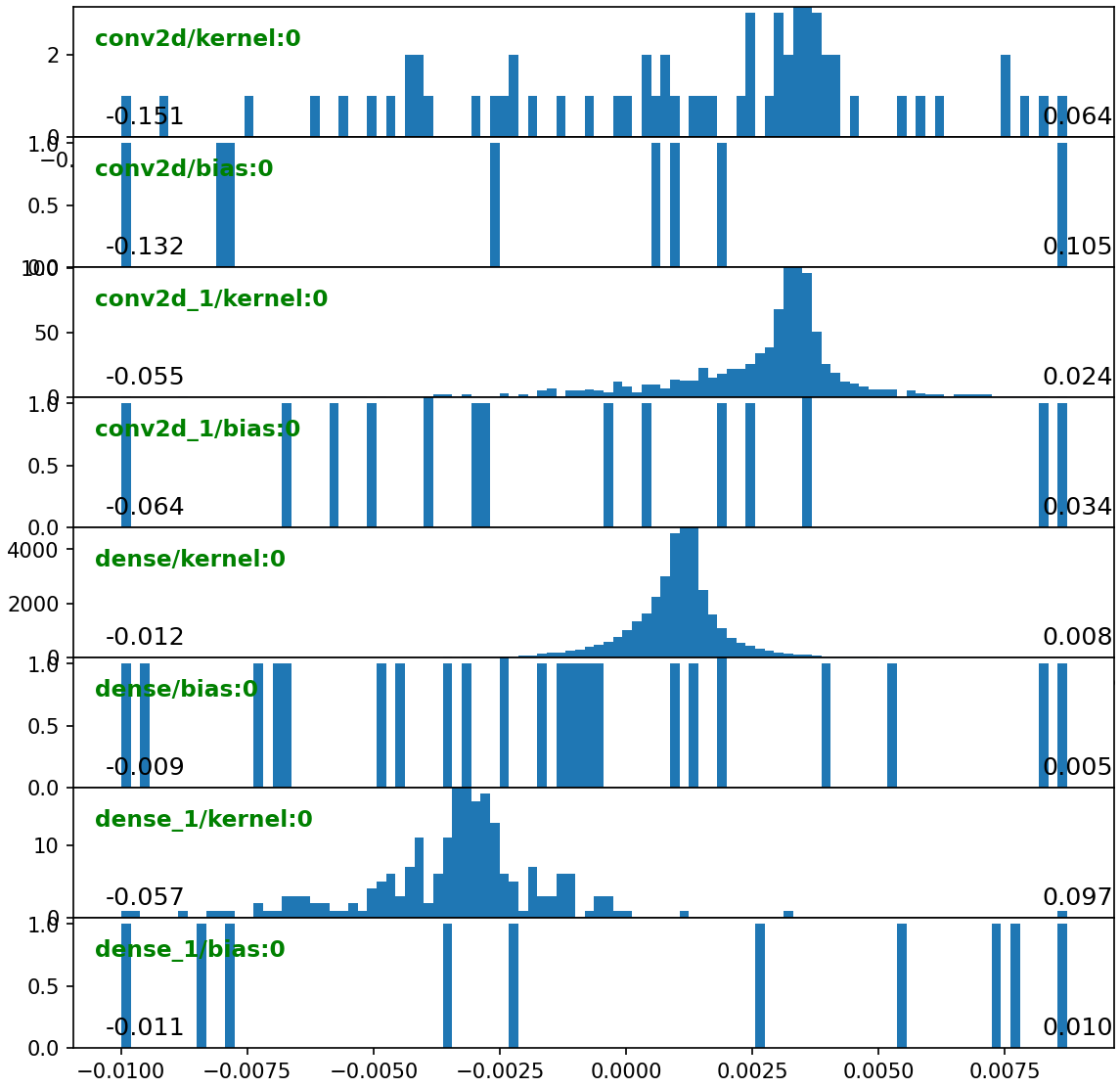


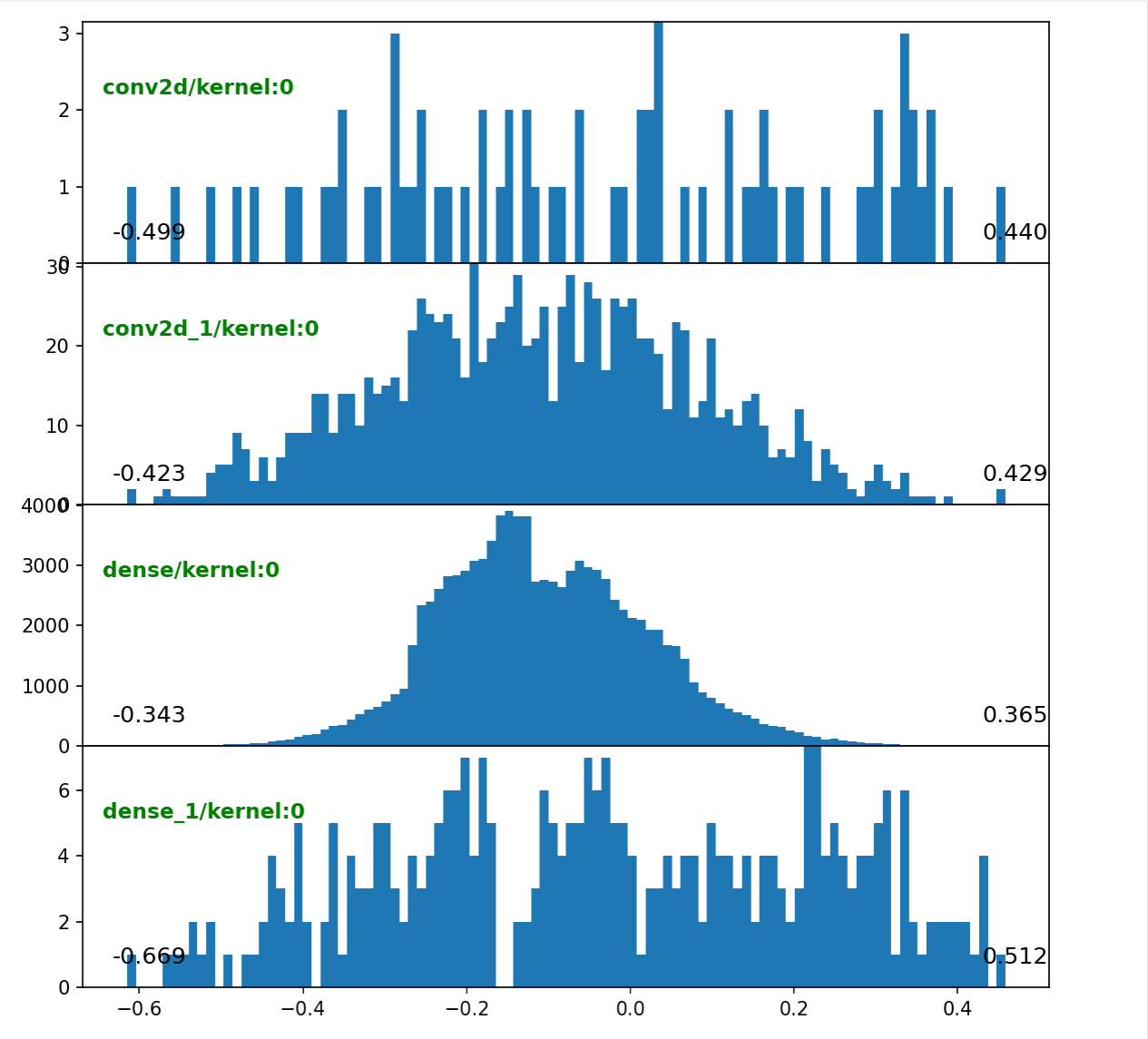

```
Validating...
TraingenLogger data saved to C:\deeptrain\examples\dir\logger\datalog_1.h5
TrainGenerator state saved
Model report generated and saved
Best model saved to C:\deeptrain\examples\dir\models\M2__model-Adam__min.269
TraingenLogger data saved to C:\deeptrain\examples\dir\logger\datalog_1.h5
TrainGenerator state saved
Model report generated and saved
```

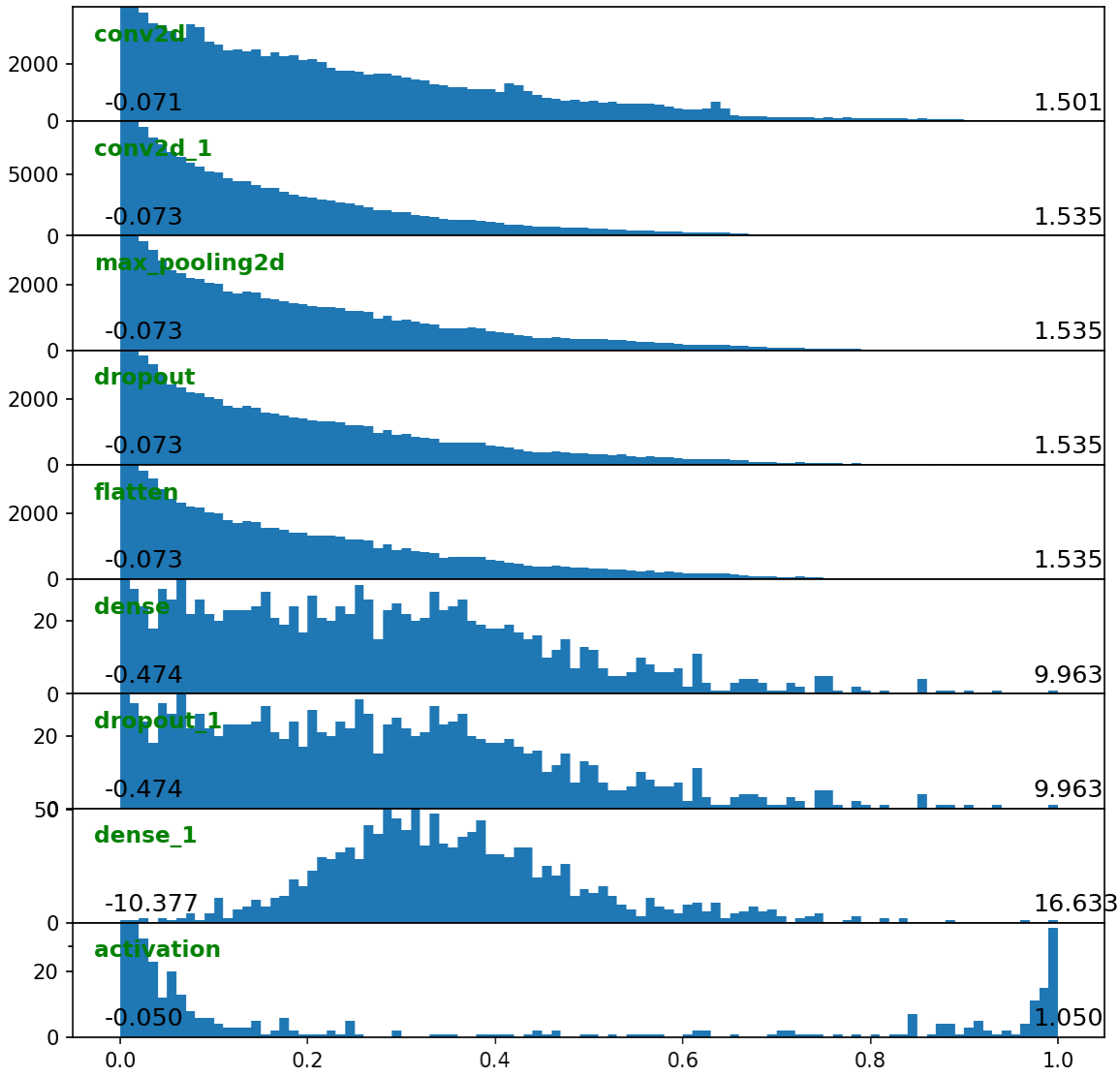

2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
10
20
30
40
Loss (train)
Acc (train)
Loss (val)
Acc (val)



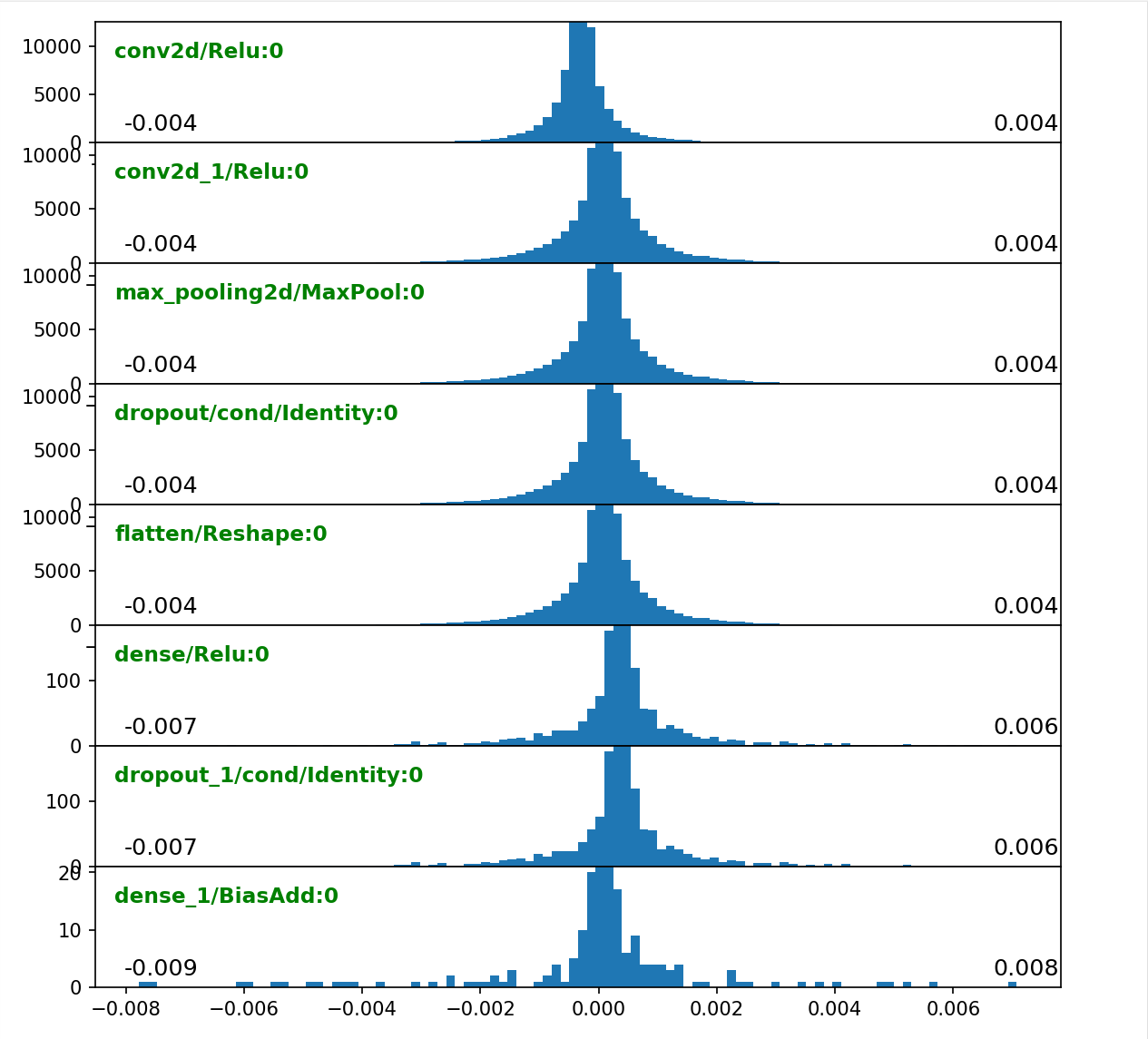

conv2d/Relu:0
-0.004
0.004
conv2d_1/Relu:0
-0.003
0.003
max_pooling2d/MaxPool:0
-0.003
0.003
dropout/cond/Identity:0
-0.003
0.003
flatten/Reshape:0
-0.003
0.003
dense/Relu:0
-0.007
0.005
dropout_1/cond/Identity:0
-0.007
0.005
dense_1/BiasAdd:0
-0.009
0.007
−0.008
−0.006
−0.004
−0.002
0.000
0.002
0.004
0.006

