cnn_v9

October 14, 2022

[ ]:
```
%env LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CONDA_PREFIX/lib/
```

```
env: LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CONDA_PREFIX/lib/
```

[ ]:
```
import os
print(os.environ["LD_LIBRARY_PATH"])
```

```
:/home/nkspartan/miniconda3/envs/tf-gpu/lib/:/home/nkspartan/miniconda3/envs/tf-
gpu/lib/
```

[ ]:
```
import tensorflow as tf
import numpy as np
import pandas as pd
import os
import keras

from keras import Sequential, models, Input
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout,␣
 ↪LeakyReLU, AveragePooling2D
from keras.optimizers import SGD, Adam
```

[ ]:
```
from tensorflow.python.client import device_lib

#print(device_lib.list_local_devices())
print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
```

```
Default GPU Device: /device:GPU:0
```

```
2022-10-14 11:39:11.186822: I tensorflow/core/platform/cpu_feature_guard.cc:193]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2022-10-14 11:39:11.209616: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:980] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-10-14 11:39:11.261664: I
```

```
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:980] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-10-14 11:39:11.261850: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:980] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-10-14 11:39:12.100458: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:980] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-10-14 11:39:12.101067: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:980] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-10-14 11:39:12.101224: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:980] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-10-14 11:39:12.101367: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1616] Created device
/device:GPU:0 with 4023 MB memory:  -> device: 0, name: NVIDIA GeForce RTX 2060,
pci bus id: 0000:08:00.0, compute capability: 7.5
```

0.1 Read the csv dataset to get the values for stage and discharge of the images

[ ]:

```
df = pd.read_csv("../../dataset/2012_2019_PlatteRiverWeir_features_merged_all.
 ↪csv")
df.head()
```

[ ]:

```
   Unnamed: 0           SensorTime          CaptureTime  \
0           0  2012-06-09 13:15:00  2012-06-09T13:09:07
1           1  2012-06-09 13:15:00  2012-06-09T13:10:29
2           2  2012-06-09 13:45:00  2012-06-09T13:44:01
3           3  2012-06-09 14:45:00  2012-06-09T14:44:30
4           4  2012-06-09 15:45:00  2012-06-09T15:44:59

                                 Filename Agency  SiteNumber TimeZone  Stage  \
0  StateLineWeir_20120609_Farrell_001.jpg   USGS     6674500      MDT   2.99
1  StateLineWeir_20120609_Farrell_002.jpg   USGS     6674500      MDT   2.99
2  StateLineWeir_20120609_Farrell_003.jpg   USGS     6674500      MDT   2.96
3  StateLineWeir_20120609_Farrell_004.jpg   USGS     6674500      MDT   2.94
4  StateLineWeir_20120609_Farrell_005.jpg   USGS     6674500      MDT   2.94

   Discharge        CalcTimestamp  …  WeirPt2X  WeirPt2Y  WwRawLineMin  \
0      916.0  2020-03-11T16:58:28  …        -1        -1           0.0
1      916.0  2020-03-11T16:58:33  …        -1        -1           0.0
```

```
2      873.0  2020-03-11T16:58:40  …        -1        -1            0.0
3      846.0  2020-03-11T16:58:47  …        -1        -1            0.0
4      846.0  2020-03-11T16:58:55  …        -1        -1            0.0

   WwRawLineMax  WwRawLineMean  WwRawLineSigma  WwCurveLineMin  \
0           0.0            0.0             0.0             0.0
1           0.0            0.0             0.0             0.0
2           0.0            0.0             0.0             0.0
3           0.0            0.0             0.0             0.0
4           0.0            0.0             0.0             0.0

   WwCurveLineMax  WwCurveLineMean  WwCurveLineSigma
0             0.0              0.0               0.0
1             0.0              0.0               0.0
2             0.0              0.0               0.0
3             0.0              0.0               0.0
4             0.0              0.0               0.0

[5 rows x 60 columns]
```

[ ]:

```
df = df[["Filename", "Stage", "Discharge"]]
```

0.1.1 Scale the data

[ ]:

```
from sklearn.preprocessing import StandardScaler
from joblib import load

#scaler = StandardScaler()
scaler = load('std_scaler.joblib')
```

[ ]:

```
df[["Stage", "Discharge"]] = scaler.fit_transform(df[["Stage", "Discharge"]])
df
```

[ ]:

```
                                     Filename     Stage  Discharge
0      StateLineWeir_20120609_Farrell_001.jpg  0.138117  -0.046094
1      StateLineWeir_20120609_Farrell_002.jpg  0.138117  -0.046094
2      StateLineWeir_20120609_Farrell_003.jpg  0.100875  -0.082160
3      StateLineWeir_20120609_Farrell_004.jpg  0.076046  -0.104807
4      StateLineWeir_20120609_Farrell_005.jpg  0.076046  -0.104807
…                                         …         …          …
42054  StateLineWeir_20191011_Farrell_409.jpg -0.420526  -0.450369
42055  StateLineWeir_20191011_Farrell_410.jpg -0.420526  -0.450369
42056  StateLineWeir_20191011_Farrell_411.jpg -0.420526  -0.450369
42057  StateLineWeir_20191011_Farrell_412.jpg -0.420526  -0.450369
42058  StateLineWeir_20191011_Farrell_413.jpg -0.420526  -0.450369

[42059 rows x 3 columns]
```

[ ]:

```
#from joblib import dump
#dump(scaler, 'std_scaler.joblib', compress=True)
```

