

Where do I put `batch_normalized = tf.layers.batch_normalization(convolved, training=(mode == 'train'))`?

To add batch normalization to your CNN layers, you need to modify the CNN section of the `OnsetNet_Gina1` class. Here's exactly where to put the batch normalization code:

python

In the CNN section of OnsetNet_Gina1.__init__, find this code:

```
convolved = tf.nn.relu(biased)
```

Right after that line, add batch normalization:

```
batch_normalized = tf.layers.batch_normalization(convolved, training=(mode == 'train'))
```

Then change the pooling to use batch_normalized instead of convolved:

```
pooled = tf.nn.max_pool(batch_normalized, ksize=pool_shape, strides=pool_shape, padding='SAME')
```

The full context would look like this:

python

Existing code:

```
conv = tf.nn.conv2d(layer_last, filters, [1, 1, 1, 1], padding=padding)
```

```
biased = tf.nn.bias_add(conv, biases)
```

```
convolved = tf.nn.relu(biased)
```

Add this line:

```
batch_normalized = tf.layers.batch_normalization(convolved, training=(mode == 'train'))
```

Change this line (replace convolved with batch_normalized):

```
pooled = tf.nn.max_pool(batch_normalized, ksize=pool_shape, strides=pool_shape, padding='SAME')
```

Also, if you're using batch normalization, you'll need to ensure your training operation includes the update operations for the batch normalization statistics. Add this near the end of the **init** method, just before creating the training operation:

python

```
# Add this before the train_op = optimizer.apply_gradients(...) line:
update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
with tf.control_dependencies(update_ops):
    train_op = optimizer.apply_gradients(list(zip(grads, tvars)), global_step=tf.train.get_or_create_global_step())
```

This ensures that the moving averages used by batch normalization are properly updated during training.

Would you like me to show the full CNN section with all the batch normalization changes integrated?

```

-----
-
AttributeError                                Traceback (most recent call
last)
Cell In[3], line 52
    49 # Update the training_args argument container in place
    50 vars(training_args).update(training_args_dict)
--> 52 train_onset_model(training_args, OnsetModel=OnsetNet_Gina2)
    55 # Results 4/21 - Hard, Challenge
    56
    57 # Completed epoch 1
    (...)
    61 # Completed epoch 10
    62 # Epoch mean cross-entropy (nats)

File /users/6/croal008/5527 Project/ddc/learn/onset_train.py:198, in
train_onset_model(args, OnsetModel)
    196     print('Creating train model')
    197     with tf.variable_scope('model_sp', reuse=None):
--> 198         model_train = OnsetModel(mode='train',
target_weight_strategy=args.weight_strategy, batch_size=args.batch_size,
**model_config)
    200 if do_train_eval or do_eval or do_cnn_export:
    201     with tf.variable_scope('model_sp', reuse=do_train):

File /users/6/croal008/5527 Project/ddc/learn/onset_net_gina2.py:93, in
OnsetNet_Gina2.__init__(self, mode, batch_size, audio_context_radius,
audio_nbands, audio_nchannels, nfeats, cnn_filter_shapes, cnn_init,
cnn_pool, cnn_rnn_zack, rnn_cell_type, rnn_size, rnn_nlayers, rnn_init,
rnn_nunroll, rnn_keep_prob, dnn_sizes, dnn_init, dnn_keep_prob,
dnn_nonlin, target_weight_strategy, grad_clip, opt, export_feat_name,
zack_hack)
    90         biased = tf.nn.bias_add(conv, biases)
    91         convolved = tf.nn.relu(biased)

```

```

--
---> 93         batch_normalized =
tf.layers.batch_normalization(convolved, training=(mode == 'train'))
    95         pool_shape = [1, ptime, pband, 1]
    96 #         pooled = tf.nn.max_pool(convolved,
ksize=pool_shape, strides=pool_shape, padding='SAME')

File
/common/software/install/manual/jupyter_kernels/DeepLearning/lib/python3.1
1/site-packages/tensorflow/python/util/lazy_loader.py:207, in
KerasLazyLoader.__getattr__(self, item)
    200     raise AttributeError(
    201         "`tf.compat.v2.keras` is not available with Keras 3. Just
use "
    202         "`import keras` instead."
    203     )
    204 elif self._tfll_submodule and self._tfll_submodule.startswith(
    205     "__internal__.legacy."
    206 ):
--> 207     raise AttributeError(
    208         f"`{item}` is not available with Keras 3."
    209     )
    210 module = self._load()
    211 return getattr(module, item)

AttributeError: `batch_normalization` is not available with Keras 3.