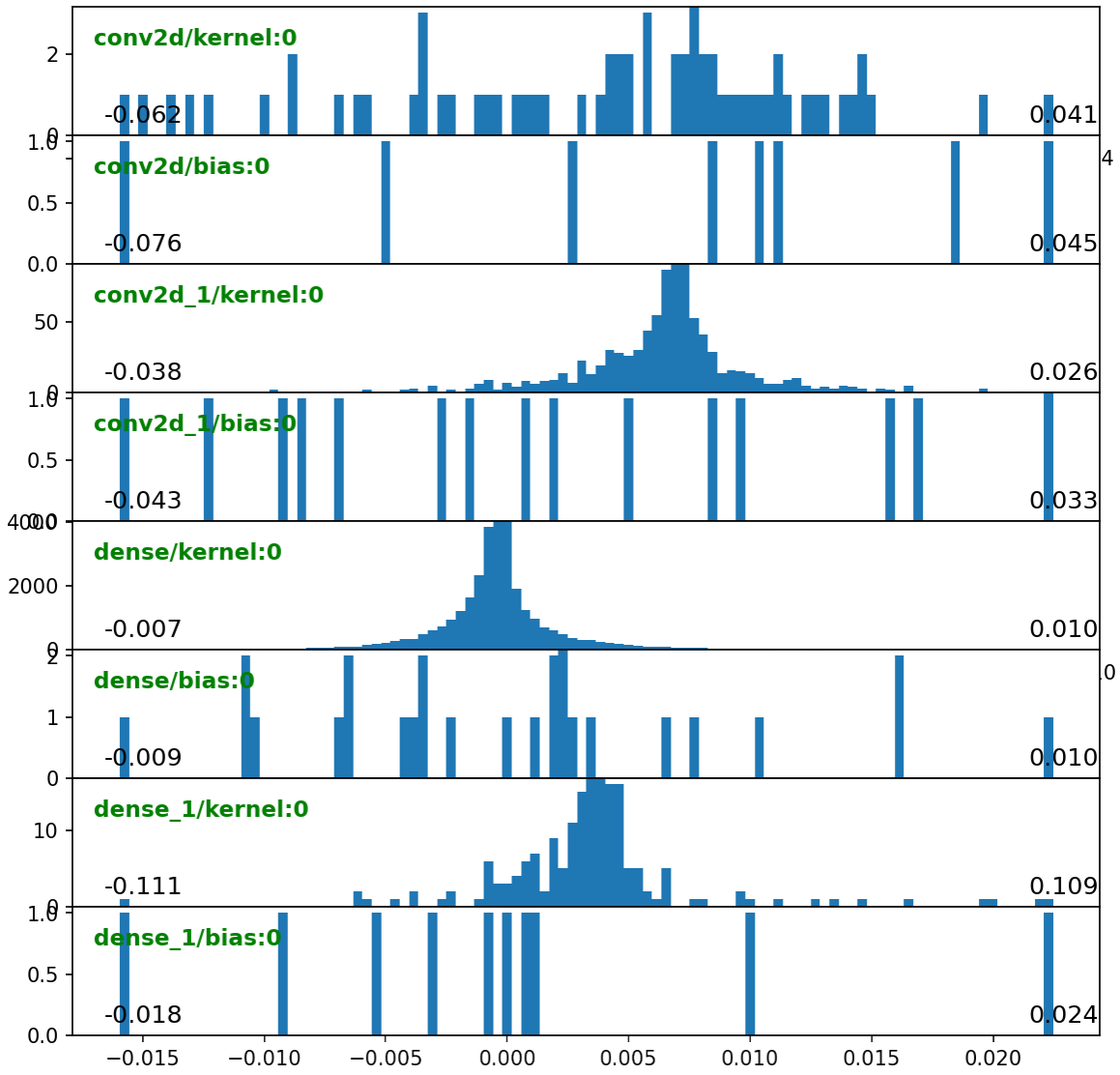

conv2d/kernel:0
-0.151
0.064
conv2d/bias:0
-0.132
0.105
conv2d_1/kernel:0
-0.055
0.024
conv2d_1/bias:0
-0.064
0.034
dense/kernel:0
-0.012
0.008
dense/bias:0
-0.009
0.005
dense_1/kernel:0
-0.057
0.097
dense_1/bias:0
-0.011
0.010
−0.0100
−0.0075
−0.0050
−0.0025
0.0000
0.0025
0.0050
0.0075


conv2d/kernel:0
-0.499
0.440
conv2d_1/kernel:0
-0.423
0.429
dense/kernel:0
-0.343
0.365
dense_1/kernel:0
-0.669
0.512
−0.6
−0.4
−0.2
0.0
0.2
0.4

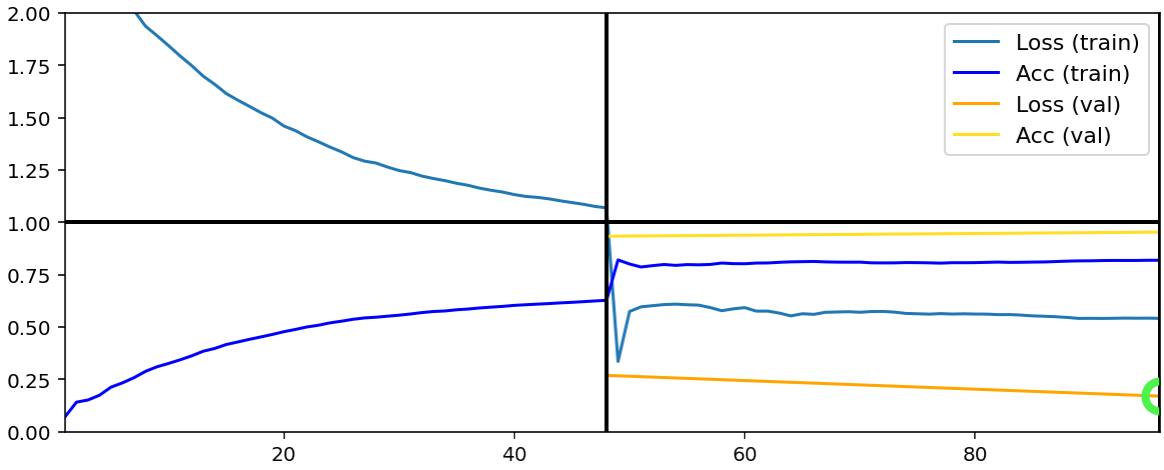


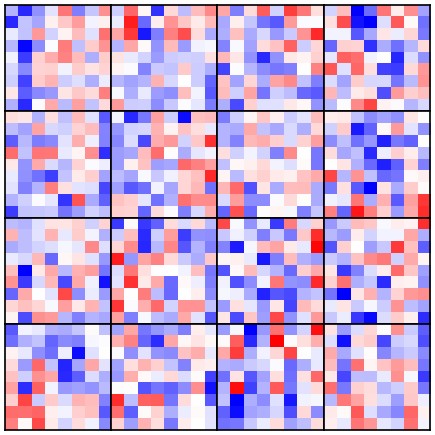

```
Data set_nums shuffled


 EPOCH 2 -- COMPLETE
```

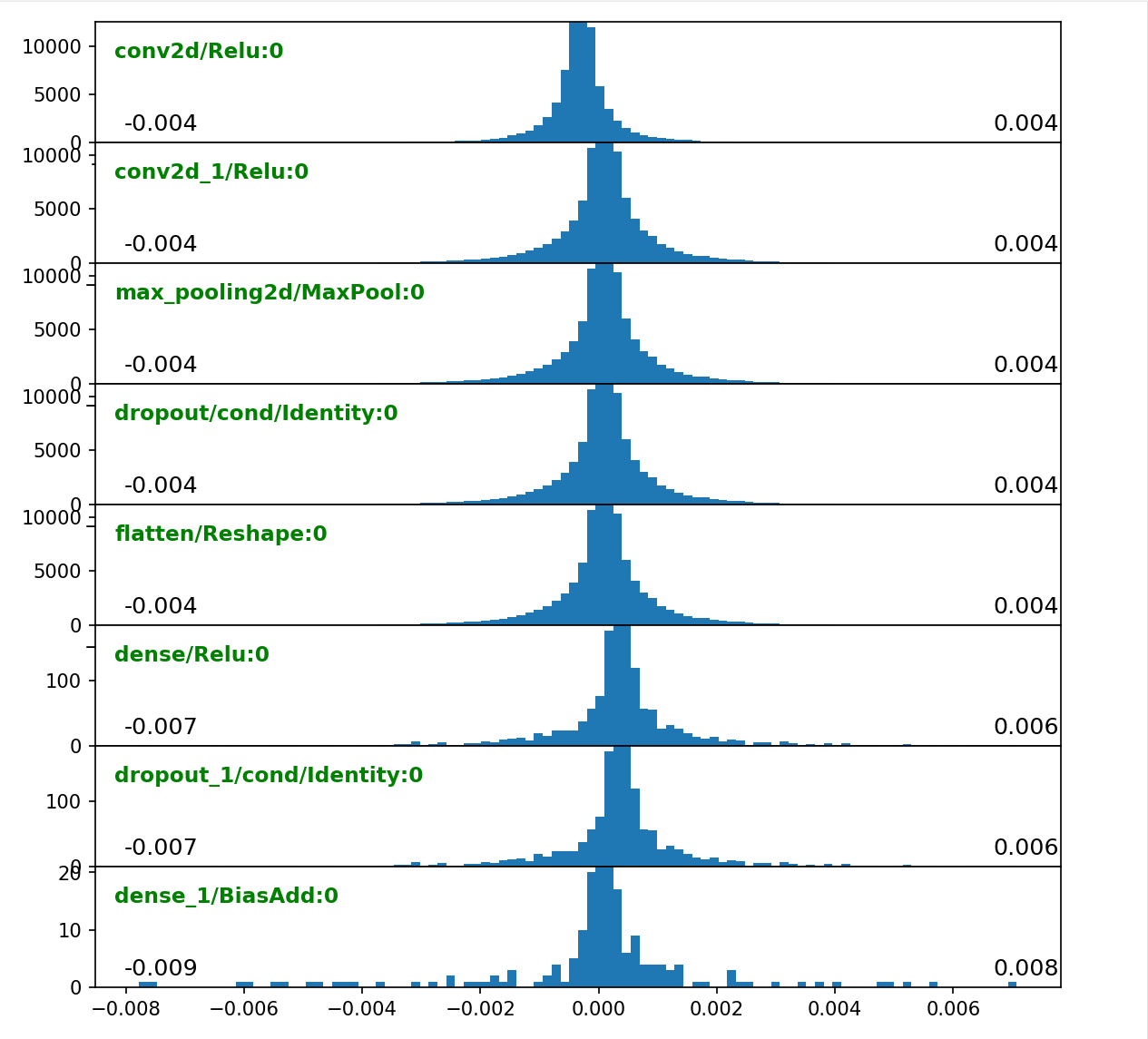

conv2d/Relu:0
-0.004
0.004
conv2d_1/Relu:0
-0.004
0.004
max_pooling2d/MaxPool:0
-0.004
0.004
dropout/cond/Identity:0
-0.004
0.004
flatten/Reshape:0
-0.004
0.004
dense/Relu:0
-0.007
0.006
dropout_1/cond/Identity:0
-0.007
0.006
dense_1/BiasAdd:0
-0.009
0.008
−0.008
−0.006
−0.004
−0.002
0.000
0.002
0.004
0.006

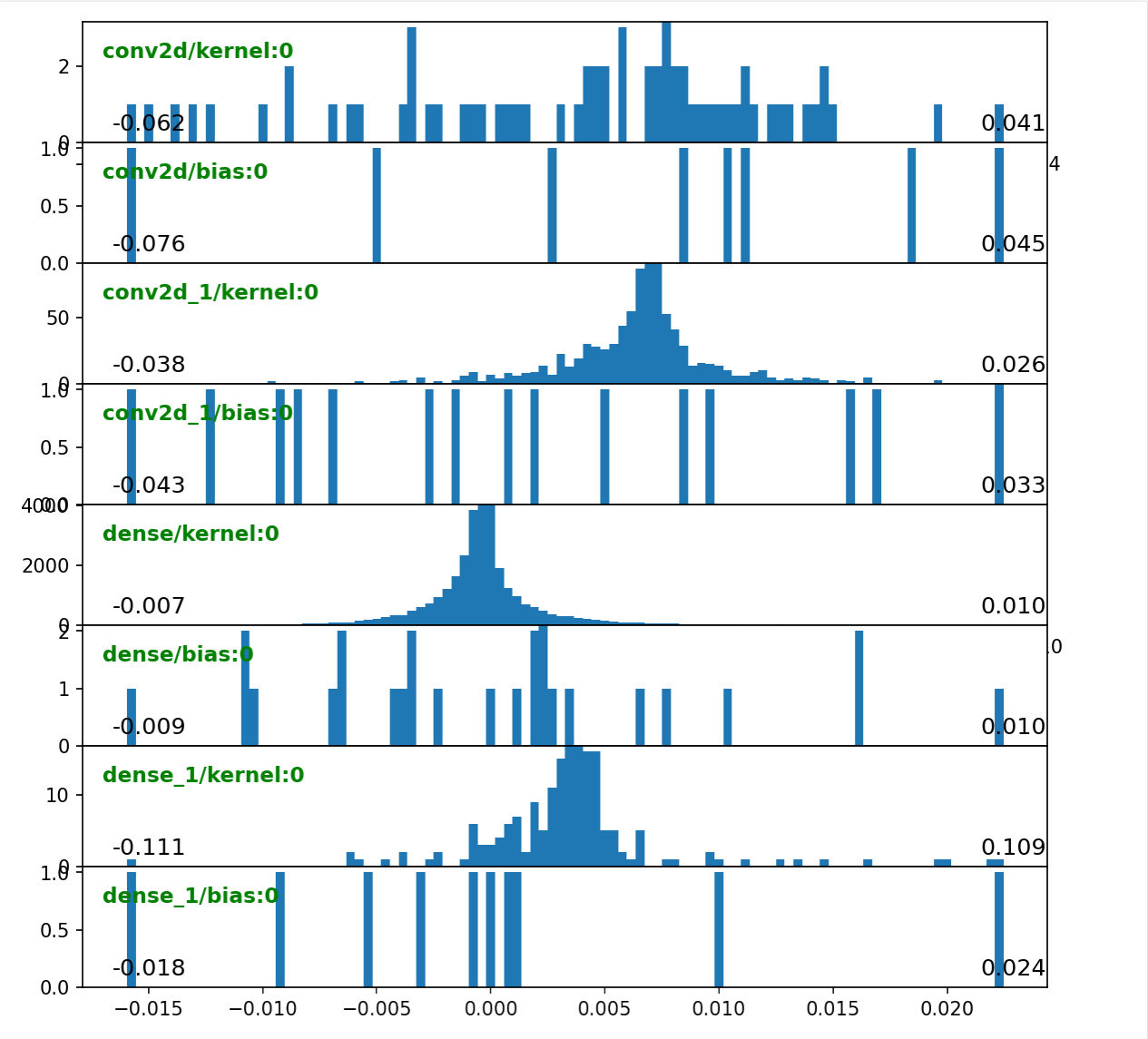


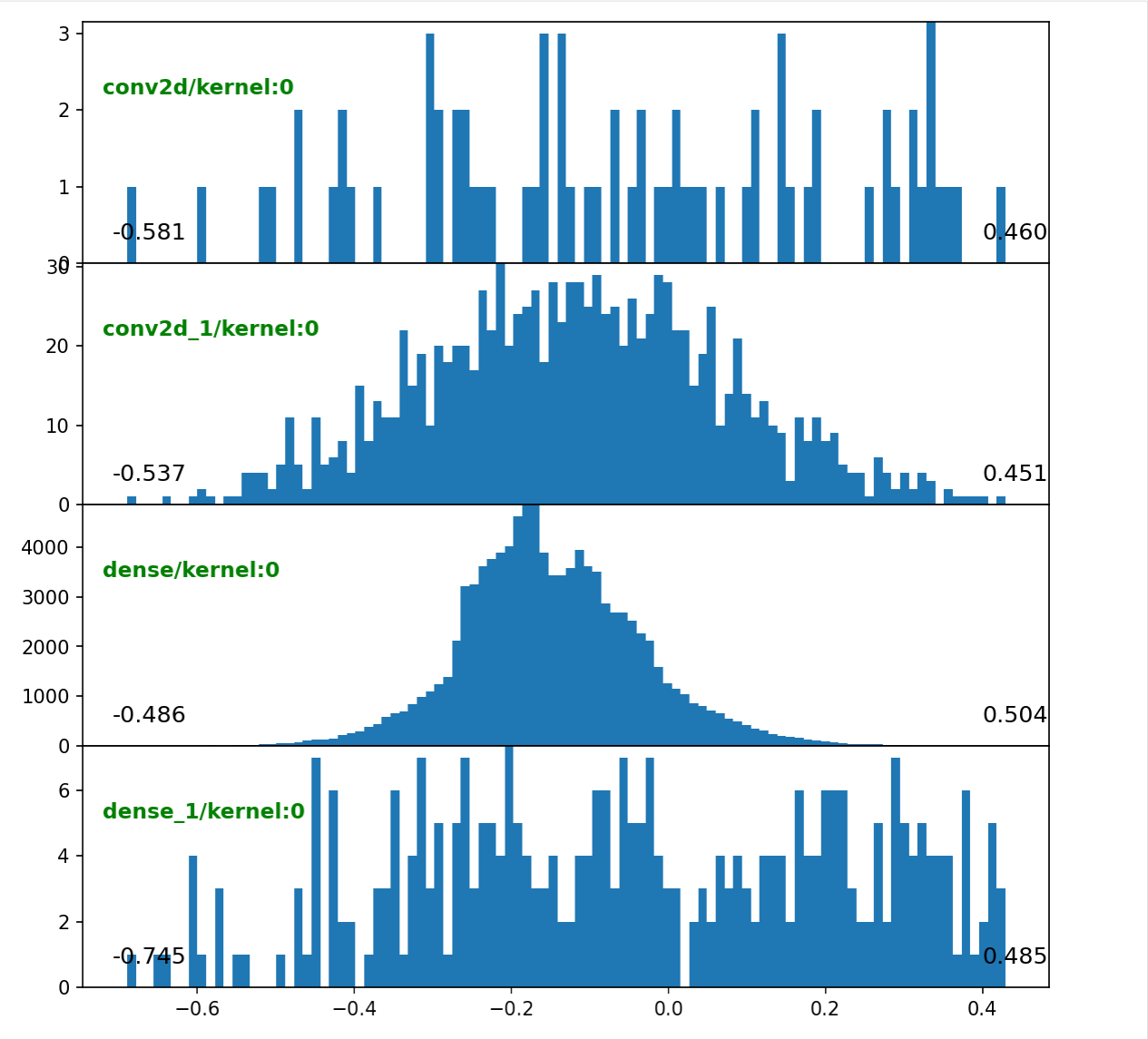

```
Validating...
TraingenLogger data saved to C:\deeptrain\examples\dir\logger\datalog_2.h5
TrainGenerator state saved
Model report generated and saved
Best model saved to C:\deeptrain\examples\dir\models\M2__model-Adam__min.170
TraingenLogger data saved to C:\deeptrain\examples\dir\logger\datalog_2.h5
TrainGenerator state saved
Model report generated and saved
```