[ ]:

```
['std_scaler.joblib']
```

0.2 Create the dataset pipeline

[ ]:

```
IMG_SIZE = 512
BATCH_SIZE = 32
```

[ ]:

```
from glob import glob

def make_dataset(path, batch_size, df, seed=None):
  np.random.seed(seed)

  rotation = tf.keras.layers.RandomRotation(0.2)
  flip = tf.keras.layers.RandomFlip("horizontal_and_vertical")
  translation = tf.keras.layers.RandomTranslation(height_factor=0.2,␣
 ↪width_factor=0.2)
  zoom = tf.keras.layers.RandomZoom(0.3)
  brightness = tf.keras.layers.RandomBrightness([-0.2,0.5])
  contrast = tf.keras.layers.RandomContrast(0.2)

  def parse_image(filename):
    image = tf.io.read_file(filename)
    image = tf.image.decode_jpeg(image, channels=3)
    #image = tf.image.resize(image, [IMG_SIZE, IMG_SIZE])

    # image augmentation
    image = brightness(image)
    image = contrast(image)
    image = zoom(image)
    image = flip(image)
    image = translation(image)
    image = rotation(image)

    image = tf.cast(image, tf.float32)
    image /= 255
    return image

  def configure_for_performance(ds):
    ds = ds.shuffle(buffer_size=100)
    ds = ds.batch(batch_size)
    ds = ds.repeat()
    ds = ds.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
    return ds
```

```
filenames = glob(path + '/*')

# make train, val and test splits of the dataset (70%, 10%, 20% split)
split1 = int(0.7 * len(filenames))
split2 = int(0.8 * len(filenames))

np.random.shuffle(filenames)
train_files = filenames[:split1] # up to split 1 (ex 70%)
val_files = filenames[split1:split2] # from ex. 70% to 80%
test_files = filenames[split2:] # from ex. 80% until the end

# create stage values
stage_train_values = [df[df.Filename == file.split('/')[-1]].Stage.values for␣
↪file in train_files]
stage_val_values = [df[df.Filename == file.split('/')[-1]].Stage.values for␣
↪file in val_files]
stage_test_values = [df[df.Filename == file.split('/')[-1]].Stage.values for␣
↪file in test_files]

# create discharge values
discharge_train_values = [df[df.Filename == file.split(
    '/')[-1]].Discharge.values for file in train_files]
discharge_val_values = [df[df.Filename == file.split(
    '/')[-1]].Discharge.values for file in val_files]
discharge_test_values = [df[df.Filename == file.split(
    '/')[-1]].Discharge.values for file in test_files]

# join stage and discharge values
stage_discharge_train_values = [[np.squeeze(s), np.squeeze(d)] for s, d in␣
↪zip(stage_train_values, discharge_train_values)]
stage_discharge_val_values = [[np.squeeze(s), np.squeeze(d)] for s, d in␣
↪zip(stage_val_values, discharge_val_values)]
stage_discharge_test_values = [[np.squeeze(s), np.squeeze(
    d)] for s, d in zip(stage_test_values, discharge_test_values)]

# create images dataset (train, val, test)
filenames_train_ds = tf.data.Dataset.from_tensor_slices(train_files)
filenames_val_ds = tf.data.Dataset.from_tensor_slices(val_files)
filenames_test_ds = tf.data.Dataset.from_tensor_slices(test_files)

images_train_ds = filenames_train_ds.map(parse_image, num_parallel_calls=6)
images_val_ds = filenames_val_ds.map(parse_image, num_parallel_calls=6)
images_test_ds = filenames_test_ds.map(parse_image, num_parallel_calls=6)

# create stage and discharge dataset (train, val, test)
stage_discharge_train_ds = tf.data.Dataset.
↪from_tensor_slices(stage_discharge_train_values)
```

```
    stage_discharge_val_ds = tf.data.Dataset.
↪from_tensor_slices(stage_discharge_val_values)
    stage_discharge_test_ds = tf.data.Dataset.from_tensor_slices(
        stage_discharge_test_values)

    # create tensorflow dataset of images and values (train, val, test)
    train_ds = tf.data.Dataset.zip((images_train_ds, stage_discharge_train_ds))
    train_ds = configure_for_performance(train_ds)
    val_ds = tf.data.Dataset.zip((images_val_ds, stage_discharge_val_ds))
    val_ds = configure_for_performance(val_ds)
    test_ds = tf.data.Dataset.zip((images_test_ds, stage_discharge_test_ds))
    test_ds = configure_for_performance(test_ds)

    return train_ds, len(train_files), val_ds, len(val_files), test_ds,␣
↪len(test_files)
```

```
[ ]: path = "../../dataset/images"

     train_ds, train_size, val_ds, val_size, test_ds, test_size = make_dataset(path,␣
     ↪BATCH_SIZE, df, 0)
```

```
[ ]: input_shape = 0
     output_shape = 0

     for image, stage_discharge in train_ds.take(1):
         print(image.numpy().shape)
         print(stage_discharge.numpy().shape)

         input_shape = image.numpy().shape[1:]
         output_shape = stage_discharge.numpy().shape[1:]
```

```
(32, 512, 512, 3)
(32, 2)
```

```
[ ]: print(input_shape)
     print(output_shape)
```

```
(512, 512, 3)
(2,)
```