```

I see the error - you're running into an issue with TensorFlow versions. The version you're using (TensorFlow 1.x with compatibility mode) doesn't support `tf.layers.batch_normalization` in the way I suggested.

Let me provide the correct implementation for your TensorFlow version:

```
python
```

```

# Instead of using tf.layers.batch_normalization, use tf.nn.batch_normalization
# First, define batch norm variables
batch_mean, batch_var = tf.nn.moments(convolved, [0, 1, 2])
scale = tf.get_variable(f'bn_scale_{i}', shape=[nfilt], initializer=tf.ones_initializer(), dtype=dtype)
beta = tf.get_variable(f'bn_beta_{i}', shape=[nfilt], initializer=tf.zeros_initializer(), dtype=dtype)
# For tracking running statistics (needed for inference)
pop_mean = tf.get_variable(f'bn_pop_mean_{i}', shape=[nfilt], initializer=tf.zeros_initializer(), dtype=dtype, trainable=False)
pop_var = tf.get_variable(f'bn_pop_var_{i}', shape=[nfilt], initializer=tf.ones_initializer(), dtype=dtype, trainable=False)

# Apply batch normalization
if mode == 'train':
    # During training, use batch statistics and update running stats
    decay = 0.99
    train_mean = batch_mean
    train_var = batch_var
    # Update population statistics for inference
    train_pop_mean = tf.assign(pop_mean, pop_mean * decay + batch_mean * (1 - decay))
    train_pop_var = tf.assign(pop_var, pop_var * decay + batch_var * (1 - decay))
    with tf.control_dependencies([train_pop_mean, train_pop_var]):
        batch_normalized = tf.nn.batch_normalization(
            convolved, train_mean, train_var, beta, scale, epsilon=1e-5)
else:
    # During inference, use population statistics
    batch_normalized = tf.nn.batch_normalization(
        convolved, pop_mean, pop_var, beta, scale, epsilon=1e-5)

# Then use batch_normalized in the pooling
pool_shape = [1, ptime, pband, 1]
pooled = tf.nn.max_pool(batch_normalized, ksize=pool_shape, strides=pool_shape, padding='SAME')

```

This approach uses the lower-level `tf.nn.batch_normalization` function which is compatible with TensorFlow 1.x. It also explicitly handles the running statistics that need to be maintained for inference.

Alternatively, you could switch to a different normalization technique that might be simpler to implement with your version of TensorFlow:

python

```
# Layer normalization (simpler alternative)
# Normalize across channels for each example
mean, variance = tf.nn.moments(convolved, axes=[3], keep_dims=True)
normalized = (convolved - mean) / tf.sqrt(variance + 1e-5)
gamma = tf.get_variable(f'ln_gamma_{i}', shape=[1, 1, 1, nfilt], initializer=tf.ones_initializer(), dtype=dtype)
beta = tf.get_variable(f'ln_beta_{i}', shape=[1, 1, 1, nfilt], initializer=tf.zeros_initializer(), dtype=dtype)
normalized = normalized * gamma + beta

# Then use normalized in pooling
pool_shape = [1, ptime, pband, 1]
pooled = tf.nn.max_pool(normalized, ksize=pool_shape, strides=pool_shape, padding='SAME')
```

This layer normalization approach might be easier to implement and can still provide many of the benefits of normalization without the complexity of tracking running statistics.

```

-----
-
TypeError                                Traceback (most recent call
last)
Cell In[2], line 52
    49 # Update the training_args argument container in place
    50 vars(training_args).update(training_args_dict)
--> 52 train_onset_model(training_args, OnsetModel=OnsetNet_Gina2)
    55 # Results 4/21 - Hard, Challenge
    56
    57 # Completed epoch 1
    (...)
    61 # Completed epoch 10
    62 # Epoch mean cross-entropy (nats)

File /users/6/croal008/5527 Project/ddc/learn/onset_train.py:198, in
train_onset_model(args, OnsetModel)
    196     print('Creating train model')
    197     with tf.variable_scope('model_sp', reuse=None):
--> 198         model_train = OnsetModel(mode='train',
target_weight_strategy=args.weight_strategy, batch_size=args.batch_size,
**model_config)
    200 if do_train_eval or do_eval or do_cnn_export:
    201     with tf.variable_scope('model_sp', reuse=do_train):

File /users/6/croal008/5527 Project/ddc/learn/onset_net_gina2.py:112, in
OnsetNet_Gina2.__init__(self, mode, batch_size, audio_context_radius,
audio_nbands, audio_nchannels, nfeats, cnn_filter_shapes, cnn_init,
cnn_pool, cnn_rnn_zack, rnn_cell_type, rnn_size, rnn_nlayers, rnn_init,
rnn_nunroll, rnn_keep_prob, dnn_sizes, dnn_init, dnn_keep_prob,
dnn_nonlin, target_weight_strategy, grad_clip, opt, export_feat_name,
zack_hack)
    110     train_pop_var = tf.assign(pop_var, pop_var * decay + batch_var
* (1 - decay))

