cnn_v6

October 13, 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
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
```

```
2022-10-13 22:40:45.131397: 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
2022-10-13 22:40:45.131676: 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
2022-10-13 22:40:45.131885: 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
2022-10-13 22:40:45.132157: 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
2022-10-13 22:40:45.132379: 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
2022-10-13 22:40:45.132570: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1616] Created device
/device:GPU:0 with 4078 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	Filename	Agency	SiteNumber	TimeZone	Stage	Discharge	CalcTimestamp	…	WeirPt2X	WeirPt2Y	WwRawLineMin	WwRawLineMax	WwRawLineMean	WwRawLineSigma	WwCurveLineMin
0	0	2012-06-09 13:15:00	2012-06-09T13:09:07	StateLineWeir_20120609_Farrell_001.jpg	USGS	6674500	MDT	2.99	916.0	2020-03-11T16:58:28	…	-1	-1	0.0	0.0	0.0	0.0	0.0
1	1	2012-06-09 13:15:00	2012-06-09T13:10:29	StateLineWeir_20120609_Farrell_002.jpg	USGS	6674500	MDT	2.99	916.0	2020-03-11T16:58:33	…	-1	-1	0.0	0.0	0.0	0.0	0.0
2	2	2012-06-09 13:45:00	2012-06-09T13:44:01	StateLineWeir_20120609_Farrell_003.jpg	USGS	6674500	MDT	2.96	873.0	2020-03-11T16:58:40	…	-1	-1	0.0	0.0	0.0	0.0	0.0
3	3	2012-06-09 14:45:00	2012-06-09T14:44:30	StateLineWeir_20120609_Farrell_004.jpg	USGS	6674500	MDT	2.94	846.0	2020-03-11T16:58:47	…	-1	-1	0.0	0.0	0.0	0.0	0.0
4	4	2012-06-09 15:45:00	2012-06-09T15:44:59	StateLineWeir_20120609_Farrell_005.jpg	USGS	6674500	MDT	2.94	846.0	2020-03-11T16:58:55	…	-1	-1	0.0	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

scaler = StandardScaler()
```

[ ]:
```
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, load
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)

  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 = 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=5)
    images_val_ds = filenames_val_ds.map(parse_image, num_parallel_calls=5)
    images_test_ds = filenames_test_ds.map(parse_image, num_parallel_calls=5)

    # 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(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))


    model.add(Flatten())
    model.add(Dense(64, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(32, activation='relu'))
    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_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)            (None, 256, 256, 64)      3136

 max_pooling2d_4 (MaxPooling  (None, 64, 64, 64)       0
 2D)

 conv2d_5 (Conv2D)            (None, 64, 64, 64)        65600

 max_pooling2d_5 (MaxPooling  (None, 32, 32, 64)       0
 2D)

 conv2d_6 (Conv2D)            (None, 32, 32, 32)        18464

 max_pooling2d_6 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_7 (Conv2D)            (None, 14, 14, 32)        9248

 max_pooling2d_7 (MaxPooling  (None, 7, 7, 32)         0
 2D)

 flatten_1 (Flatten)          (None, 1568)              0

 dense_5 (Dense)              (None, 64)                100416

 dense_6 (Dense)              (None, 64)                4160

 dense_7 (Dense)              (None, 32)                2080

 dense_8 (Dense)              (None, 32)                1056

 dense_9 (Dense)              (None, 2)                 66

=================================================================
Total params: 204,226
Trainable params: 204,226
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, batch_size=32,␣
 ↪epochs=10, steps=32, val_steps=32, callbacks=[]):
    return model.fit(training_values, validation_data=validation_values,␣
 ↪batch_size=batch_size, 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, batch_size=0, epochs=20, steps=np.
 ↪ceil(train_size / BATCH_SIZE), val_steps=np.ceil(val_size / BATCH_SIZE),␣
 ↪callbacks=[tensorboard_callback, checkpoint_callback])
```