0.3 Create model

```
[ ]: def create_model(input_shape, output_shape):
         model = Sequential()

         model.add(Input(shape=input_shape))
```

```
    model.add(Conv2D(64, kernel_size=(4, 4), strides=(2, 2), padding='same',␣
↪activation=LeakyReLU()))
    model.add(MaxPooling2D(pool_size=(4, 4)))

    model.add(Conv2D(64, kernel_size=(4, 4), activation=LeakyReLU(),␣
↪padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same'))
    #model.add(AveragePooling2D(pool_size=(2, 2)))

    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
    model.add(AveragePooling2D(pool_size=(2, 2)))


    model.add(Flatten())
    model.add(Dense(128, activation='tanh'))
    model.add(Dropout(0.2))
    model.add(Dense(64, activation='tanh'))
    model.add(Dense(32, activation='tanh'))
    model.add(Dense(32, activation='tanh'))
    model.add(Dense(output_shape, activation='linear')) # linear regression␣
↪output layer

    return model
```

```
[ ]: model = create_model(input_shape, output_shape[0])
```

```
[ ]: model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_16 (Conv2D)          (None, 256, 256, 64)      3136

 max_pooling2d_8 (MaxPooling  (None, 64, 64, 64)       0
 2D)

 conv2d_17 (Conv2D)          (None, 64, 64, 64)        65600

 max_pooling2d_9 (MaxPooling  (None, 32, 32, 64)       0
 2D)

 conv2d_18 (Conv2D)          (None, 32, 32, 32)        18464

 conv2d_19 (Conv2D)          (None, 30, 30, 32)        9248
 average_pooling2d_5 (Averag  (None, 15, 15, 32)       0
 ePooling2D)

 flatten_4 (Flatten)         (None, 7200)              0

 dense_20 (Dense)            (None, 128)               921728

 dropout_2 (Dropout)         (None, 128)               0

 dense_21 (Dense)            (None, 64)                8256

 dense_22 (Dense)            (None, 32)                2080

 dense_23 (Dense)            (None, 32)                1056

 dense_24 (Dense)            (None, 2)                 66

=================================================================
Total params: 1,029,634
Trainable params: 1,029,634
Non-trainable params: 0
_________________________________________________________________
```

[ ]:

```
def compile_model(loss_func, optimizer, metrics=["accuracy"]):
    model.compile(loss=loss_func, optimizer=optimizer, metrics=metrics)
```

[ ]:

```
sgd = SGD(learning_rate=0.01, decay=1e-4, momentum=0.9, nesterov=True)
adam = Adam(learning_rate=1e-3, decay=1e-3 / 100)

compile_model('mse', adam,  [
              'mse', tf.keras.metrics.RootMeanSquaredError(name='rmse'), 'mae',␣
 ↪'mape'])
```

[ ]:

```
def fit_model(training_values, validation_values=None, epochs=10, steps=32,␣
 ↪val_steps=32, callbacks=[]):
    return model.fit(training_values, validation_data=validation_values,␣
 ↪epochs=epochs, steps_per_epoch=steps, validation_steps=val_steps,␣
 ↪callbacks=callbacks)
```

[ ]:

```
import datetime

date_actual = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
log_dir = "logs/fit/" + date_actual
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,␣
 ↪histogram_freq=1)
```

```
checkpoint_callback = tf.keras.callbacks.
↪ModelCheckpoint(filepath=f"model_weights/{date_actual}_cnn_best_weights.
↪hdf5",
                                monitor='val_mse',
                                verbose=1,
                                save_best_only=True)
```