```

```

111     with tf.control_dependencies([train_pop_mean, train_pop_var]):
--> 112         batch_normalized = tf.nn.batch_normalization(
113             convolved, train_mean, train_var, beta, scale,
epsilon=1e-5)
114     else:
115         # During inference, use population statistics
116         batch_normalized = tf.nn.batch_normalization(
117             convolved, pop_mean, pop_var, beta, scale, epsilon=1e-5)

```

File

```

/common/software/install/manual/jupyter_kernels/DeepLearning/lib/python3.1
1/site-packages/tensorflow/python/util/traceback_utils.py:153, in
filter_traceback.<locals>.error_handler(*args, **kwargs)
151 except Exception as e:
152     filtered_tb = _process_traceback_frames(e.__traceback__)
--> 153     raise e.with_traceback(filtered_tb) from None
154 finally:
155     del filtered_tb

```

File

```

/common/software/install/manual/jupyter_kernels/DeepLearning/lib/python3.1
1/site-packages/tensorflow/python/util/dispatch.py:1254, in
add_dispatch_support.<locals>.decorator.
<locals>.op_dispatch_handler(*args, **kwargs)
1252 if iterable_params is not None:
1253     args, kwargs = replace_iterable_params(args, kwargs,
iterable_params)
-> 1254 result = api_dispatcher.Dispatch(args, kwargs)
1255 if result is not NotImplemented:
1256     return result

```

TypeError: Got an unexpected keyword argument 'epsilon'

I see the new error. The issue is with the `epsilon` parameter. Let me provide a corrected version that will work with your TensorFlow

version:

```
python

# Fix the batch normalization implementation
# First, define batch norm variables
batch_mean, batch_var = tf.nn.moments(convolved, [0, 1, 2])
scale = tf.get_variable(f'bn_scale_{i}', shape=[nfilt], initializer=tf.ones_initializer(), dtype=dtype)
beta = tf.get_variable(f'bn_beta_{i}', shape=[nfilt], initializer=tf.zeros_initializer(), dtype=dtype)
# For tracking running statistics (needed for inference)
pop_mean = tf.get_variable(f'bn_pop_mean_{i}', shape=[nfilt], initializer=tf.zeros_initializer(), dtype=dtype, trainable=False)
pop_var = tf.get_variable(f'bn_pop_var_{i}', shape=[nfilt], initializer=tf.ones_initializer(), dtype=dtype, trainable=False)

# Apply batch normalization
if mode == 'train':
    # During training, use batch statistics and update running stats
    decay = 0.99
    train_mean = batch_mean
    train_var = batch_var
    # Update population statistics for inference
    train_pop_mean = tf.assign(pop_mean, pop_mean * decay + batch_mean * (1 - decay))
    train_pop_var = tf.assign(pop_var, pop_var * decay + batch_var * (1 - decay))
    with tf.control_dependencies([train_pop_mean, train_pop_var]):
        # Note: removed epsilon parameter, using variance_epsilon instead
        batch_normalized = tf.nn.batch_normalization(
            convolved, train_mean, train_var, beta, scale, variance_epsilon=1e-5)
else:
    # During inference, use population statistics
    batch_normalized = tf.nn.batch_normalization(
        convolved, pop_mean, pop_var, beta, scale, variance_epsilon=1e-5)
```

The key change is replacing `epsilon=1e-5` with `variance_epsilon=1e-5`.