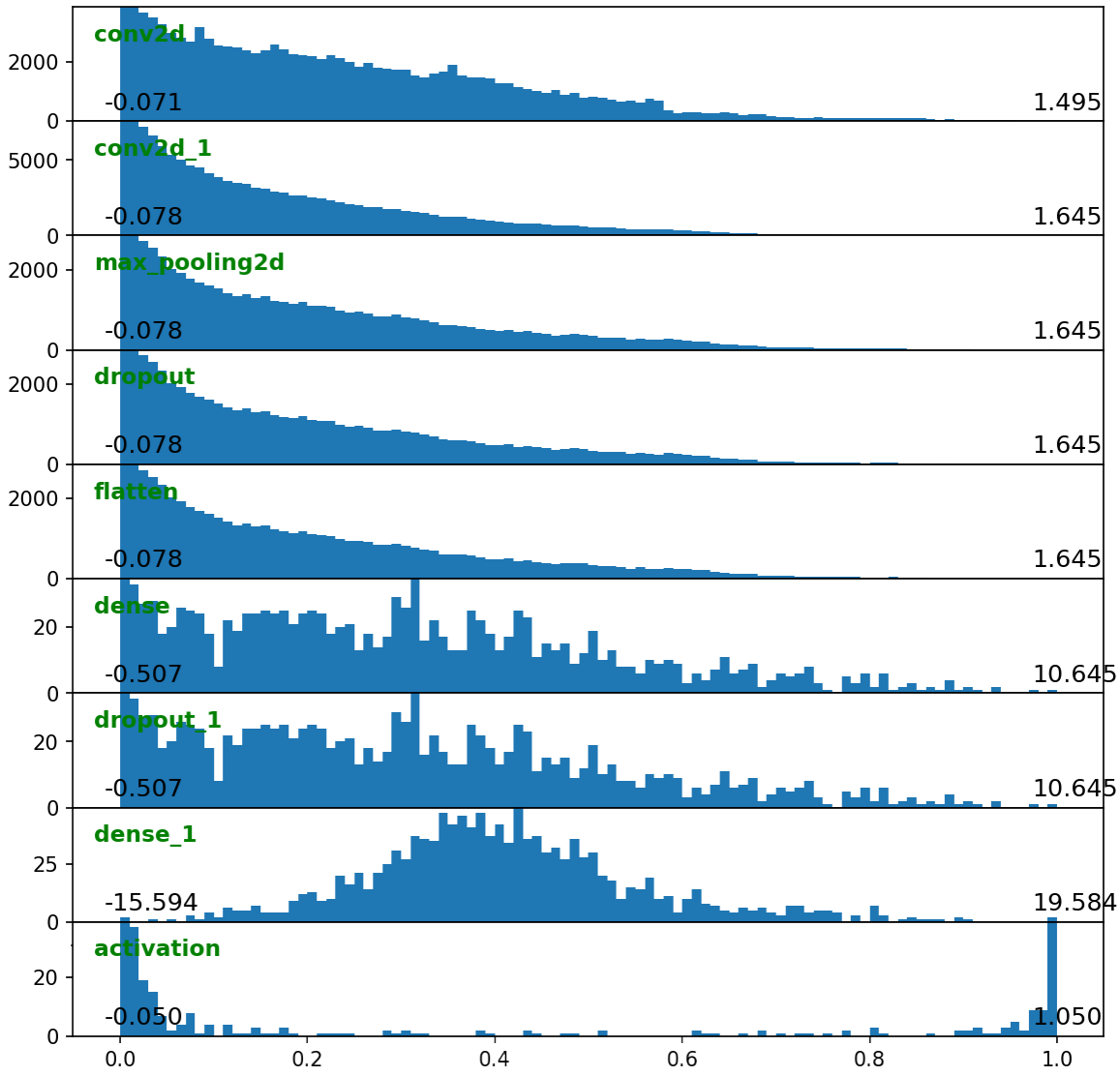

2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
20
40
60
80
Loss (train)
Acc (train)
Loss (val)
Acc (val)



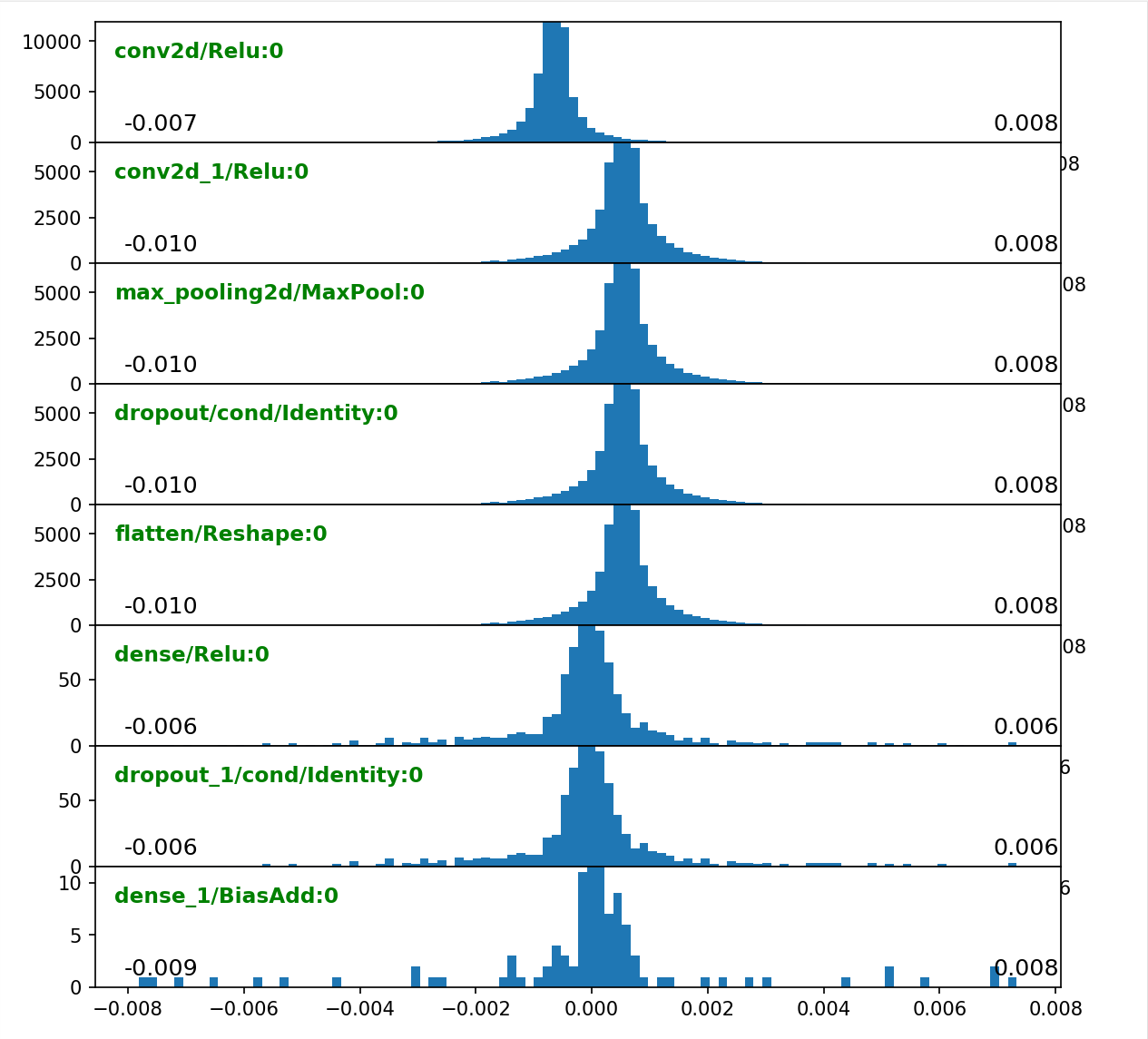

conv2d/Relu:0
-0.004
0.004
conv2d_1/Relu:0
-0.004
0.004
max_pooling2d/MaxPool:0
-0.004
0.004
dropout/cond/Identity:0
-0.004
0.004
flatten/Reshape:0
-0.004
0.004
dense/Relu:0
-0.007
0.006
dropout_1/cond/Identity:0
-0.007
0.006
dense_1/BiasAdd:0
-0.009
0.008
−0.008
−0.006
−0.004
−0.002
0.000
0.002
0.004
0.006

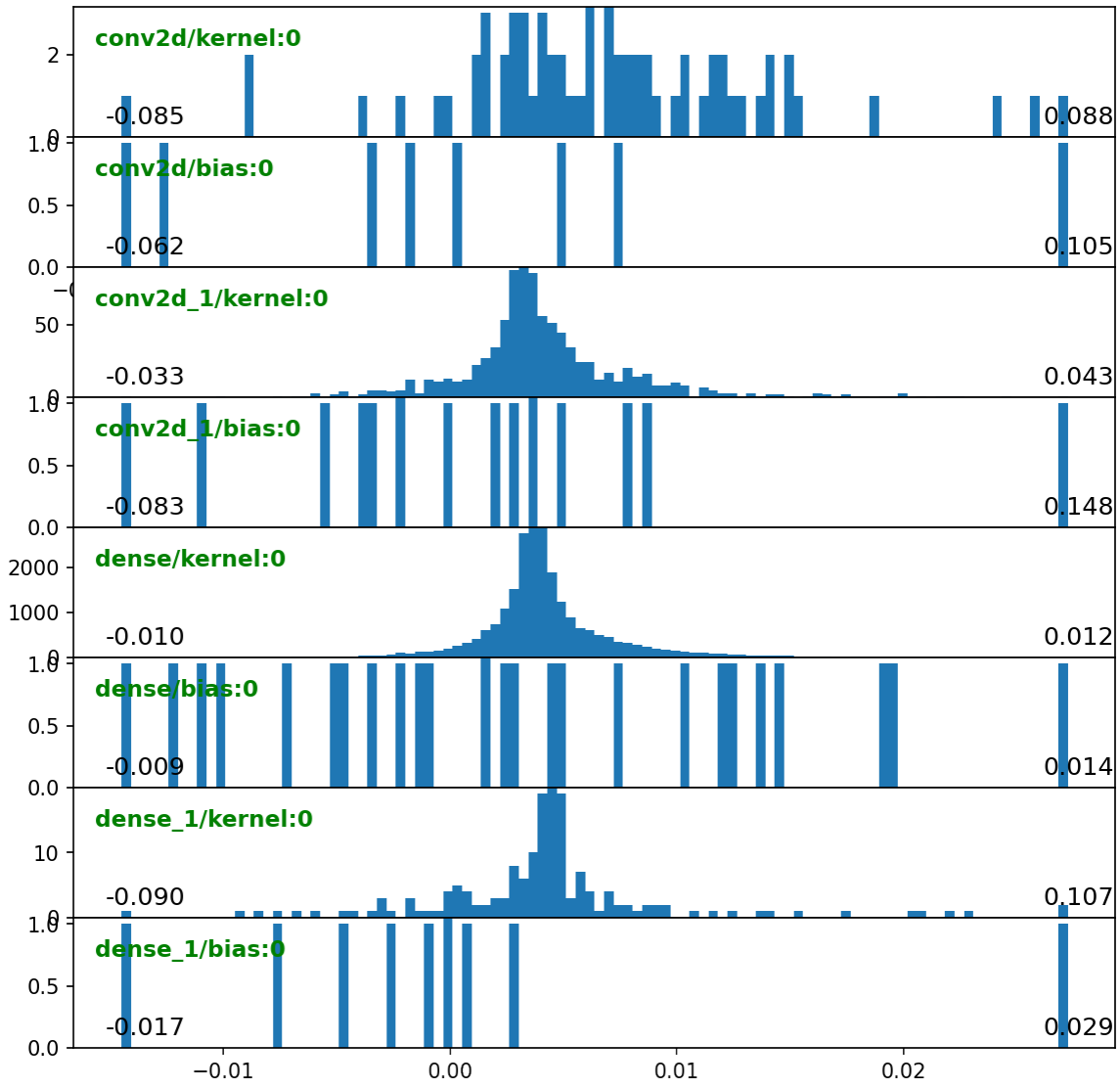

conv2d/kernel:0
-0.062
0.041
conv2d/bias:0
-0.076
0.045
conv2d_1/kernel:0
-0.038
0.026
conv2d_1/bias:0
-0.043
0.033
dense/kernel:0
-0.007
0.010
dense/bias:0
-0.009
0.010
dense_1/kernel:0
-0.111
0.109
dense_1/bias:0
-0.018
0.024
−0.015
−0.010
−0.005
0.000
0.005
0.010
0.015
0.020


conv2d/kernel:0
-0.581
0.460
conv2d_1/kernel:0
-0.537
0.451
dense/kernel:0
-0.486
0.504
dense_1/kernel:0
-0.745
0.485
−0.6
−0.4
−0.2
0.0
0.2
0.4

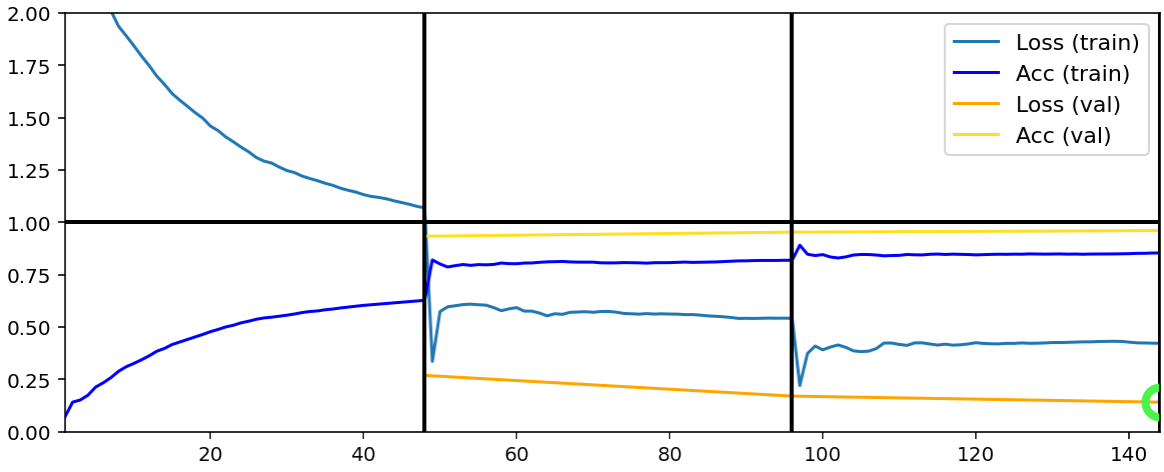


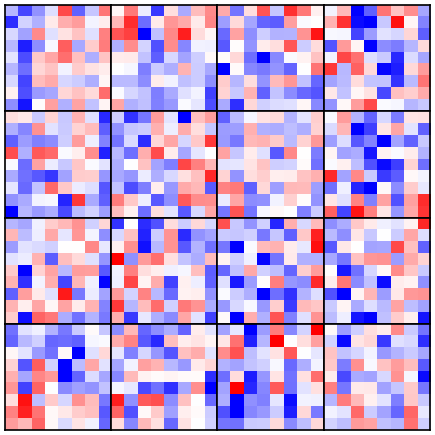

```
Data set_nums shuffled


 EPOCH 3 -- COMPLETE
```