[ ]:

```
# batch_size = 0 because we already have batch size in tf dataset
history = fit_model(train_ds, val_ds, epochs=25, steps=np.ceil(train_size /␣
↪BATCH_SIZE), val_steps=np.ceil(val_size / BATCH_SIZE),␣
↪callbacks=[tensorboard_callback, checkpoint_callback])
```

```
Epoch 1/25
920/921 [============================>.] - ETA: 0s - loss: 0.1127 - mse: 0.1127
- rmse: 0.3357 - mae: 0.1834 - mape: 84.4494
Epoch 1: val_mse improved from inf to 0.01864, saving model to
model_weights/20221014-142946_cnn_best_weights.hdf5
921/921 [==============================] - 104s 113ms/step - loss: 0.1127 - mse:
0.1127 - rmse: 0.3357 - mae: 0.1833 - mape: 84.4466 - val_loss: 0.0186 -
val_mse: 0.0186 - val_rmse: 0.1365 - val_mae: 0.0920 - val_mape: 47.8679
Epoch 2/25
920/921 [============================>.] - ETA: 0s - loss: 0.0149 - mse: 0.0149
- rmse: 0.1219 - mae: 0.0828 - mape: 36.7606
Epoch 2: val_mse improved from 0.01864 to 0.01040, saving model to
model_weights/20221014-142946_cnn_best_weights.hdf5
921/921 [==============================] - 104s 113ms/step - loss: 0.0149 - mse:
0.0149 - rmse: 0.1219 - mae: 0.0828 - mape: 36.7605 - val_loss: 0.0104 -
val_mse: 0.0104 - val_rmse: 0.1020 - val_mae: 0.0681 - val_mape: 25.4223
Epoch 3/25
920/921 [============================>.] - ETA: 0s - loss: 0.0094 - mse: 0.0094
- rmse: 0.0968 - mae: 0.0665 - mape: 32.3209
Epoch 3: val_mse improved from 0.01040 to 0.00554, saving model to
model_weights/20221014-142946_cnn_best_weights.hdf5
921/921 [==============================] - 101s 110ms/step - loss: 0.0094 - mse:
0.0094 - rmse: 0.0968 - mae: 0.0665 - mape: 32.3200 - val_loss: 0.0055 -
val_mse: 0.0055 - val_rmse: 0.0744 - val_mae: 0.0505 - val_mape: 20.0428
Epoch 4/25
920/921 [============================>.] - ETA: 0s - loss: 0.0064 - mse: 0.0064
- rmse: 0.0802 - mae: 0.0547 - mape: 31.1844
Epoch 4: val_mse improved from 0.00554 to 0.00465, saving model to
model_weights/20221014-142946_cnn_best_weights.hdf5
921/921 [==============================] - 107s 117ms/step - loss: 0.0064 - mse:
0.0064 - rmse: 0.0802 - mae: 0.0547 - mape: 31.1835 - val_loss: 0.0047 -
val_mse: 0.0047 - val_rmse: 0.0682 - val_mae: 0.0468 - val_mape: 21.0501
Epoch 5/25
920/921 [============================>.] - ETA: 0s - loss: 0.0057 - mse: 0.0057
- rmse: 0.0757 - mae: 0.0518 - mape: 25.5136
```

```
Epoch 5: val_mse did not improve from 0.00465
921/921 [==============================] - 107s 116ms/step - loss: 0.0057 - mse:
0.0057 - rmse: 0.0757 - mae: 0.0518 - mape: 25.5131 - val_loss: 0.0058 -
val_mse: 0.0058 - val_rmse: 0.0762 - val_mae: 0.0516 - val_mape: 20.2169
Epoch 6/25
920/921 [============================>.] - ETA: 0s - loss: 0.0047 - mse: 0.0047
- rmse: 0.0683 - mae: 0.0462 - mape: 25.0749
Epoch 6: val_mse improved from 0.00465 to 0.00308, saving model to
model_weights/20221014-142946_cnn_best_weights.hdf5
921/921 [==============================] - 111s 121ms/step - loss: 0.0047 - mse:
0.0047 - rmse: 0.0683 - mae: 0.0462 - mape: 25.0741 - val_loss: 0.0031 -
val_mse: 0.0031 - val_rmse: 0.0555 - val_mae: 0.0380 - val_mape: 15.7025
Epoch 7/25
920/921 [============================>.] - ETA: 0s - loss: 0.0044 - mse: 0.0044
- rmse: 0.0661 - mae: 0.0445 - mape: 22.8654
Epoch 7: val_mse did not improve from 0.00308
921/921 [==============================] - 111s 120ms/step - loss: 0.0044 - mse:
0.0044 - rmse: 0.0661 - mae: 0.0445 - mape: 22.8650 - val_loss: 0.0033 -
val_mse: 0.0033 - val_rmse: 0.0573 - val_mae: 0.0370 - val_mape: 16.1186
Epoch 8/25
920/921 [============================>.] - ETA: 0s - loss: 0.0039 - mse: 0.0039
- rmse: 0.0628 - mae: 0.0422 - mape: 23.0011
Epoch 8: val_mse did not improve from 0.00308
921/921 [==============================] - 110s 119ms/step - loss: 0.0039 - mse:
0.0039 - rmse: 0.0628 - mae: 0.0422 - mape: 23.0004 - val_loss: 0.0039 -
val_mse: 0.0039 - val_rmse: 0.0627 - val_mae: 0.0468 - val_mape: 14.7012
Epoch 9/25
920/921 [============================>.] - ETA: 0s - loss: 0.0121 - mse: 0.0121
- rmse: 0.1100 - mae: 0.0640 - mape: 29.0305
Epoch 9: val_mse did not improve from 0.00308
921/921 [==============================] - 110s 120ms/step - loss: 0.0121 - mse:
0.0121 - rmse: 0.1100 - mae: 0.0640 - mape: 29.0299 - val_loss: 0.0055 -
val_mse: 0.0055 - val_rmse: 0.0745 - val_mae: 0.0486 - val_mape: 18.6308
Epoch 10/25
920/921 [============================>.] - ETA: 0s - loss: 0.0054 - mse: 0.0054
- rmse: 0.0736 - mae: 0.0492 - mape: 28.6709
Epoch 10: val_mse did not improve from 0.00308
921/921 [==============================] - 102s 111ms/step - loss: 0.0054 - mse:
0.0054 - rmse: 0.0736 - mae: 0.0492 - mape: 28.6699 - val_loss: 0.0034 -
val_mse: 0.0034 - val_rmse: 0.0586 - val_mae: 0.0418 - val_mape: 18.0765
Epoch 11/25
920/921 [============================>.] - ETA: 0s - loss: 0.0039 - mse: 0.0039
- rmse: 0.0621 - mae: 0.0412 - mape: 19.3116
Epoch 11: val_mse did not improve from 0.00308
921/921 [==============================] - 103s 112ms/step - loss: 0.0039 - mse:
0.0039 - rmse: 0.0621 - mae: 0.0412 - mape: 19.3199 - val_loss: 0.0033 -
val_mse: 0.0033 - val_rmse: 0.0578 - val_mae: 0.0395 - val_mape: 20.0692
Epoch 12/25
```

```
920/921 [============================>.] - ETA: 0s - loss: 0.0032 - mse: 0.0032
- rmse: 0.0564 - mae: 0.0374 - mape: 19.3046
Epoch 12: val_mse improved from 0.00308 to 0.00249, saving model to
model_weights/20221014-142946_cnn_best_weights.hdf5
921/921 [==============================] - 101s 109ms/step - loss: 0.0032 - mse:
0.0032 - rmse: 0.0564 - mae: 0.0374 - mape: 19.3041 - val_loss: 0.0025 -
val_mse: 0.0025 - val_rmse: 0.0499 - val_mae: 0.0328 - val_mape: 15.0176
Epoch 13/25
920/921 [============================>.] - ETA: 0s - loss: 0.0030 - mse: 0.0030