Alternatively, since you're having compatibility issues with batch normalization, I'd suggest trying the simpler layer normalization approach:

python

```
# Simpler alternative: Layer normalization
# Normalize across channels for each example
mean, variance = tf.nn.moments(convolved, [3], keep_dims=True)
normalized = (convolved - mean) / tf.sqrt(variance + 1e-5)
gamma = tf.get_variable(f'ln_gamma_{i}', [1, 1, 1, nfilt], initializer=tf.ones_initializer(), dtype=dtype)
beta = tf.get_variable(f'ln_beta_{i}', [1, 1, 1, nfilt], initializer=tf.zeros_initializer(), dtype=dtype)
normalized = normalized * gamma + beta

# Then use normalized in pooling
pool_shape = [1, ptime, pband, 1]
pooled = tf.nn.max_pool(normalized, ksize=pool_shape, strides=pool_shape, padding='SAME')
```

If you're continuing to encounter version compatibility issues with the normalized approaches, another alternative would be to simply focus on other improvements such as:

1. Adding the RNN layers (rnn_nlayers = 2)
2. Changing the activation function to ReLU
3. Implementing the learning rate decay
4. Switching from SGD to Adam

These changes are less likely to run into compatibility issues while still providing significant performance improvements.

G What about batch2dnorm?

You're right to suggest `tf.nn.batch2d_normalization` as an alternative! This could be a more compatible option for your TensorFlow 1.x implementation.

Here's how you could implement it in your CNN layers:

```
python

# For a 2D batch normalization approach
# Add after the convolved = tf.nn.relu(biased) line in your CNN section

# Create variables for batch normalization
scale = tf.get_variable(f'bn_scale_{i}', shape=[nfilt],
                        initializer=tf.ones_initializer(), dtype=dtype)
beta = tf.get_variable(f'bn_beta_{i}', shape=[nfilt],
                      initializer=tf.zeros_initializer(), dtype=dtype)

# For tracking mean and variance
pop_mean = tf.get_variable(f'bn_pop_mean_{i}', shape=[nfilt],
                          initializer=tf.zeros_initializer(),
                          dtype=dtype, trainable=False)
pop_var = tf.get_variable(f'bn_pop_var_{i}', shape=[nfilt],
                        initializer=tf.ones_initializer(),
                        dtype=dtype, trainable=False)

# Reshape for batch normalization (flatten the spatial dimensions)
shape = convolved.get_shape().as_list()
reshaped_conv = tf.reshape(convolved, [-1, shape[1] * shape[2], shape[3]])

# Apply batch normalization
if mode == 'train':
    batch_mean, batch_var = tf.nn.moments(reshaped_conv, [0, 1])
    decay = 0.99
    train_mean = tf.assign(pop_mean, pop_mean * decay + batch_mean * (1 - decay))
    train_var = tf.assign(pop_var, pop_var * decay + batch_var * (1 - decay))
```

```

with tf.control_dependencies([train_mean, train_var]):
    normalized = tf.nn.batch_normalization(
        reshaped_conv, batch_mean, batch_var, beta, scale, variance_epsilon=1e-5)
else:
    normalized = tf.nn.batch_normalization(
        reshaped_conv, pop_mean, pop_var, beta, scale, variance_epsilon=1e-5)

# Reshape back to original shape
batch_normalized = tf.reshape(normalized, shape)

# Then continue with pooling
pool_shape = [1, ptime, pband, 1]
pooled = tf.nn.max_pool(batch_normalized, ksize=pool_shape, strides=pool_shape, padding='SAME')

```

This approach:

1. Creates variables to store scale, shift, and running statistics
2. Reshapes the convolution output to collapse spatial dimensions
3. Applies batch normalization
4. Reshapes back to original tensor shape
5. Continues with pooling

If you're still encountering issues with the batch normalization implementations, I'd recommend trying the simplest approach to make progress: instance normalization. This requires minimal changes and is very stable:

python



```

# Instance normalization (normalize each feature map independently per example)
# Simple approach that often works well for audio/image tasks

# Calculate mean and variance for each example and feature map
inst_mean, inst_var = tf.nn.moments(convolved, axes=[1, 2], keep_dims=True)

```

```
normalized = (convolved - inst_mean) / tf.sqrt(inst_var + 1e-5)

# No need for running stats as each example is normalized independently
gamma = tf.get_variable(f'norm_gamma_{i}', shape=[1, 1, 1, nfilt],
                        initializer=tf.ones_initializer(), dtype=dtype)
beta = tf.get_variable(f'norm_beta_{i}', shape=[1, 1, 1, nfilt],
                      initializer=tf.zeros_initializer(), dtype=dtype)

normalized = normalized * gamma + beta

# Use normalized result in pooling
pool_shape = [1, ptime, pband, 1]
pooled = tf.nn.max_pool(normalized, ksize=pool_shape, strides=pool_shape, padding='SAME')
```

This instance normalization approach is simpler and often works well for audio processing tasks, without requiring running statistics.