```
Epoch 1/20
920/921 [============================>.] - ETA: 0s - loss: 0.1046 - mse: 0.1046
- rmse: 0.3234 - mae: 0.1727 - mape: 74.3008
Epoch 1: val_mse improved from inf to 0.02681, saving model to
model_weights/20221013-230243_cnn_best_weights.hdf5
921/921 [==============================] - 104s 113ms/step - loss: 0.1046 - mse:
0.1046 - rmse: 0.3234 - mae: 0.1727 - mape: 74.2984 - val_loss: 0.0268 -
val_mse: 0.0268 - val_rmse: 0.1637 - val_mae: 0.1172 - val_mape: 44.1462
Epoch 2/20
920/921 [============================>.] - ETA: 0s - loss: 0.0131 - mse: 0.0131
- rmse: 0.1147 - mae: 0.0754 - mape: 44.8075
Epoch 2: val_mse improved from 0.02681 to 0.01726, saving model to
model_weights/20221013-230243_cnn_best_weights.hdf5
921/921 [==============================] - 103s 112ms/step - loss: 0.0131 - mse:
0.0131 - rmse: 0.1147 - mae: 0.0754 - mape: 44.8062 - val_loss: 0.0173 -
```

```
val_mse: 0.0173 - val_rmse: 0.1314 - val_mae: 0.0962 - val_mape: 55.7794
Epoch 3/20
920/921 [============================>.] - ETA: 0s - loss: 0.0085 - mse: 0.0085
- rmse: 0.0923 - mae: 0.0611 - mape: 36.4498
Epoch 3: val_mse improved from 0.01726 to 0.01293, saving model to
model_weights/20221013-230243_cnn_best_weights.hdf5
921/921 [==============================] - 103s 112ms/step - loss: 0.0085 - mse:
0.0085 - rmse: 0.0923 - mae: 0.0611 - mape: 36.4496 - val_loss: 0.0129 -
val_mse: 0.0129 - val_rmse: 0.1137 - val_mae: 0.0866 - val_mape: 35.9213
Epoch 4/20
920/921 [============================>.] - ETA: 0s - loss: 0.0064 - mse: 0.0064
- rmse: 0.0798 - mae: 0.0536 - mape: 26.3036
Epoch 4: val_mse improved from 0.01293 to 0.00534, saving model to
model_weights/20221013-230243_cnn_best_weights.hdf5
921/921 [==============================] - 101s 110ms/step - loss: 0.0064 - mse:
0.0064 - rmse: 0.0798 - mae: 0.0536 - mape: 26.3028 - val_loss: 0.0053 -
val_mse: 0.0053 - val_rmse: 0.0731 - val_mae: 0.0497 - val_mape: 21.6067
Epoch 5/20
920/921 [============================>.] - ETA: 0s - loss: 0.0054 - mse: 0.0054
- rmse: 0.0737 - mae: 0.0494 - mape: 29.9201
Epoch 5: val_mse improved from 0.00534 to 0.00533, saving model to
model_weights/20221013-230243_cnn_best_weights.hdf5
921/921 [==============================] - 102s 111ms/step - loss: 0.0054 - mse:
0.0054 - rmse: 0.0737 - mae: 0.0494 - mape: 29.9194 - val_loss: 0.0053 -
val_mse: 0.0053 - val_rmse: 0.0730 - val_mae: 0.0501 - val_mape: 20.1663
Epoch 6/20
920/921 [============================>.] - ETA: 0s - loss: 0.0050 - mse: 0.0050
- rmse: 0.0710 - mae: 0.0477 - mape: 26.0078
Epoch 6: val_mse did not improve from 0.00533
921/921 [==============================] - 102s 110ms/step - loss: 0.0050 - mse:
0.0050 - rmse: 0.0710 - mae: 0.0477 - mape: 26.0110 - val_loss: 0.0070 -
val_mse: 0.0070 - val_rmse: 0.0834 - val_mae: 0.0595 - val_mape: 22.7570
Epoch 7/20
920/921 [============================>.] - ETA: 0s - loss: 0.0044 - mse: 0.0044
- rmse: 0.0661 - mae: 0.0441 - mape: 21.5896
Epoch 7: val_mse did not improve from 0.00533
921/921 [==============================] - 101s 110ms/step - loss: 0.0044 - mse:
0.0044 - rmse: 0.0661 - mae: 0.0441 - mape: 21.5889 - val_loss: 0.0065 -
val_mse: 0.0065 - val_rmse: 0.0803 - val_mae: 0.0593 - val_mape: 20.8868
Epoch 8/20
920/921 [============================>.] - ETA: 0s - loss: 0.0045 - mse: 0.0045
- rmse: 0.0671 - mae: 0.0444 - mape: 24.6374
Epoch 8: val_mse improved from 0.00533 to 0.00368, saving model to
model_weights/20221013-230243_cnn_best_weights.hdf5
921/921 [==============================] - 102s 111ms/step - loss: 0.0045 - mse:
0.0045 - rmse: 0.0671 - mae: 0.0444 - mape: 24.6366 - val_loss: 0.0037 -
val_mse: 0.0037 - val_rmse: 0.0606 - val_mae: 0.0423 - val_mape: 21.1125
Epoch 9/20
```