- rmse: 0.0548 - mae: 0.0363 - mape: 18.6699
Epoch 13: val_mse improved from 0.00249 to 0.00225, saving model to
model_weights/20221014-142946_cnn_best_weights.hdf5
921/921 [==============================] - 109s 118ms/step - loss: 0.0030 - mse:
0.0030 - rmse: 0.0548 - mae: 0.0363 - mape: 18.6693 - val_loss: 0.0023 -
val_mse: 0.0023 - val_rmse: 0.0474 - val_mae: 0.0325 - val_mape: 14.6038
Epoch 14/25
920/921 [============================>.] - ETA: 0s - loss: 0.0027 - mse: 0.0027
- rmse: 0.0523 - mae: 0.0346 - mape: 16.8435
Epoch 14: val_mse did not improve from 0.00225
921/921 [==============================] - 111s 121ms/step - loss: 0.0027 - mse:
0.0027 - rmse: 0.0523 - mae: 0.0346 - mape: 16.8445 - val_loss: 0.0027 -
val_mse: 0.0027 - val_rmse: 0.0521 - val_mae: 0.0388 - val_mape: 14.0081
Epoch 15/25
920/921 [============================>.] - ETA: 0s - loss: 0.0027 - mse: 0.0027
- rmse: 0.0515 - mae: 0.0339 - mape: 15.9887
Epoch 15: val_mse improved from 0.00225 to 0.00207, saving model to
model_weights/20221014-142946_cnn_best_weights.hdf5
921/921 [==============================] - 108s 117ms/step - loss: 0.0027 - mse:
0.0027 - rmse: 0.0515 - mae: 0.0339 - mape: 15.9882 - val_loss: 0.0021 -
val_mse: 0.0021 - val_rmse: 0.0455 - val_mae: 0.0299 - val_mape: 11.6090
Epoch 16/25
920/921 [============================>.] - ETA: 0s - loss: 0.0058 - mse: 0.0058
- rmse: 0.0758 - mae: 0.0458 - mape: 27.4673
Epoch 16: val_mse did not improve from 0.00207
921/921 [==============================] - 104s 113ms/step - loss: 0.0058 - mse:
0.0058 - rmse: 0.0758 - mae: 0.0458 - mape: 27.4666 - val_loss: 0.0029 -
val_mse: 0.0029 - val_rmse: 0.0539 - val_mae: 0.0357 - val_mape: 15.2613
Epoch 17/25
920/921 [============================>.] - ETA: 0s - loss: 0.0063 - mse: 0.0063
- rmse: 0.0794 - mae: 0.0490 - mape: 24.2476
Epoch 17: val_mse did not improve from 0.00207
921/921 [==============================] - 111s 121ms/step - loss: 0.0063 - mse:
0.0063 - rmse: 0.0794 - mae: 0.0490 - mape: 24.2472 - val_loss: 0.0022 -
val_mse: 0.0022 - val_rmse: 0.0464 - val_mae: 0.0304 - val_mape: 12.3537
Epoch 18/25
920/921 [============================>.] - ETA: 0s - loss: 0.0030 - mse: 0.0030
- rmse: 0.0551 - mae: 0.0361 - mape: 23.2764
Epoch 18: val_mse improved from 0.00207 to 0.00190, saving model to
```

```
model_weights/20221014-142946_cnn_best_weights.hdf5
921/921 [==============================] - 113s 123ms/step - loss: 0.0030 - mse:
0.0030 - rmse: 0.0551 - mae: 0.0361 - mape: 23.2757 - val_loss: 0.0019 -
val_mse: 0.0019 - val_rmse: 0.0436 - val_mae: 0.0285 - val_mape: 11.7761
Epoch 19/25
920/921 [============================>.] - ETA: 0s - loss: 0.0026 - mse: 0.0026
- rmse: 0.0515 - mae: 0.0333 - mape: 20.4848
Epoch 19: val_mse did not improve from 0.00190
921/921 [==============================] - 111s 121ms/step - loss: 0.0026 - mse:
0.0026 - rmse: 0.0515 - mae: 0.0333 - mape: 20.4843 - val_loss: 0.0026 -
val_mse: 0.0026 - val_rmse: 0.0509 - val_mae: 0.0366 - val_mape: 18.1334
Epoch 20/25
920/921 [============================>.] - ETA: 0s - loss: 0.0024 - mse: 0.0024
- rmse: 0.0493 - mae: 0.0320 - mape: 18.9836
Epoch 20: val_mse improved from 0.00190 to 0.00161, saving model to
model_weights/20221014-142946_cnn_best_weights.hdf5
921/921 [==============================] - 112s 121ms/step - loss: 0.0024 - mse:
0.0024 - rmse: 0.0493 - mae: 0.0320 - mape: 18.9831 - val_loss: 0.0016 -
val_mse: 0.0016 - val_rmse: 0.0402 - val_mae: 0.0280 - val_mape: 10.5432
Epoch 21/25
920/921 [============================>.] - ETA: 0s - loss: 0.0023 - mse: 0.0023
- rmse: 0.0475 - mae: 0.0309 - mape: 17.3490
Epoch 21: val_mse did not improve from 0.00161
921/921 [==============================] - 112s 121ms/step - loss: 0.0023 - mse:
0.0023 - rmse: 0.0475 - mae: 0.0309 - mape: 17.3485 - val_loss: 0.0017 -
val_mse: 0.0017 - val_rmse: 0.0412 - val_mae: 0.0266 - val_mape: 11.5662
Epoch 22/25
920/921 [============================>.] - ETA: 0s - loss: 0.0025 - mse: 0.0025
- rmse: 0.0505 - mae: 0.0325 - mape: 17.3100
Epoch 22: val_mse did not improve from 0.00161
921/921 [==============================] - 112s 121ms/step - loss: 0.0025 - mse:
0.0025 - rmse: 0.0505 - mae: 0.0325 - mape: 17.3094 - val_loss: 0.0019 -
val_mse: 0.0019 - val_rmse: 0.0431 - val_mae: 0.0283 - val_mape: 12.5938
Epoch 23/25
920/921 [============================>.] - ETA: 0s - loss: 0.0021 - mse: 0.0021
- rmse: 0.0461 - mae: 0.0297 - mape: 19.0827
Epoch 23: val_mse did not improve from 0.00161
921/921 [==============================] - 111s 121ms/step - loss: 0.0021 - mse:
0.0021 - rmse: 0.0461 - mae: 0.0297 - mape: 19.0822 - val_loss: 0.0018 -
val_mse: 0.0018 - val_rmse: 0.0428 - val_mae: 0.0290 - val_mape: 13.3845
Epoch 24/25
920/921 [============================>.] - ETA: 0s - loss: 0.0022 - mse: 0.0022
- rmse: 0.0474 - mae: 0.0307 - mape: 14.6383
Epoch 24: val_mse improved from 0.00161 to 0.00160, saving model to
model_weights/20221014-142946_cnn_best_weights.hdf5
921/921 [==============================] - 112s 121ms/step - loss: 0.0022 - mse:
0.0022 - rmse: 0.0474 - mae: 0.0307 - mape: 14.6384 - val_loss: 0.0016 -
val_mse: 0.0016 - val_rmse: 0.0400 - val_mae: 0.0261 - val_mape: 10.2080
```

```
Epoch 25/25
920/921 [============================>.] - ETA: 0s - loss: 0.0019 - mse: 0.0019
- rmse: 0.0433 - mae: 0.0279 - mape: 15.0261
Epoch 25: val_mse improved from 0.00160 to 0.00147, saving model to
model_weights/20221014-142946_cnn_best_weights.hdf5
921/921 [==============================] - 113s 123ms/step - loss: 0.0019 - mse:
0.0019 - rmse: 0.0433 - mae: 0.0279 - mape: 15.0258 - val_loss: 0.0015 -
val_mse: 0.0015 - val_rmse: 0.0384 - val_mae: 0.0260 - val_mape: 10.8777
```