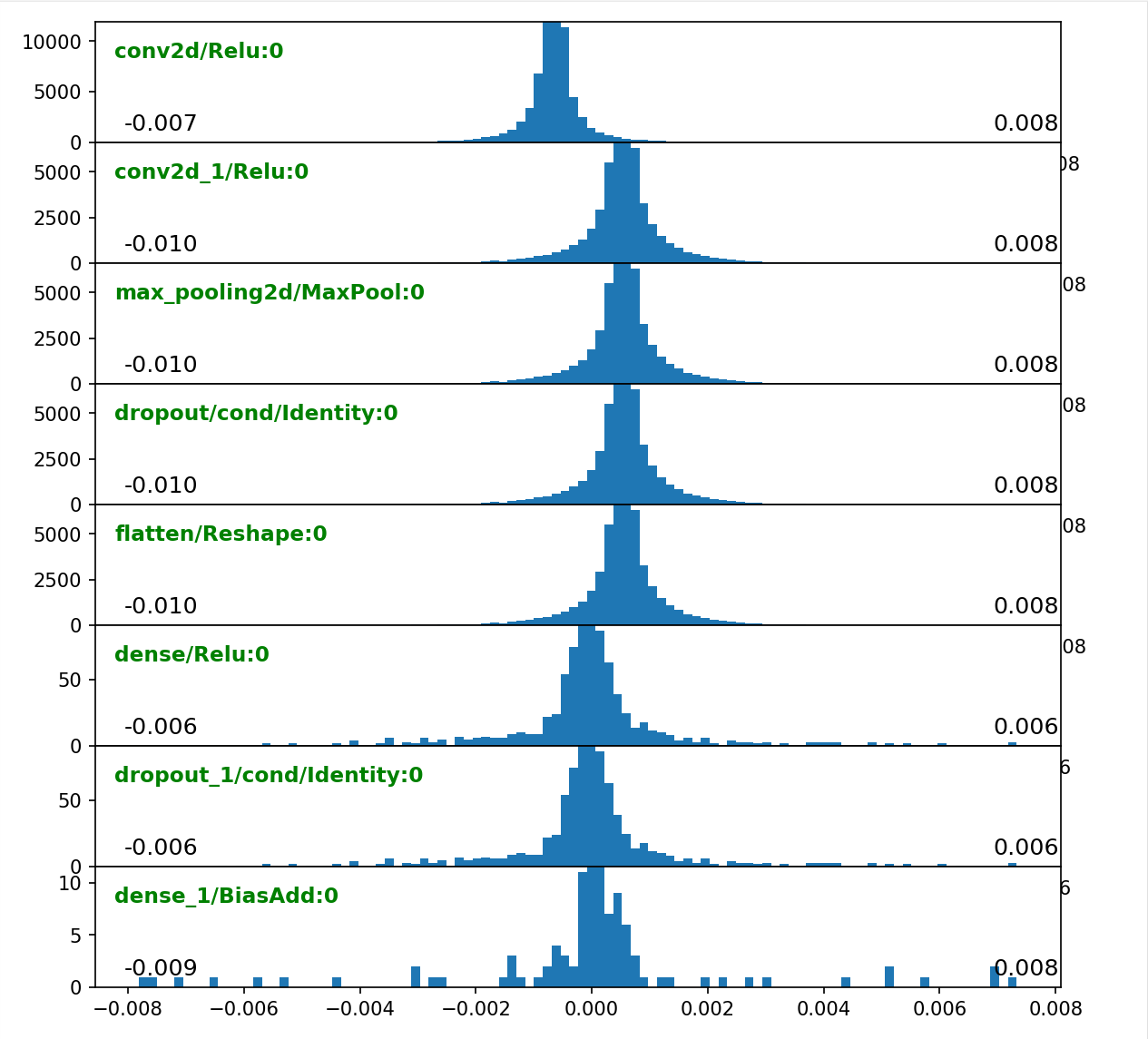

conv2d/Relu:0
-0.007
0.008
conv2d_1/Relu:0
-0.010
0.008
max_pooling2d/MaxPool:0
-0.010
0.008
dropout/cond/Identity:0
-0.010
0.008
flatten/Reshape:0
-0.010
0.008
dense/Relu:0
-0.006
0.006
dropout_1/cond/Identity:0
-0.006
0.006
dense_1/BiasAdd:0
-0.009
0.008
−0.008
−0.006
−0.004
−0.002
0.000
0.002
0.004
0.006
0.008

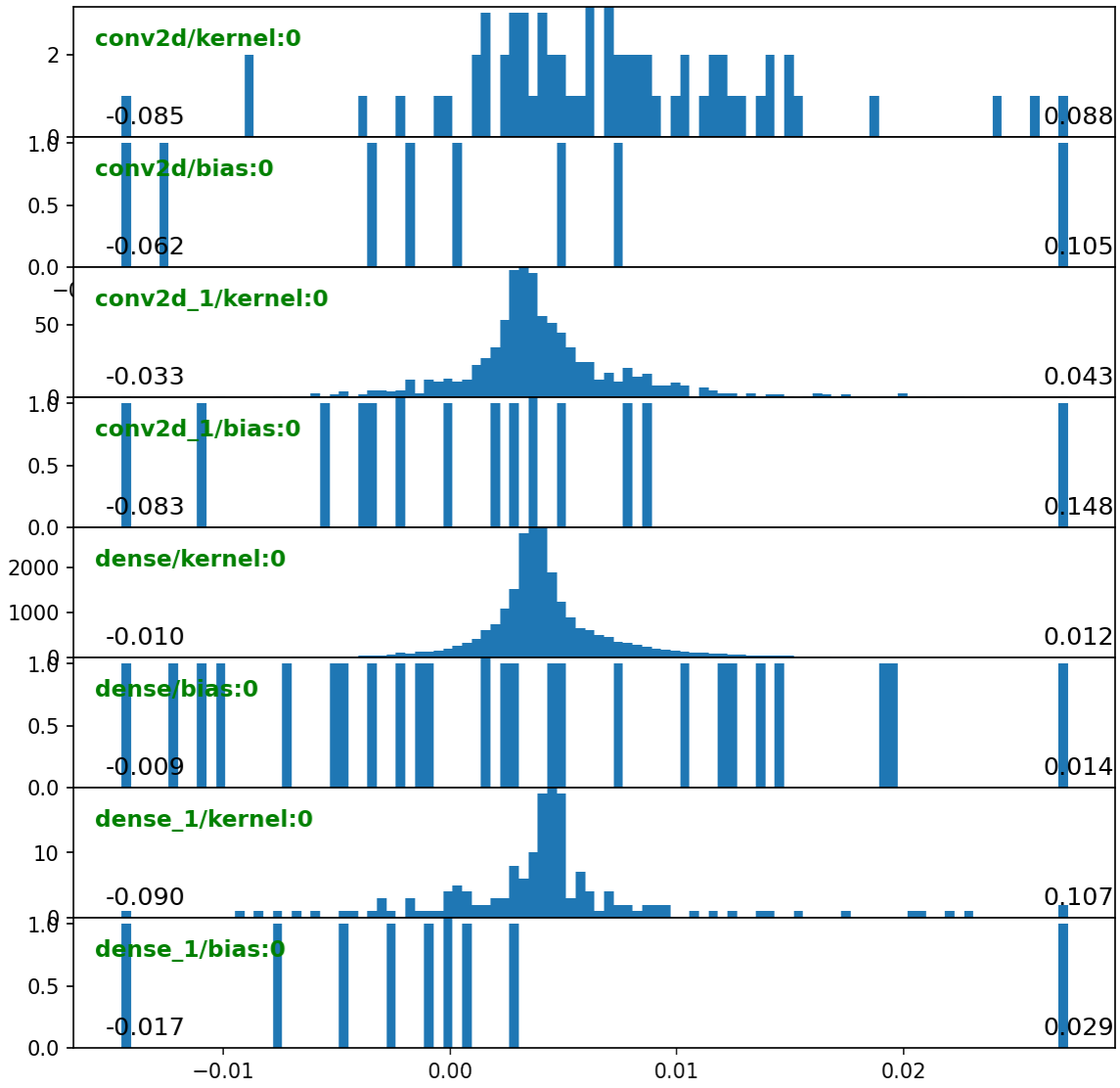


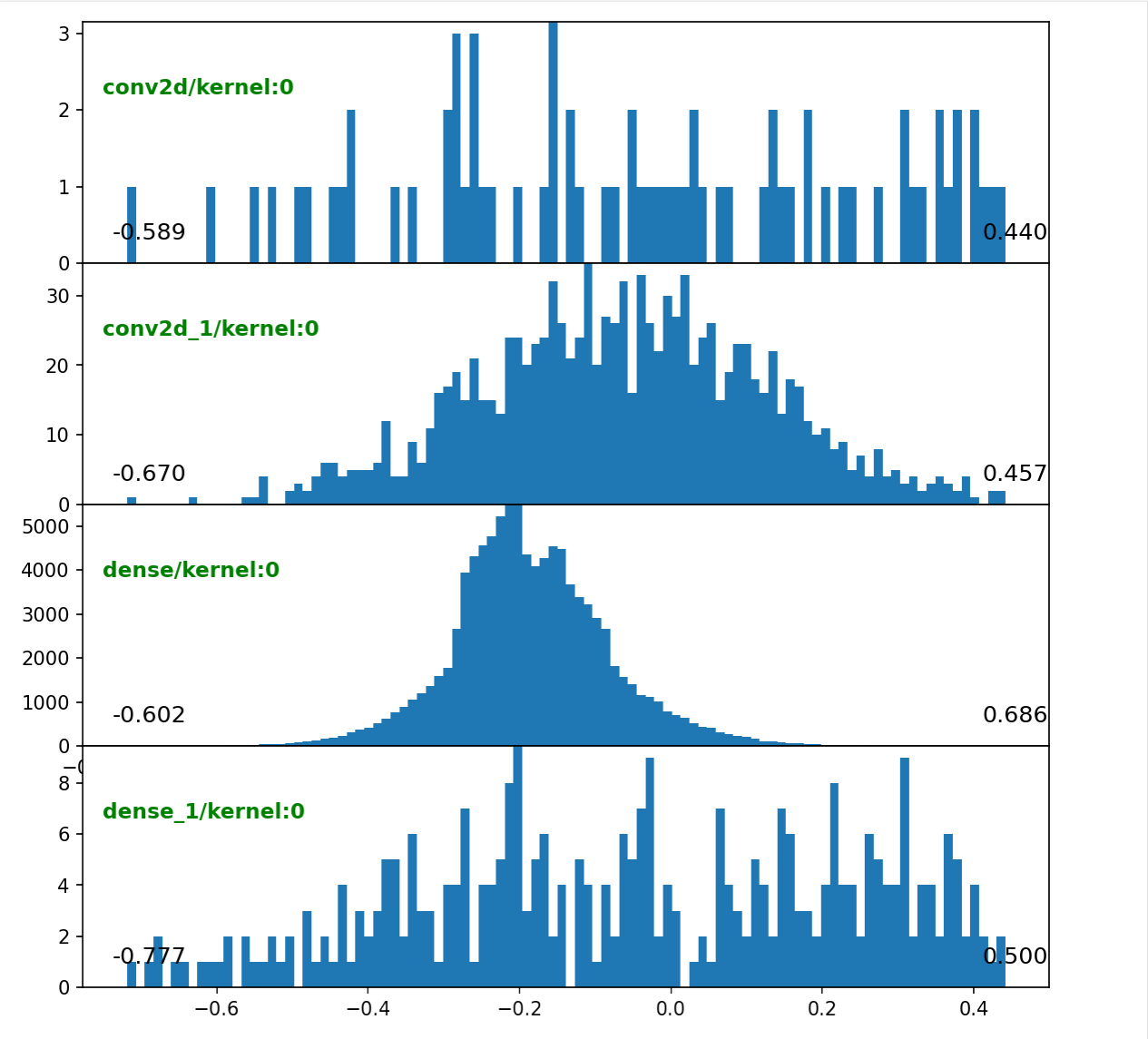

```
Validating...
TraingenLogger data saved to C:\deeptrain\examples\dir\logger\datalog_3.h5
TrainGenerator state saved
Model report generated and saved
Best model saved to C:\deeptrain\examples\dir\models\M2__model-Adam__min.141
TraingenLogger data saved to C:\deeptrain\examples\dir\logger\datalog_3.h5
TrainGenerator state saved
Model report generated and saved
```

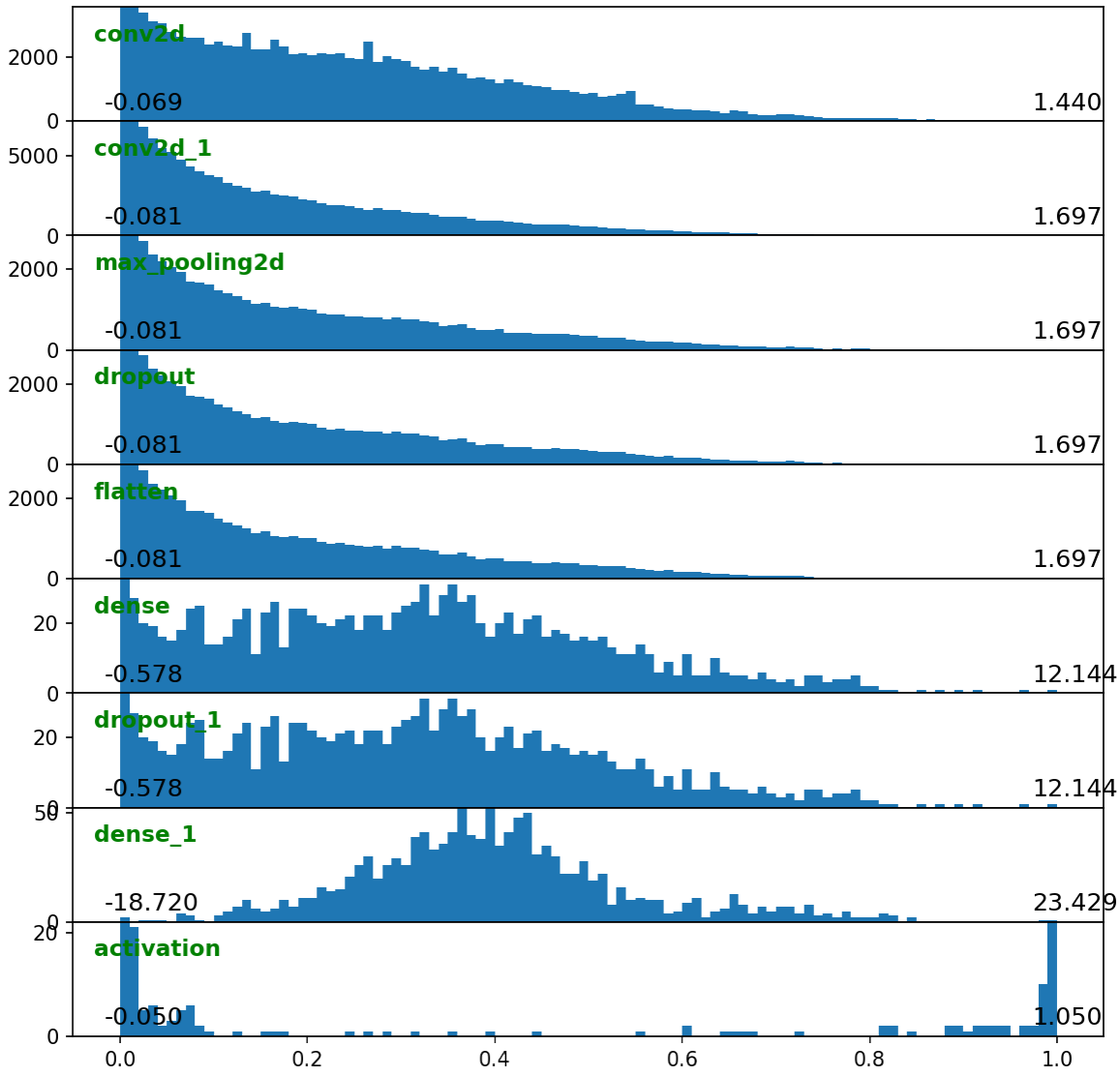

2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
20
40
60
80
100
120
140
Loss (train)
Acc (train)
Loss (val)
Acc (val)