0.4 Evaluate model

[ ]:
```
print(date_actual)
```

```
20221014-142946
```

[ ]:
```
best_model = models.load_model(f'model_weights/{date_actual}_cnn_best_weights.
 ↪hdf5')
```

[ ]:
```
def evaluate_model(model, test_values, steps):
  score = model.evaluate(test_values, steps=steps)
  return score
```

[ ]:
```
test_loss, test_mse, test_rmse, test_mae, test_mape =␣
 ↪evaluate_model(best_model, test_ds, steps=np.ceil(test_size / BATCH_SIZE))
```

```
263/263 [==============================] - 17s 63ms/step - loss: 0.0042 - mse:
0.0042 - rmse: 0.0649 - mae: 0.0269 - mape: 18.0256
```

[ ]:
```
predictions = best_model.predict(test_ds, steps=np.ceil(test_size / BATCH_SIZE))
```

```
263/263 [==============================] - 17s 63ms/step
```

[ ]:
```
for image, stage_discharge in test_ds.take(1):
        predictions = best_model.predict(x=image)

        stage_discharge_test_values = stage_discharge[:2].numpy()
        predictions_values = predictions[:2]

        diff = predictions_values.flatten() - stage_discharge_test_values.
 ↪flatten()
        percentDiff = (diff / stage_discharge_test_values.flatten()) * 100
        absPercentDiff = np.abs(percentDiff)
        # compute the mean and standard deviation of the absolute percentage
        # difference
        mean = np.mean(absPercentDiff)
        std = np.std(absPercentDiff)
        # finally, show some statistics on our model
        print(mean)
        print(std)

        stage_discharge_test_values = stage_discharge[:10]
        predictions_values = predictions[:10]

        for i in range(len(stage_discharge_test_values.numpy())):
                print(f"pred stage: {scaler.
 ↪inverse_transform(predictions_values)[i][0]}, actual stage: {scaler.
 ↪inverse_transform(stage_discharge_test_values)[i][0]}")
                print(f"pred discharge: {scaler.
 ↪inverse_transform(predictions_values)[i][1]}, actual discharge: {scaler.
 ↪inverse_transform(stage_discharge_test_values)[i][1]}")
```

```
1/1 [==============================] - 0s 115ms/step
1.1849493671727356
0.3153837321914593
pred stage: 5.19284725189209, actual stage: 5.22
pred discharge: 4701.67333984375, actual discharge: 4760.0
pred stage: 3.822572708129883, actual stage: 3.81
pred discharge: 2047.3829345703125, actual discharge: 2040.0
pred stage: 2.2581021785736084, actual stage: 2.24
pred discharge: 188.7901153564453, actual discharge: 197.0
pred stage: 3.6058669090270996, actual stage: 3.47
pred discharge: 1850.30322265625, actual discharge: 1620.0
pred stage: 3.1100921630859375, actual stage: 3.07
pred discharge: 1071.93798828125, actual discharge: 1040.0
pred stage: 4.335366249084473, actual stage: 4.33
pred discharge: 3157.54248046875, actual discharge: 3140.0
pred stage: 2.583793878555298, actual stage: 2.54
pred discharge: 449.4427185058594, actual discharge: 425.0
pred stage: 2.45621657371521, actual stage: 2.47
pred discharge: 354.2511901855469, actual discharge: 375.0
pred stage: 3.130676746368408, actual stage: 3.13
pred discharge: 1150.66552734375, actual discharge: 1140.0
pred stage: 2.3002612590789795, actual stage: 2.28
pred discharge: -28.020427703857422, actual discharge: 172.0
```