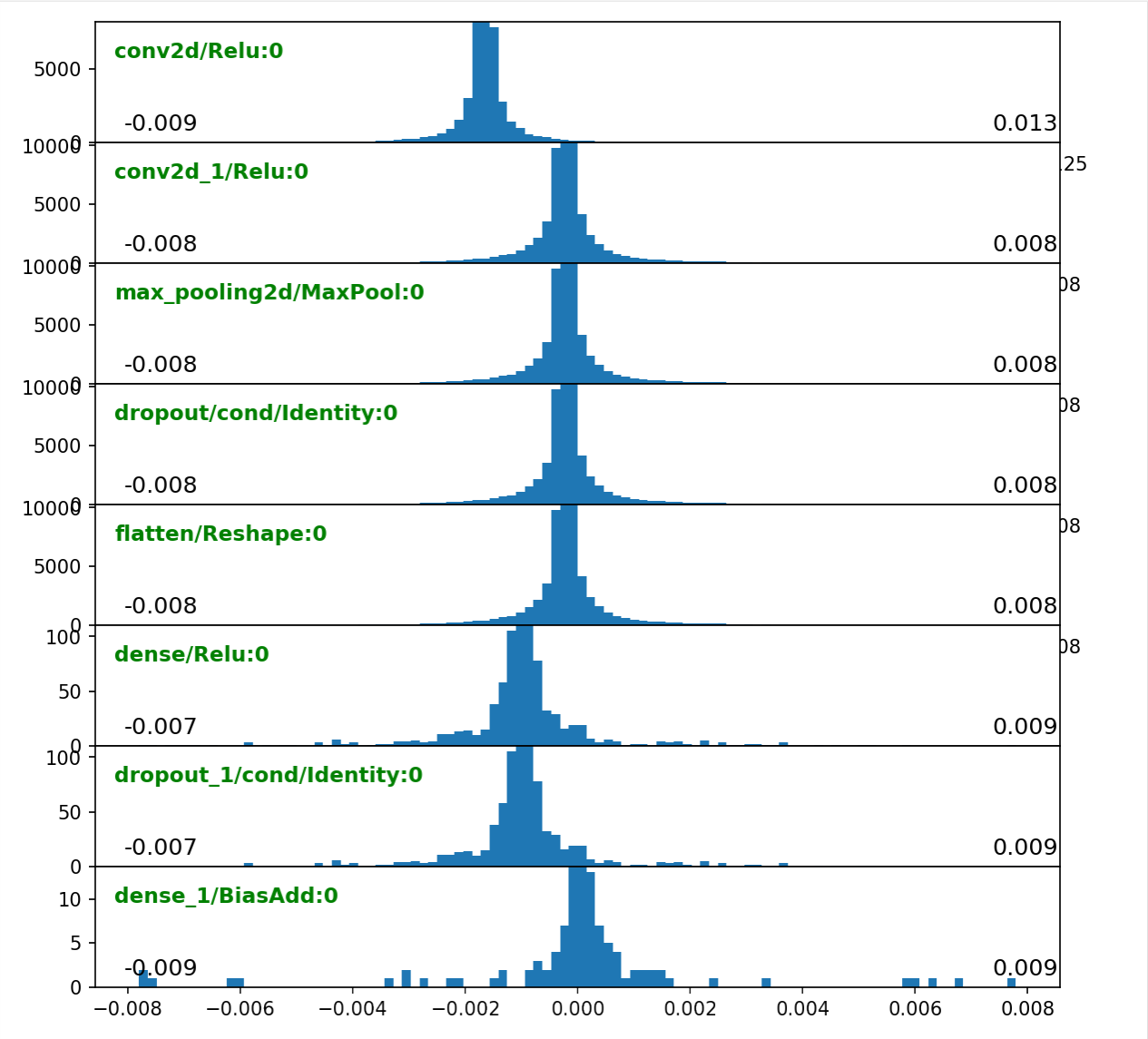

conv2d/Relu:0
-0.007
0.008
conv2d_1/Relu:0
-0.010
0.008
max_pooling2d/MaxPool:0
-0.010
0.008
dropout/cond/Identity:0
-0.010
0.008
flatten/Reshape:0
-0.010
0.008
dense/Relu:0
-0.006
0.006
dropout_1/cond/Identity:0
-0.006
0.006
dense_1/BiasAdd:0
-0.009
0.008
−0.008
−0.006
−0.004
−0.002
0.000
0.002
0.004
0.006
0.008

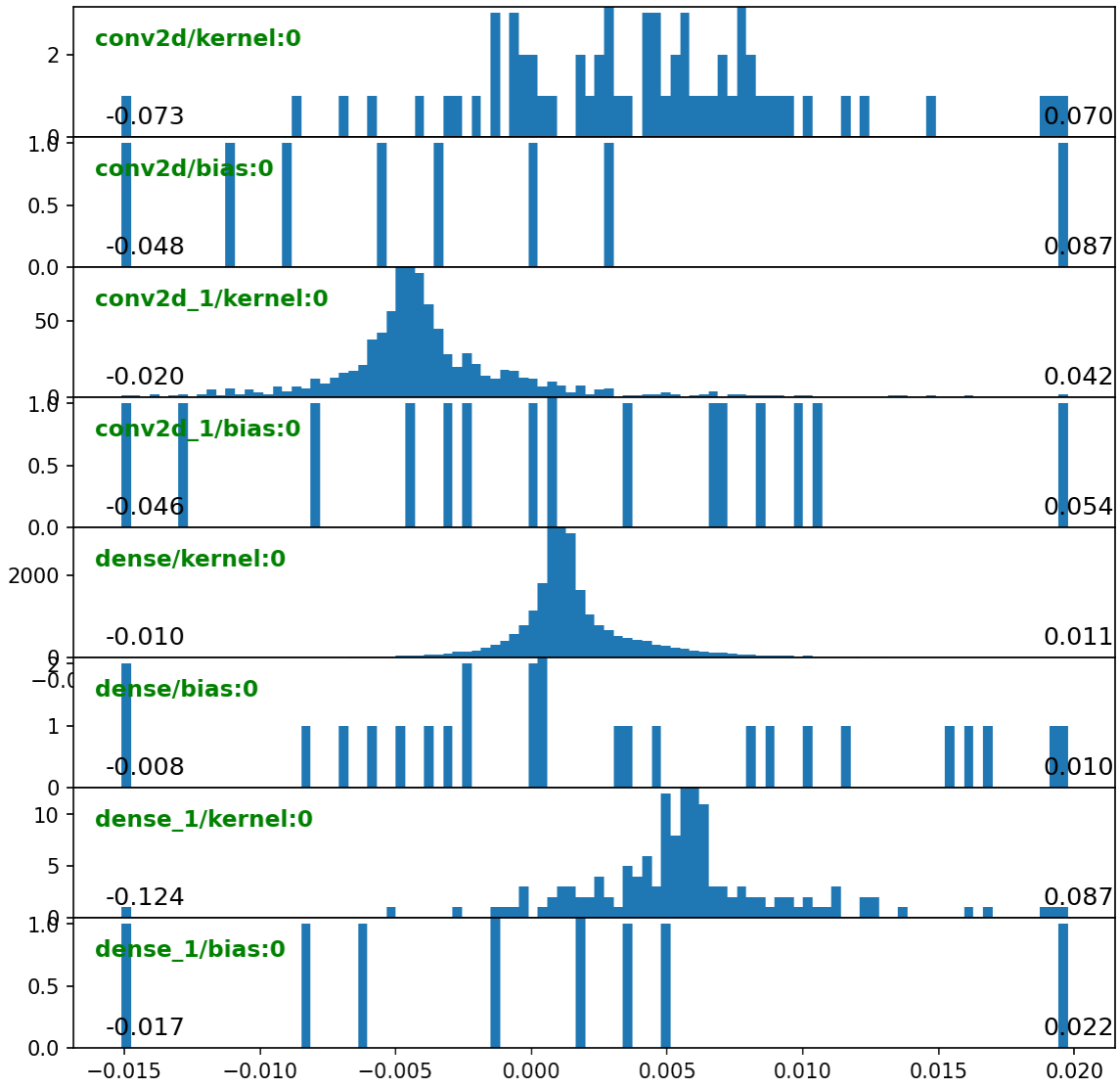

conv2d/kernel:0
-0.085
0.088
conv2d/bias:0
-0.062
0.105
conv2d_1/kernel:0
-0.033
0.043
conv2d_1/bias:0
-0.083
0.148
dense/kernel:0
-0.010
0.012
dense/bias:0
-0.009
0.014
dense_1/kernel:0
-0.090
0.107
dense_1/bias:0
-0.017
0.029
−0.01
0.00
0.01
0.02


conv2d/kernel:0
-0.589
0.440
conv2d_1/kernel:0
-0.670
0.457
dense/kernel:0
-0.602
0.686
dense_1/kernel:0
-0.777
0.500
−0.6
−0.4
−0.2
0.0
0.2
0.4

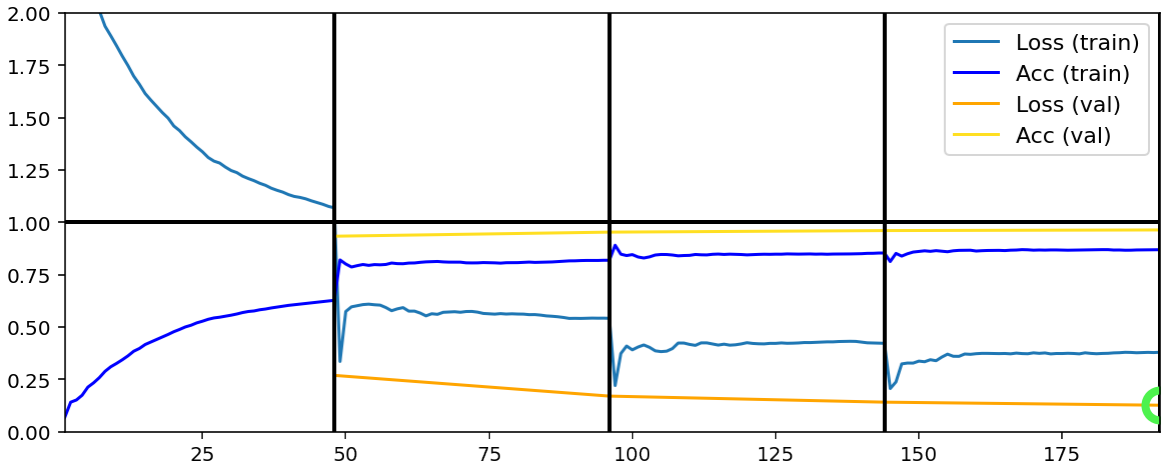


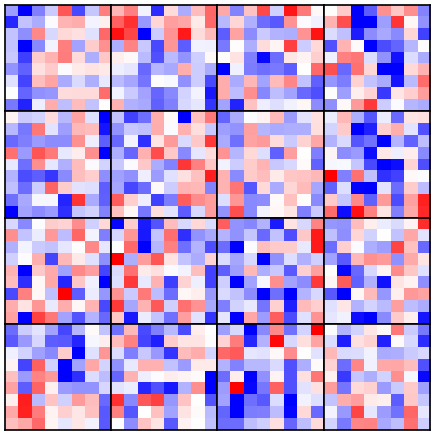

```
Data set_nums shuffled


 EPOCH 4 -- COMPLETE
```

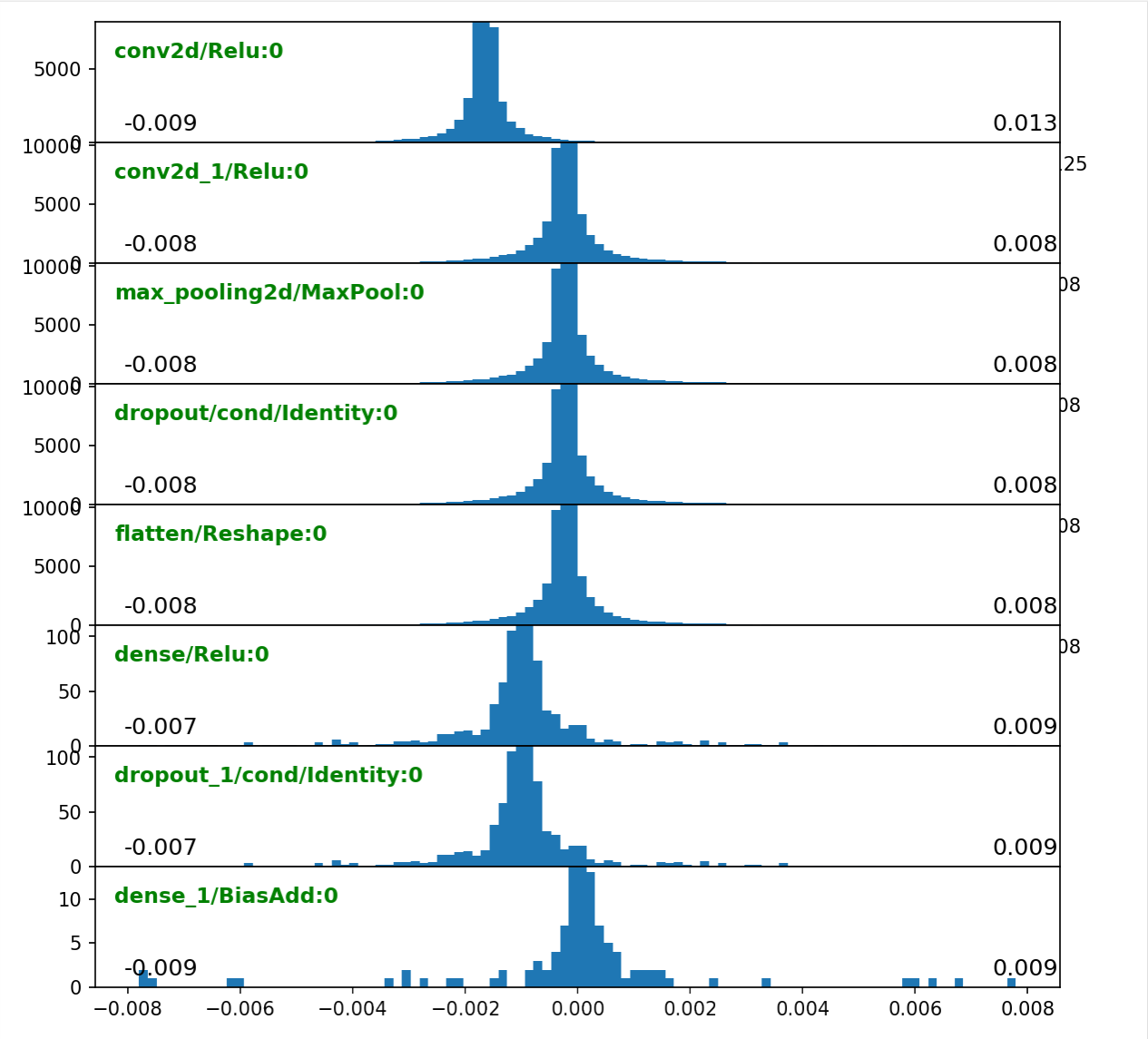

conv2d/Relu:0
-0.009
0.013
conv2d_1/Relu:0
-0.008
0.008
max_pooling2d/MaxPool:0
-0.008
0.008
dropout/cond/Identity:0
-0.008
0.008
flatten/Reshape:0
-0.008
0.008
dense/Relu:0
-0.007
0.009
dropout_1/cond/Identity:0
-0.007
0.009
dense_1/BiasAdd:0
-0.009
0.009
−0.008
−0.006
−0.004
−0.002
0.000
0.002
0.004
0.006
0.008

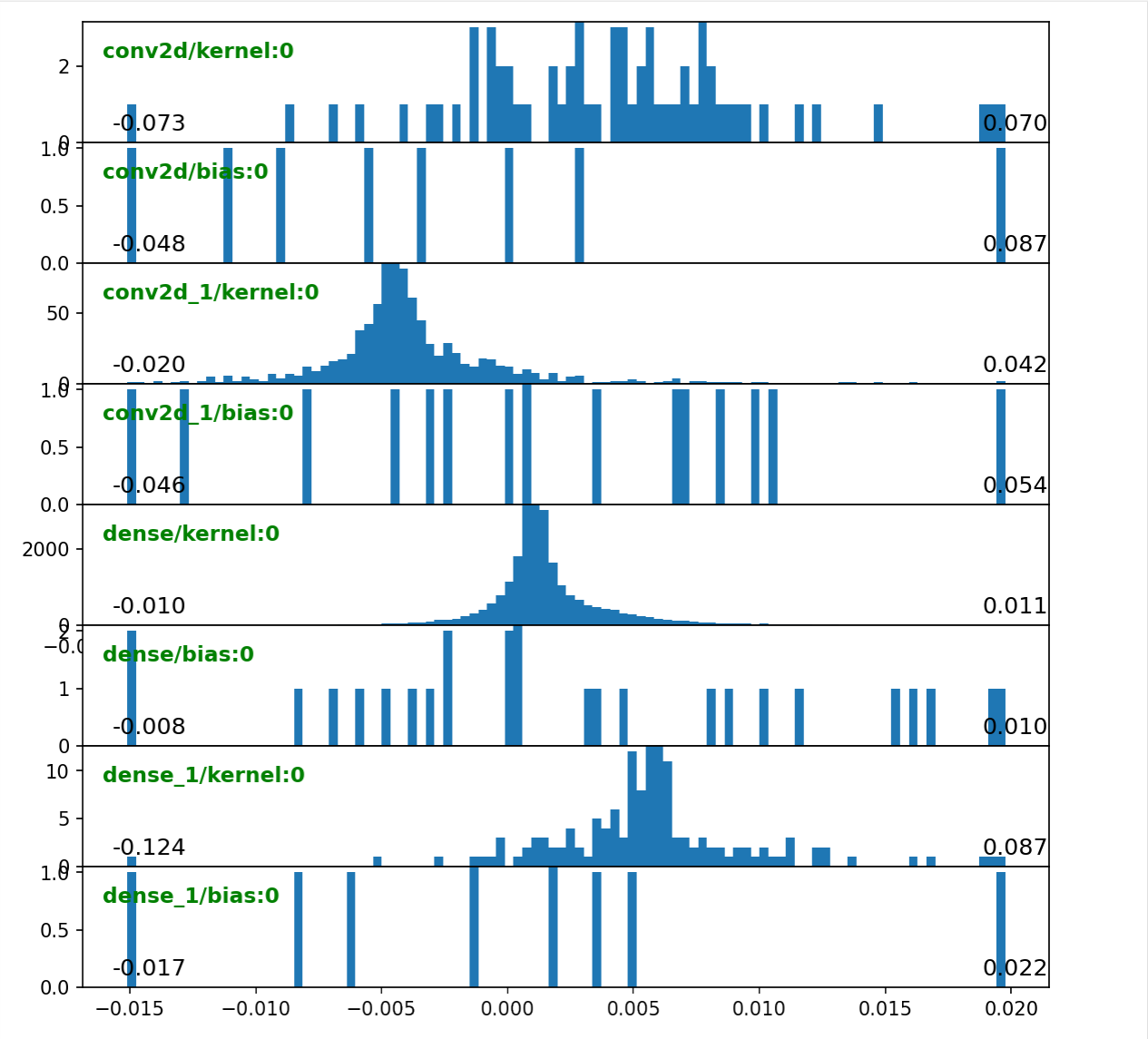


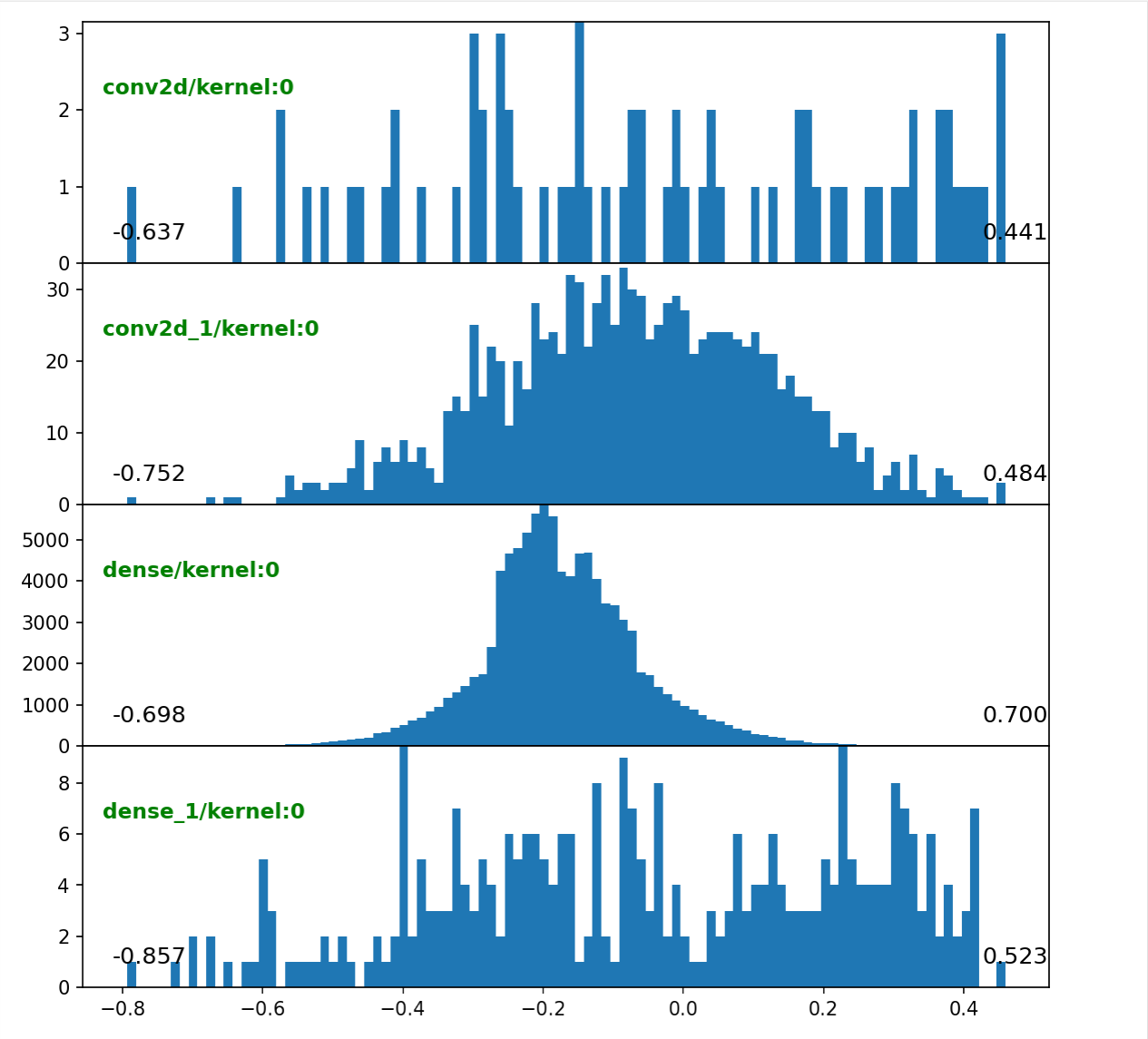

```
Validating...
TraingenLogger data saved to C:\deeptrain\examples\dir\logger\datalog_4.h5
TrainGenerator state saved
Model report generated and saved
Best model saved to C:\deeptrain\examples\dir\models\M2__model-Adam__min.126
TraingenLogger data saved to C:\deeptrain\examples\dir\logger\datalog_4.h5
TrainGenerator state saved
Model report generated and saved
```

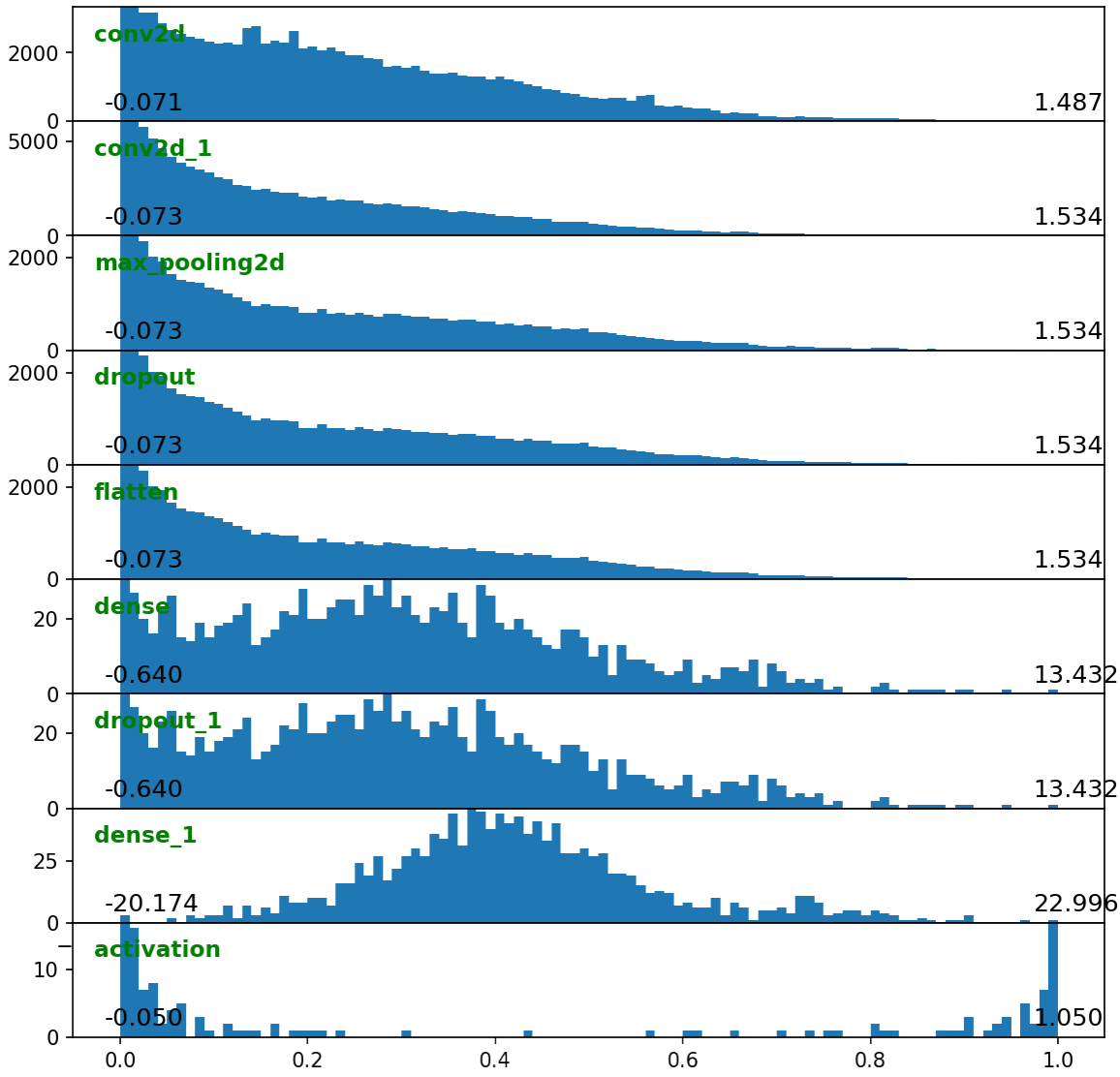

2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
25
50
75
100
125
150
175
Loss (train)
Acc (train)
Loss (val)
Acc (val)




conv2d/Relu:0
-0.009
0.013
conv2d_1/Relu:0
-0.008
0.008
max_pooling2d/MaxPool:0
-0.008
0.008
dropout/cond/Identity:0
-0.008
0.008
flatten/Reshape:0
-0.008
0.008
dense/Relu:0
-0.007
0.009
dropout_1/cond/Identity:0
-0.007
0.009
dense_1/BiasAdd:0
-0.009
0.009
−0.008
−0.006
−0.004
−0.002
0.000
0.002
0.004
0.006
0.008


conv2d/kernel:0
-0.073
0.070
conv2d/bias:0
-0.048
0.087
conv2d_1/kernel:0
-0.020
0.042
conv2d_1/bias:0
-0.046
0.054
dense/kernel:0
-0.010
0.011
dense/bias:0
-0.008
0.010
dense_1/kernel:0
-0.124
0.087
dense_1/bias:0
-0.017
0.022
−0.015
−0.010
−0.005
0.000
0.005
0.010
0.015
0.020


conv2d/kernel:0
-0.637
0.441
conv2d_1/kernel:0
-0.752
0.484
dense/kernel:0
-0.698
0.700
dense_1/kernel:0
-0.857
0.523
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4




```
Training has concluded.
```

We can thus track the progression of layer gradients & activations epoch-to-epoch.

`examples/callbacks/mnist.py` shows an additional callback; notebook shortened.

3.7 Introspection

3.7.1 Inspecting gradients

This example assumes you've read `advanced.ipynb`, and covers:

- Inspecting gradients per layer
- Estimating good values of gradient clipping threshold

[1]:
```
import deeptrain
deeptrain.util.misc.append_examples_dir_to_sys_path()

from utils import make_autoencoder, init_session
from utils import AE_CONFIGS as C

from tensorflow.keras.optimizers import Adam
import numpy as np
```

Configure training

[2]:
```
C['traingen']['iter_verbosity'] = 0  # silence iteration printing since currently␣
↪irrelevant
tg = init_session(C, make_autoencoder)
```

```
WARNING: multiple file extensions found in `path`; only .npy will be used
Discovered 48 files with matching format
48 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

WARNING: multiple file extensions found in `path`; only .npy will be used
Discovered 36 files with matching format
36 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

NOTE: will exclude `labels` from saving when `input_as_labels=True`; to keep 'labels',
↪ add '{labels}'to `saveskip_list` instead
Preloading superbatch ... WARNING: multiple file extensions found in `path`; only .
↪npy will be used
Discovered 48 files with matching format
... finished, w/ 6144 total samples
Train initial data prepared
Preloading superbatch ... WARNING: multiple file extensions found in `path`; only .
↪npy will be used
Discovered 36 files with matching format
... finished, w/ 4608 total samples
Val initial data prepared
Logging ON; directory (new): C:\deeptrain\examples\dir\logs\M3__model-nadam__min999.
↪000
```

Expected gradient norm estimation

We iterate over entire train dataset, gathering gradients from every fit and computing and storing their L2-norms.

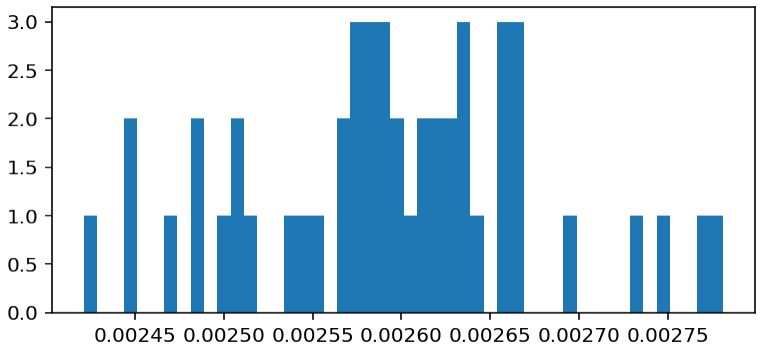
[3]:
```
grad_norms, *_ = tg.gradient_norm_over_dataset()
```

```
Computing gradient l2-norm over datagen batches, in inference mode
WARNING: datagen states will be reset
'.' = slice processed, '|' = batch processed

Data set_nums shuffled

||||||||||10||||||||||20||||||||||30||||||||||40||||||||
```

(continues on next page)

(continued from previous page)

```
Data set_nums shuffled


GRADIENT L2-NORM (AVG, MAX) = (0.003, 0.003), computed over 48 batches, 48 datagen␣
↪updates
```



We can now restart training with setting optimizer `clipnorm` to 1.5x average value, avoiding extreme gradients while not clipping most standard gradients

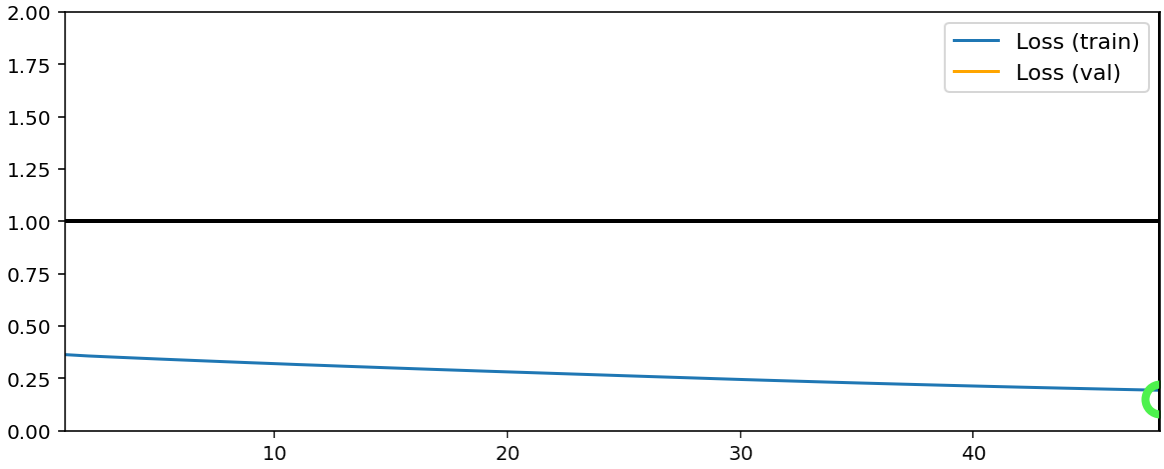
[4]:
```
C['model']['optimizer'] = Adam(clipnorm=1.5 * np.mean(grad_norms))
tg = init_session(C, make_autoencoder)
tg.epochs = 1  # train just for demo
tg.train()
```

```
WARNING: multiple file extensions found in `path`; only .npy will be used
Discovered 48 files with matching format
48 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

WARNING: multiple file extensions found in `path`; only .npy will be used
Discovered 36 files with matching format
36 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

NOTE: will exclude `labels` from saving when `input_as_labels=True`; to keep 'labels',
↪ add '{labels}'to `saveskip_list` instead
Preloading superbatch ... WARNING: multiple file extensions found in `path`; only .
↪npy will be used
Discovered 48 files with matching format
... finished, w/ 6144 total samples
Train initial data prepared
Preloading superbatch ... WARNING: multiple file extensions found in `path`; only .
↪npy will be used
Discovered 36 files with matching format
... finished, w/ 4608 total samples
Val initial data prepared
Logging ON; directory (new): C:\deeptrain\examples\dir\logs\M4__model-Adam__min999.000

Data set_nums shuffled
```

(continues on next page)

(continued from previous page)

```
 EPOCH 1 -- COMPLETE


Validating...
TrainGenerator state saved
Model report generated and saved
Best model saved to C:\deeptrain\examples\dir\models\M4__model-Adam__min.152
TrainGenerator state saved
Model report generated and saved
```