[ ]:

```

```

0.5 Visualize layers

[ ]:

```
layer_outputs = [layer.output for layer in best_model.layers[:12]]
# Extracts the outputs of the top 12 layers
activation_model = models.Model(inputs=best_model.input, outputs=layer_outputs)␣
 ↪# Creates a model that will return these outputs, given the model input
```

[ ]:

```
activations = activation_model.predict(test_ds.take(1))
```

```
1/1 [==============================] - 0s 161ms/step
```

[ ]:

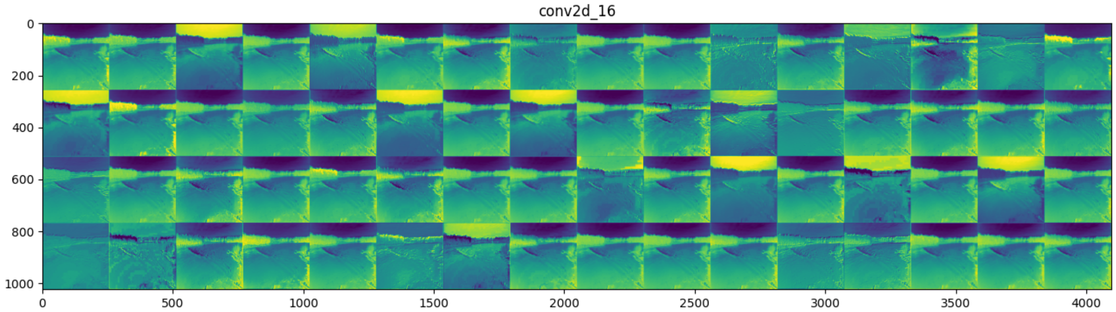
```
import matplotlib.pyplot as plt

layer_names = []
for layer in best_model.layers[:12]:
    layer_names.append(layer.name) # Names of the layers, so you can have them␣
 ↪as part of your plot

images_per_row = 16

for layer_name, layer_activation in zip(layer_names, activations): # Displays␣
 ↪the feature maps
    n_features = layer_activation.shape[-1] # Number of features in the feature␣
 ↪map
    size = layer_activation.shape[1] #The feature map has shape (1, size, size,␣
 ↪n_features).
    n_cols = n_features // images_per_row # Tiles the activation channels in␣
 ↪this matrix
    display_grid = np.zeros((size * n_cols, images_per_row * size))

    print(layer_name)
    if ("flatten" in layer_name): break

    for col in range(n_cols): # Tiles each filter into a big horizontal grid
        for row in range(images_per_row):
            channel_image = layer_activation[0,
                                             :, :,
                                             col * images_per_row + row]
            channel_image -= channel_image.mean() # Post-processes the feature␣
 ↪to make it visually palatable
            channel_image /= channel_image.std()
            channel_image *= 64
            channel_image += 128
            channel_image = np.clip(channel_image, 0, 255).astype('uint8')
            display_grid[col * size : (col + 1) * size, # Displays the grid
                         row * size : (row + 1) * size] = channel_image
    scale = 1. / size
    plt.figure(figsize=(scale * display_grid.shape[1],
                        scale * display_grid.shape[0]))
    plt.title(layer_name)
    plt.grid(False)
    plt.imshow(display_grid, aspect='auto', cmap='viridis')
```

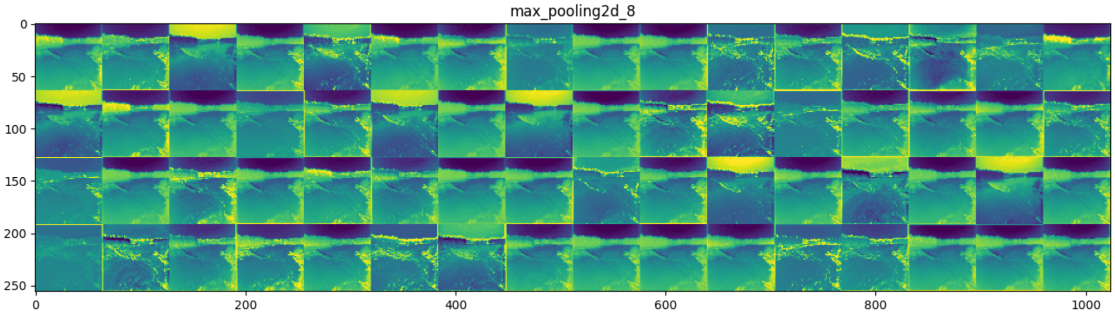
```
conv2d_16
max_pooling2d_8
conv2d_17
```