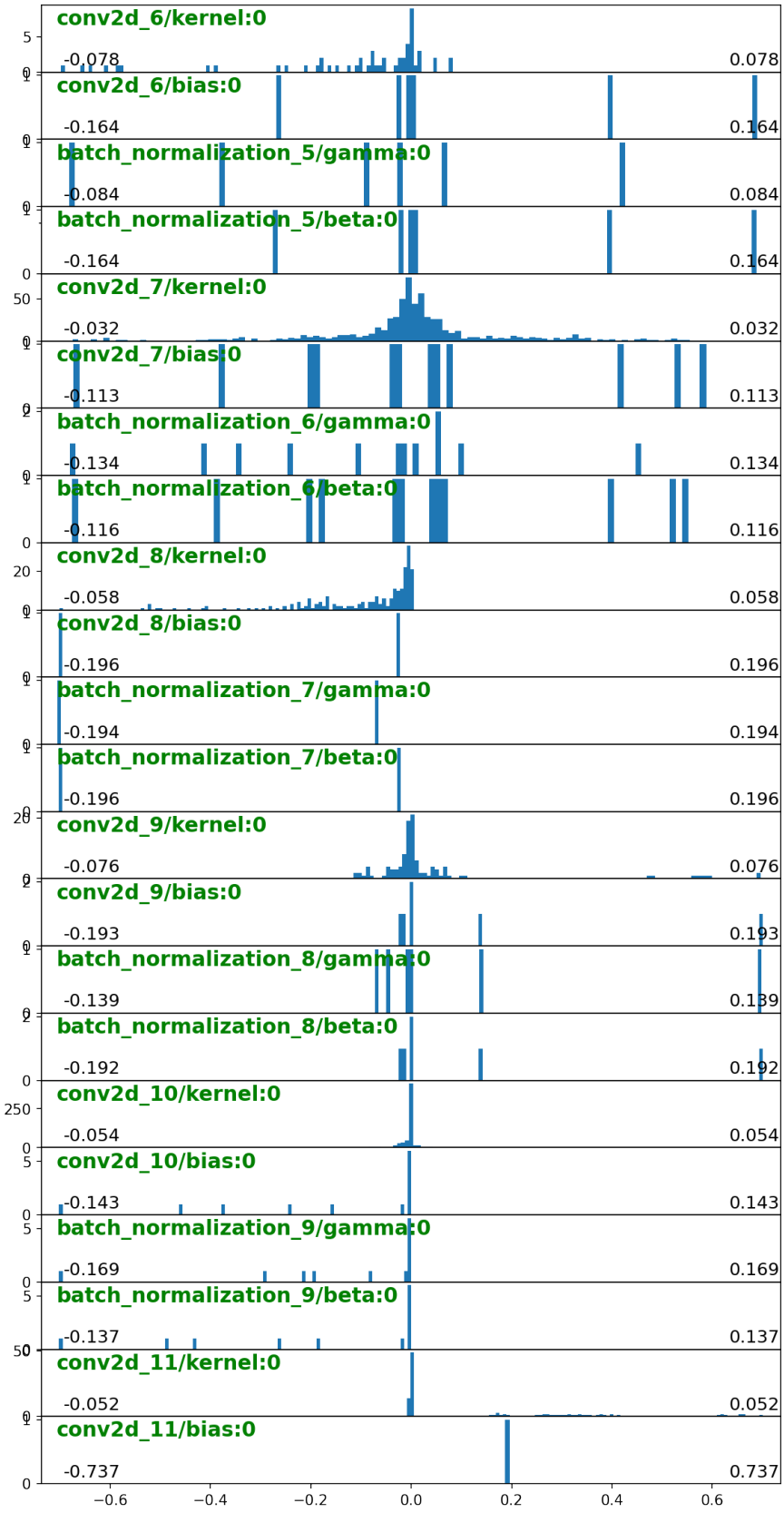

```
Training has concluded.
```

Complete gradient sum

This time we run a cumulative sum over actual gradient tensors, preserving and returning their shapes, allowing per-weight visualization

[5]:
```
plot_kw = {'h': 2}  # double default height since we expect many weights
grads_sum, *_ = tg.gradient_sum_over_dataset(plot_kw=plot_kw)
```

```
Computing gradients sum over datagen batches, in inference mode
WARNING: datagen states will be reset
'.' = slice processed, '|' = batch processed

Data set_nums shuffled

||||||||||10|||||||||20|||||||||30|||||||||40||||||||
Data set_nums shuffled


GRADIENTS SUM computed over 48 batches, 48 datagen updates:
```


conv2d_6/kernel:0
-0.078
0.078
conv2d_6/bias:0
-0.164
0.164
batch_normalization_5/gamma:0
-0.084
0.084
batch_normalization_5/beta:0
-0.164
0.164
conv2d_7/kernel:0
-0.032
0.032
conv2d_7/bias:0
-0.113
0.113
batch_normalization_6/gamma:0
-0.134
0.134
batch_normalization_6/beta:0
-0.116
0.116
conv2d_8/kernel:0
-0.058
0.058
conv2d_8/bias:0
-0.196
0.196
batch_normalization_7/gamma:0
-0.194
0.194
batch_normalization_7/beta:0
-0.196
0.196
conv2d_9/kernel:0
-0.076
0.076
conv2d_9/bias:0
-0.193
0.193
batch_normalization_8/gamma:0
-0.139
0.139
batch_normalization_8/beta:0
-0.192
0.192
conv2d_10/kernel:0
-0.054
0.054
conv2d_10/bias:0
-0.143
0.143
batch_normalization_9/gamma:0
-0.169
0.169
batch_normalization_9/beta:0
-0.137
0.137
conv2d_11/kernel:0
-0.052
0.052
conv2d_11/bias:0
-0.737
0.737
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6

We can use the mean of `grads_sum` to set `clipvalue` instead of `clipnorm`.

3.7.2 Inspecting internals

This example assumes you've read `advanced.ipynb`, and covers:

- Inspecting useful internal `TrainGenerator` & `DataGenerator` attributes
- Inspecting train / validation interruptions

[1]:
```
import deeptrain
deeptrain.util.misc.append_examples_dir_to_sys_path()  # for `from utils import`

from utils import make_autoencoder, init_session
from utils import AE_CONFIGS as C
```

Configure & train

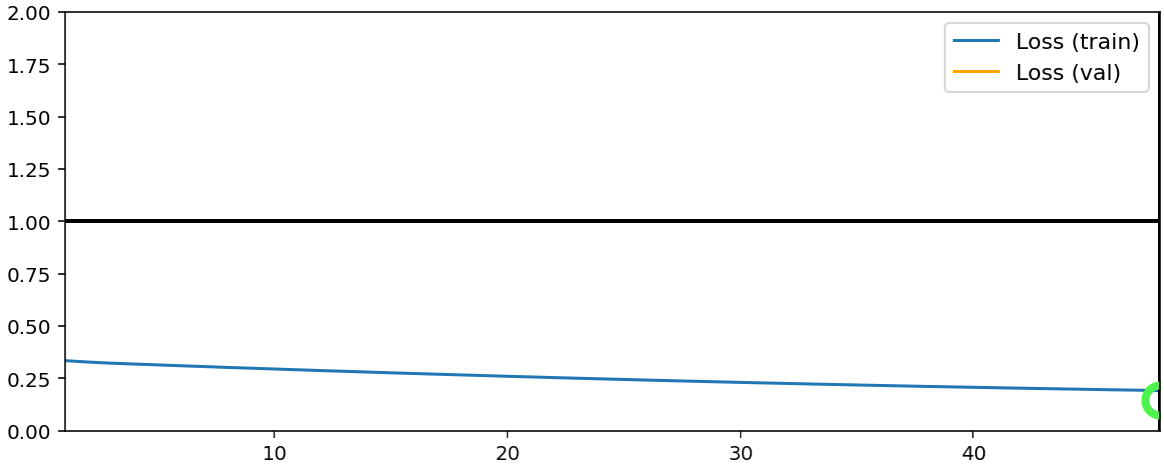
[2]:
```
C['traingen']['epochs'] = 1  # don't need more
C['traingen']['iter_verbosity'] = 0  # don't need progress printing here
tg = init_session(C, make_autoencoder)
dg = tg.datagen
vdg = tg.val_datagen

tg.train()
```

```
WARNING: multiple file extensions found in `path`; only .npy will be used
Discovered 48 files with matching format
48 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

WARNING: multiple file extensions found in `path`; only .npy will be used
Discovered 36 files with matching format
36 set nums inferred; if more are expected, ensure file names contain a common␣
↪substring w/ a number (e.g. 'train1.npy', 'train2.npy', etc)
DataGenerator initiated

NOTE: will exclude `labels` from saving when `input_as_labels=True`; to keep 'labels',
↪ add '{labels}'to `saveskip_list` instead
Preloading superbatch ... WARNING: multiple file extensions found in `path`; only .
↪npy will be used
Discovered 48 files with matching format
... finished, w/ 6144 total samples
Train initial data prepared
Preloading superbatch ... WARNING: multiple file extensions found in `path`; only .
↪npy will be used
Discovered 36 files with matching format
... finished, w/ 4608 total samples
Val initial data prepared
Logging ON; directory (new): C:\deeptrain\examples\dir\logs\M5__model-nadam__min999.
↪000

Data set_nums shuffled
```

(continues on next page)

(continued from previous page)

```
 EPOCH 1 -- COMPLETE


Validating...
TrainGenerator state saved
Model report generated and saved
Best model saved to C:\deeptrain\examples\dir\models\M5__model-nadam__min.144
TrainGenerator state saved
Model report generated and saved
```




```
Training has concluded.
```

Arguments passed to TrainGenerator

Can see the arguments passed at `__init__`; this is saved in the state file, useful for seeing how exactly training was instantiated. Some objects are stored as string to allow pickling

[3]:
```
from pprint import pprint
pprint(tg._passed_args)
```

```
{'best_models_dir': 'C:\\deeptrain\\examples\\dir\\models',
 'datagen': 'DataGenerator',
 'epochs': 1,
 'eval_fn': 'predict',
 'input_as_labels': True,
 'iter_verbosity': 0,
 'logs_dir': 'C:\\deeptrain\\examples\\dir\\logs',
 'max_is_best': False,
 'model': 'Functional',
 'model_configs': {'activation': ['relu', 'relu', 'relu', 'relu', 'relu'],
                   'batch_shape': (128, 28, 28, 1),
                   'filters': [6, 12, 2, 6, 12],
                   'input_dropout': 0.5,
                   'kernel_size': [(3, 3), (3, 3), (3, 3), (3, 3), (3, 3)],
                   'loss': 'mse',
                   'metrics': None,
                   'optimizer': 'nadam',
```

(continues on next page)

(continued from previous page)

```
                    'preout_dropout': 0.4,
                    'strides': [(2, 2), (2, 2), 1, 1, 1],
                    'up_sampling_2d': [None, None, None, (2, 2), (2, 2)]},
 'plot_configs': {'0': {'legend_kw': {'fontsize': 11}}},
 'val_datagen': 'DataGenerator'}
```

Code used in training & initial attributes

- `TrainGenerator`'s attributes at end of `__init__` are logged at end of `TrainGenerator.__init__`
  - savepath: `logdir/misc/init_state.json`
- Source code used to run training (`__main__`) is also logged, assuming ran as a `.py` file (not IPython excerpt or Jupyter notebook)
  - savepath: `logdir/misc/init_script.txt`

[4]:
```
import json
with open(tg.get_last_log('init_state'), 'r') as f:
    j = json.load(f)
    pprint(j)
```

```
{'_batches_fit': '0',
 '_batches_validated': '0',
 '_class_labels_cache': '[]',
 '_epoch': '0',
 '_eval_fn': 'fn',
 '_eval_fn_name': 'predict',
 '_fit_fn': 'fn',
 '_fit_fn_name': 'train_on_batch',
 '_fit_iters': '0',
 '_hist_vlines': '[]',
 '_history_fig': 'None',
 '_imports': "{'PIL': 1, 'LZ4F': 1}",
 '_inferred_batch_size': 'None',
 '_init_callbacks_called': 'True',
 '_labels': '[]',
 '_labels_cache': '[]',
 '_max_set_name_chars': '3',
 '_passed_args': 'dict',
 '_preds_cache': '[]',
 '_save_from_on_val_end': 'False',
 '_set_name': '1',
 '_set_name_cache': '[]',
 '_set_num': '1',
 '_sw_cache': '[]',
 '_temp_history_empty': "{'loss': []}",
 '_times_validated': '0',
 '_train_loop_done': 'False',
 '_train_new_batch_notified': 'False',
 '_train_postiter_processed': 'True',
 '_train_val_x_ticks': '[]',
 '_train_x_ticks': '[]',
 '_val_epoch': '0',
 '_val_hist_vlines': '[]',
 '_val_iters': '0',
 '_val_loop_done': 'False',
```

(continues on next page)

(continued from previous page)


```
'_val_max_set_name_chars': '2',
'_val_new_batch_notified': 'False',
'_val_postiter_processed': 'True',
'_val_set_name': '1',
'_val_set_name_cache': '[]',
'_val_set_num': '1',
'_val_temp_history_empty': "{'loss': []}",
'_val_train_x_ticks': '[]',
'_val_x_ticks': '[]',
'alias_to_metric': "{'acc': 'accuracy', 'mae': 'mean_absolute_error', 'mse': "
                   "'mean_squared_error', 'mape': "
                   "'mean_absolute_percentage_error', 'msle': "
                   "'mean_squared_logarithmic_error', 'kld': "
                   "'kullback_leibler_divergence', 'cosine': "
                   "'cosine_similarity', 'f1': 'f1_score', 'f1-score': "
                   "'f1_score'}",
'batch_size': '128',
'best_key_metric': '999',
'best_models_dir': 'C:\\deeptrain\\examples\\dir\\models',
'best_subset_nums': '[]',
'best_subset_size': 'None',
'callbacks': '[]',
'check_model_health': 'True',
'checkpoints_overwrite_duplicates': 'True',
'class_weights': 'None',
'custom_metrics': '{}',
'datagen': 'deeptrain.data_generator.DataGenerator',
'dynamic_predict_threshold': '0.5',
'dynamic_predict_threshold_min_max': 'None',
'epochs': '1',
'final_fig_dir': 'None',
'history': "{'loss': []}",
'input_as_labels': 'True',
'iter_verbosity': '0',
'key_metric': 'loss',
'key_metric_fn': 'mean_squared_error',
'key_metric_history': '[]',
'loadpath': 'None',
'loadskip_list': "['{auto}', 'model_name', 'model_base_name', 'model_num', "
                 "'use_passed_dirs_over_loaded', 'logdir', "
                 "'_init_callbacks_called']",
'logdir': 'C:\\deeptrain\\examples\\dir\\logs\\M5__model-nadam__min999.000',
'logs_dir': 'C:\\deeptrain\\examples\\dir\\logs',
'logs_use_full_model_name': 'True',
'loss_weighted_slices_range': 'None',
'max_checkpoints': '5',
'max_is_best': 'False',
'max_one_best_save': 'True',
'metric_printskip_configs': "{'train': [], 'val': []}",
'metric_to_alias': "{'loss': 'Loss', 'accuracy': 'Acc', 'f1_score': 'F1', "
                   "'tnr': '0-Acc', 'tpr': '1-Acc', 'mean_absolute_error': "
                   "'MAE', 'mean_squared_error': 'MSE'}",
'model': 'tensorflow.python.keras.engine.functional.Functional',
'model_base_name': 'model',
'model_configs': "{'batch_shape': (128, 28, 28, 1), 'loss': 'mse', 'metrics': "
                 "None, 'optimizer': 'nadam', 'activation': ['relu', 'relu', "
                 "'relu', 'relu', 'relu'], 'filters': [6, 12, 2, 6, 12], "
```


(continues on next page)

(continued from previous page)

```
                  "'kernel_size': [(3, 3), (3, 3), (3, 3), (3, 3), (3, 3)], "
                  "'strides': [(2, 2), (2, 2), 1, 1, 1], 'up_sampling_2d': "
                  "[None, None, None, (2, 2), (2, 2)], 'input_dropout': 0.5, "
                  "'preout_dropout': 0.4}",
 'model_name': 'M5__model-nadam__min999.000',
 'model_name_configs': "{'optimizer': '', 'lr': '', 'best_key_metric': "
                       "'__min'}",
 'model_num': '5',
 'model_save_kw': "{'include_optimizer': True, 'save_format': 'h5'}",
 'model_save_weights_kw': "{'save_format': 'h5'}",
 'name_process_key_fn': 'NAME_PROCESS_KEY_FN',
 'new_model_num': 'True',
 'optimizer_load_configs': 'None',
 'optimizer_save_configs': 'None',
 'plot_configs': "{'0': {'legend_kw': {'fontsize': 11}, 'metrics': {'train': "
                 "['loss'], 'val': ['loss']}, 'x_ticks': {'train': "
                 "['_train_x_ticks'], 'val': ['_val_train_x_ticks']}, "
                 "'vhlines': {'v': '_hist_vlines', 'h': 1}, 'mark_best_cfg': "
                 "{'val': 'loss', 'max_is_best': False}, 'ylims': (0, 2), "
                 "'linewidth': [1.5, 1.5], 'linestyle': ['-', '-'], 'color': "
                 "['#1f77b4', 'orange']}, 'fig_kw': {'figsize': (12, 7)}}",
 'plot_first_pane_max_vals': '2',
 'plot_history_freq': "{'epoch': 1}",
 'pred_weighted_slices_range': 'None',
 'predict_threshold': '0.5',
 'report_configs': 'dict',
 'report_fontpath': 'C:\\deeptrain\\deeptrain\\util\\fonts\\consola.ttf',
 'reset_statefuls': 'False',
 'saveskip_list': "['model', 'optimizer_state', 'callbacks', 'key_metric_fn', "
                  "'custom_metrics', 'metric_to_alias', 'alias_to_metric', "
                  "'name_process_key_fn', '_fit_fn', '_eval_fn', '_labels', "
                  "'_preds', '_y_true', '_y_preds', '_labels_cache', "
                  "'_preds_cache', '_sw_cache', '_imports', '_history_fig', "
                  "'_val_max_set_name_chars', '_max_set_name_chars', "
                  "'_inferred_batch_size', '_class_labels_cache', "
                  "'_temp_history_empty', '_val_temp_history_empty', "
                  "'_val_sw', '_set_num', '_val_set_num', 'labels']",
 'temp_checkpoint_freq': 'None',
 'temp_history': "{'loss': []}",
 'train_metrics': "['loss']",
 'unique_checkpoint_freq': "{'epoch': 1}",
 'val_class_weights': 'None',
 'val_datagen': 'deeptrain.data_generator.DataGenerator',
 'val_freq': "{'epoch': 1}",
 'val_history': "{'loss': []}",
 'val_metrics': "['loss']",
 'val_temp_history': "{'loss': []}"}
```

Save directories

[5]:
```
print("Best model directory:", tg.best_models_dir)
print("Checkpoint directory:", tg.logdir)
print("Model full name:", tg.model_name)
```

```
Best model directory: C:\deeptrain\examples\dir\models
```

(continues on next page)

(continued from previous page)

```
Checkpoint directory: C:\deeptrain\examples\dir\logs\M5__model-nadam__min999.000
Model full name: M5__model-nadam__min.144
```

Interrupts

Interrupts can be inspected by checking pertinent attributes manually (`_train_loop_done`, `_train_postiter_processed`, `_val_loop_done`, `_val_postiter_processed`), or calling `interrupt_status()` which checks these and prints an appropriate message.

[6]:
```
tg.interrupt_status()
```

```
No interrupts detected.

Flags checked:
        _train_loop_done          = False
        _train_postiter_processed = True
        _val_loop_done            = False
        _val_postiter_processed   = True
```

[6]:
```
(False, False)
```

Interrupts can be manual (`KeyboardInterrupt`) or due to a `raise Exception`; either interrupts the flow of train/validation, so knowing at which point the fault occurred allows us to correct manually (e.g. execute portion of code after an exception)

Interrupt example

[7]:
```
tg._train_loop_done = True
tg._val_loop_done = True
tg._val_postiter_processed = True
```

At this point `_on_val_end()` is called automatically, so if you're able to access such a state, it means the call didn't finish or was never initiated.

[8]:
```
tg.interrupt_status()
```

```
Incomplete or not called `_on_val_end()` within `validate()`.
Interrupted: train[no], validation[yes].

Flags checked:
        _train_loop_done          = True
        _train_postiter_processed = True
        _val_loop_done            = True
        _val_postiter_processed   = True
```

[8]:
```
(False, True)
```

Example 2

[9]:
```
tg._val_loop_done = False
tg._val_postiter_processed = False
tg.interrupt_status()
```

```
Interrupted during validation loop within `validate()`; incomplete or not called `_
↪val_postiter_processing()`.
Interrupted: train[no], validation[yes].

Flags checked:
        _train_loop_done          = True
        _train_postiter_processed = True
        _val_loop_done            = False
        _val_postiter_processed   = False
```

[9]:
```
(False, True)
```

[10]:
```
help(tg.train)
```

```
Help on method train in module deeptrain.train_generator:

train() method of deeptrain.train_generator.TrainGenerator instance
    The train loop.

        - Fetches data from `get_data`
        - Fits data via `fin_fn`
        - Processes fit metrics in `_train_postiter_processing`
        - Stores metrics in `history`
        - Applies `'train:iter'`, `'train:batch'`, and `'train:epoch'`
          callbacks
        - Calls `validate` when appropriate

    **Interruption**:

        - *Safe*: during `get_data`, which can be called indefinitely
          without changing any attributes.
        - *Avoid*: during `_train_postiter_processing`, where `fit_fn` is
          applied and weights are updated - but metrics aren't stored, and
          `_train_postiter_processed=False`, restarting the loop without
          recording progress.
        - Best bet is during :meth:`validate`, as `get_data` may be too brief.
```

[11]:
```
help(tg.validate)
```

```
Help on method validate in module deeptrain.train_generator:

validate(record_progress=True, clear_cache=True, restart=False, use_callbacks=True)␣
↪method of deeptrain.train_generator.TrainGenerator instance
    Validation loop.

        - Fetches data from `get_data`
        - Applies function based on `_eval_fn_name`
        - Processes and caches metrics/predictions in
          `_val_postiter_processing`
        - Applies `'val:iter'`, `'val:batch'`, and `'val:epoch'` callbacks
        - Calls `_on_val_end` at end of validation to compute metrics
          and store them in `val_history`
        - Applies `'val_end'` and maybe `('val_end': 'train:epoch')` callbacks
        - If `restart`, calls :meth:`reset_validation`.

    **Arguments**:
        record_progress: bool
```

(continues on next page)

(continued from previous page)

```
            If False, won't update `val_history`, `_val_iters`,
            `_batches_validated`.
        clear_cache: bool
            If False, won't call :meth:`clear_cache`; useful for keeping
            preds & labels acquired during validation.
        restart: bool
            If True, will call :meth:`reset_valiation` before validation loop
            to reset validation attributes; useful for starting afresh (e.g.
            if interrupted).
        use_callbacks: bool
            If False, won't call :meth:`apply_callbacks`
            or :meth:`plot_history`.

    **Interruption:**

        - *Safe*: during `get_data`, which can be called indefinitely
          without changing any attributes.
        - *Avoid*: during `_val_postiter_processing`. Model remains
          unaffected*, but caches are updated; a restart may yield duplicate
          appending, which will error or yield inaccuracies.
          (* forward pass may consume random seed if random ops are used)
        - *In practice*: prefer interrupting immediately after
          `_print_iter_progress` executes.
```

Interrupts can also be inspected by checking `temp_history`, `val_temp_history`, and cache attributes (e.g. `_preds_cache`); cache attributes clear by default when `validate()` finishes. Check `help(train)` and `help(validate)` for further interrupt guidelines.

DataGenerator attributes

`set_nums_to_process` are the set nums remaining until end of epoch, which are then reset to `set_nums_original`. "Set" refers to data file to load.

[12]:
```
# We can check which set numbers remain to be processed in epoch or validation:
print(dg.set_nums_to_process)
print(vdg.set_nums_to_process)
# We can arbitrarily append to or pop from the list to skip or repeat a batch
```

```
['42', '38', '34', '20', '25', '41', '14', '33', '30', '5', '19', '32', '11', '28',
↪'46', '40', '27', '24', '2', '21', '9', '17', '1', '29', '43', '26', '23', '36', '7
↪', '6', '48', '4', '39', '13', '12', '37', '45', '18', '44', '35', '10', '31', '22',
↪ '47', '8', '16', '15', '3']
['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16
↪', '17', '18', '19', '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30
↪', '31', '32', '33', '34', '35', '36']
```

Info function

Lastly, we can access most of the above via `info()`:

[13]:
```
tg.info()
```

```
Epochs: 1/1
Train batches fit: 0/48 (in current epoch)
Val   batches fit: 0/36 (in current validation)
--------------------------------------------------------------------------------
Best model directory: C:\deeptrain\examples\dir\models
Checkpoint directory: C:\deeptrain\examples\dir\logs\M5__model-nadam__min999.000
Load path: None
Model full name: M5__model-nadam__min.144
--------------------------------------------------------------------------------
Interrupted during validation loop within `validate()`; incomplete or not called `_
↪val_postiter_processing()`.
Interrupted: train[no], validation[yes].

Flags checked:
        _train_loop_done          = True
        _train_postiter_processed = True
        _val_loop_done            = False
        _val_postiter_processed   = False
```

3.8 How to . . . ?

3.8.1 Change default configs

Edit `deeptrain.util.configs`.

- Do **not** edit `deeptrain.util._default_configs`, this will break `DeepTrain`.
- Arguments defined in `TrainGenerator.__init__()` will override those specified in the configs (the defaults have no overlaps), so no point in specifying them in `configs`.

3.8.2 Run examples

`pip install deeptrain` excludes data by default. To acquire, you can:

1. Build data by running scripts in `examples/preprocessing`. Or,
2. Clone repository and copy-paste `examples/dir/data` into the pip-installed `deeptrain` directory.

With the data you can run the `.ipynb` with Jupyter or equivalent `.py` scripts in IPython. Note for docs notebook examples, code isn't exact, and excludes some formatting irrelevant the examples (e.g. many used `os.environ['SCALEFIG'] = '.7'`).

3.8.3 Save train state

1. Using `TrainGenerator.save()`, which saves:
   - `TrainGenerator` attributes
   - DataGenerator' (both) attributes
   - Model state (layer weights, optimizer weights, and/or architecture)
2. Using `TrainGenerator.checkpoint()`, which saves what `.save()` saves, plus:
   - `TrainGenerator` report, made by `logging.generate_report()`

- Train & val history figure

3. Saving behavior is configured for objects by respective attributes (defaults in `_default_configs`):
   - `TrainGenerator`: `saveskip_list`
   - `DataGenerator` (for each): `saveskip_list`
   - `model`: `model_save_kw`, `model_save_weights_kw`, `optimizer_save_configs`
   - `Preprocessor` (of each `DataGenerator`): `saveskip_list`

Example in *Deeper into DeepTrain*.

3.8.4 Load train state

1. Using `TrainGenerator.load()`, which may load everything saved via `TrainGenerator.save()` and `TrainGenerator.checkpoint()`.
2. Loading behavior is configured for objects by respective attributes (defaults in `_default_configs`):
   - `TrainGenerator`: `loadskip_list`
   - `DataGenerator` (for each): `loadskip_list`
   - `model`: `optimizer_load_configs`
   - `Preprocessor` (of each `DataGenerator`): `loadskip_list`

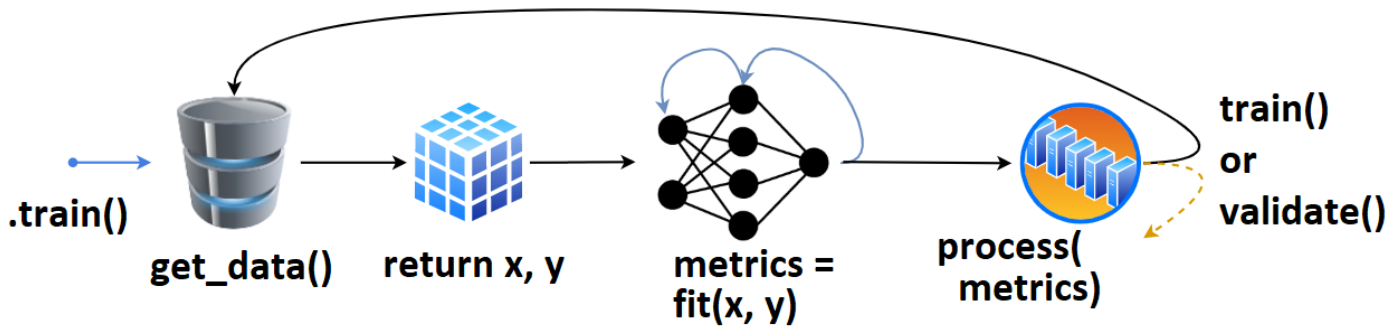
Example in *Deeper into DeepTrain*.

3.8.5 Use custom train / evaluation function

Set `fit_fn` / `eval_fn`; see docs in `TrainGenerator()`.

3.9 How does . . . work?

3.9.1 TrainGenerator

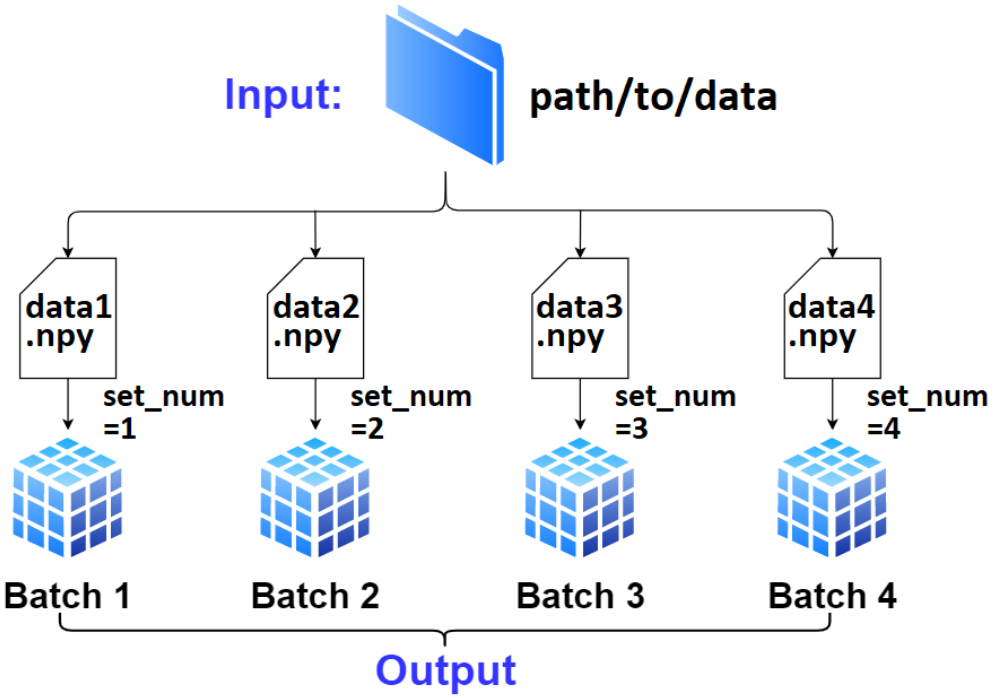


1. User defines `tg = TrainGenerator(**configs)`,
2. calls `tg.train()`.
3. `get_data()` is called, returning data & labels,

4. fed to `model.fit()`, returning `metrics`,
5. which are then printed, recorded.
6. The loop repeats, or `validate()` is called.

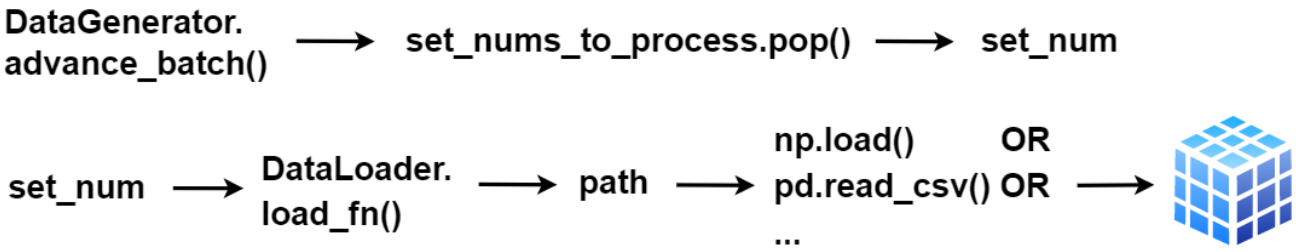
Once `validate()` finishes, training may checkpoint, and `train()` is called again. That's the (simlpified) high-level overview. Callbacks and other behavior can be configured for every stage of training.

3.9.2 DataGenerator



1. User defines `dg = DataGenerator(**configs)`.
2. If not specified, `dg` infers the number of batches, file extension, data loader, and other necessary info solely from `data_path` / `labels_path`; this is "**AutoData**".
    - Only required is proper file naming; there's to be a "common" off of which `dg` can enlist `set_nums`, which is how it tracks all data internally.
    - Exception to above is if the path is to a single file containing all data; see `DataGenerator()`.
3. Data (`x`) and labels (`y`) can be fetched with `DataGenerator.get()`; by default it'll validate the batch and reset necessary attributes in case data "runs out" - to prevent this, pass `skip_validation=True`.
4. To move on to next batch (which `.get()` won't do automatically), call `DataGenerator.advance_batch()`.
5. The getting, advancing, and resetting is handled automatically within `TrainGenerator.train()` and `TrainGenerator.validate()` at various stages.

3.9.3 DataLoader



- `DataGenerator()` is a "middle-man" between `TrainGenerator()` and the data, orchestrating *which* data is fetched at a point in training.
- The actual loading is handled by `DataLoader()`, with the customizable `DataLoader.load_fn()`.

3.10 API Reference

- genindex
- modindex
- search