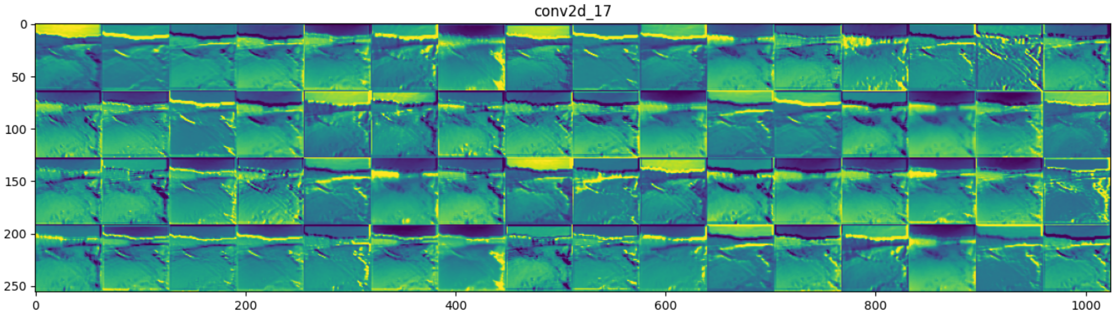
```
max_pooling2d_9
conv2d_18
conv2d_19
average_pooling2d_5
```

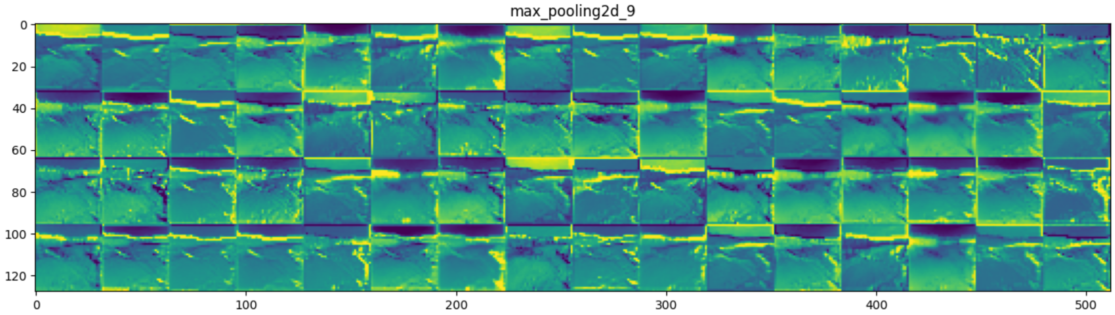
```
/tmp/ipykernel_11977/2269795348.py:24: RuntimeWarning: invalid value encountered
in divide
  channel_image /= channel_image.std()
```

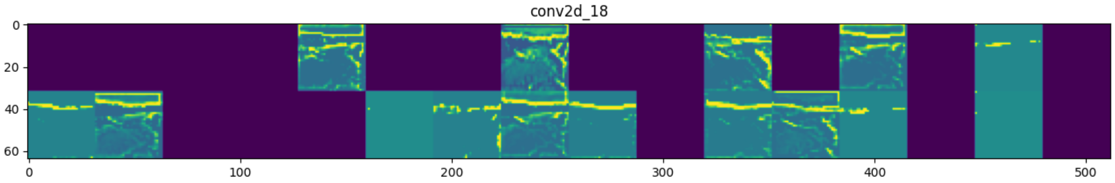
```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
Cell In [154], line 13
     11 size = layer_activation.shape[1] #The feature map has shape (1, size,␣
 ↪size, n_features).
     12 n_cols = n_features // images_per_row # Tiles the activation channels i␣
 ↪this matrix
---> 13 display_grid = np.zeros((size * n_cols, images_per_row * size))
     15 print(layer_name)
     16 if ("flatten" in layer_name): break

MemoryError: Unable to allocate 2.72 TiB for an array with shape (3240000,␣
 ↪115200) and data type float64
```

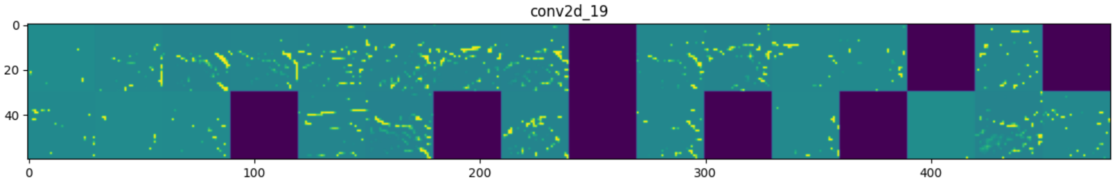


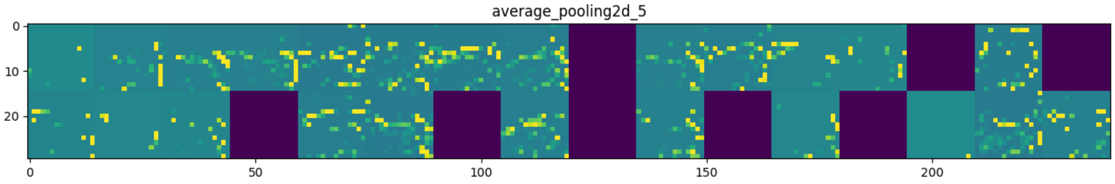



conv2d_17
0
50
100
150
200
250
0
200
400
600
800
1000


max_pooling2d_9
0
20
40
60
80
100
120
0
100
200
300
400
500


conv2d_18
0
20
40
60
0
100
200
300
400
500


conv2d_19
0
20
40
0
100
200
300
400


average_pooling2d_5
0
10
20
0
50
100
150
200