```
920/921 [============================>.] - ETA: 0s - loss: 0.0037 - mse: 0.0037
- rmse: 0.0610 - mae: 0.0403 - mape: 22.6372
Epoch 9: val_mse improved from 0.00368 to 0.00304, saving model to
model_weights/20221013-230243_cnn_best_weights.hdf5
921/921 [==============================] - 102s 110ms/step - loss: 0.0037 - mse:
0.0037 - rmse: 0.0610 - mae: 0.0403 - mape: 22.6366 - val_loss: 0.0030 -
val_mse: 0.0030 - val_rmse: 0.0551 - val_mae: 0.0377 - val_mape: 16.4290
Epoch 10/20
920/921 [============================>.] - ETA: 0s - loss: 0.0037 - mse: 0.0037
- rmse: 0.0607 - mae: 0.0397 - mape: 20.0673
Epoch 10: val_mse did not improve from 0.00304
921/921 [==============================] - 97s 106ms/step - loss: 0.0037 - mse:
0.0037 - rmse: 0.0607 - mae: 0.0397 - mape: 20.0667 - val_loss: 0.0053 -
val_mse: 0.0053 - val_rmse: 0.0728 - val_mae: 0.0474 - val_mape: 18.2901
Epoch 11/20
920/921 [============================>.] - ETA: 0s - loss: 0.0040 - mse: 0.0040
- rmse: 0.0631 - mae: 0.0414 - mape: 18.0414
Epoch 11: val_mse improved from 0.00304 to 0.00276, saving model to
model_weights/20221013-230243_cnn_best_weights.hdf5
921/921 [==============================] - 95s 103ms/step - loss: 0.0040 - mse:
0.0040 - rmse: 0.0631 - mae: 0.0414 - mape: 18.0408 - val_loss: 0.0028 -
val_mse: 0.0028 - val_rmse: 0.0526 - val_mae: 0.0353 - val_mape: 12.9992
Epoch 12/20
920/921 [============================>.] - ETA: 0s - loss: 0.0027 - mse: 0.0027
- rmse: 0.0524 - mae: 0.0341 - mape: 18.6730
Epoch 12: val_mse did not improve from 0.00276
921/921 [==============================] - 102s 110ms/step - loss: 0.0027 - mse:
0.0027 - rmse: 0.0524 - mae: 0.0341 - mape: 18.6726 - val_loss: 0.0168 -
val_mse: 0.0168 - val_rmse: 0.1297 - val_mae: 0.0831 - val_mape: 24.1903
Epoch 13/20
920/921 [============================>.] - ETA: 0s - loss: 0.0067 - mse: 0.0067
- rmse: 0.0818 - mae: 0.0499 - mape: 27.1674
Epoch 13: val_mse did not improve from 0.00276
921/921 [==============================] - 100s 109ms/step - loss: 0.0067 - mse:
0.0067 - rmse: 0.0818 - mae: 0.0499 - mape: 27.1667 - val_loss: 0.0033 -
val_mse: 0.0033 - val_rmse: 0.0573 - val_mae: 0.0399 - val_mape: 16.1353
Epoch 14/20
920/921 [============================>.] - ETA: 0s - loss: 0.0024 - mse: 0.0024
- rmse: 0.0489 - mae: 0.0318 - mape: 16.7597
Epoch 14: val_mse improved from 0.00276 to 0.00235, saving model to
model_weights/20221013-230243_cnn_best_weights.hdf5
921/921 [==============================] - 97s 105ms/step - loss: 0.0024 - mse:
0.0024 - rmse: 0.0489 - mae: 0.0318 - mape: 16.7592 - val_loss: 0.0024 -
val_mse: 0.0024 - val_rmse: 0.0485 - val_mae: 0.0329 - val_mape: 13.1014
Epoch 15/20
920/921 [============================>.] - ETA: 0s - loss: 0.0023 - mse: 0.0023
- rmse: 0.0476 - mae: 0.0312 - mape: 17.8224
Epoch 15: val_mse did not improve from 0.00235
```

```
921/921 [==============================] - 104s 113ms/step - loss: 0.0023 - mse:
0.0023 - rmse: 0.0476 - mae: 0.0312 - mape: 17.8218 - val_loss: 0.0024 -
val_mse: 0.0024 - val_rmse: 0.0485 - val_mae: 0.0329 - val_mape: 11.7082
Epoch 16/20
920/921 [============================>.] - ETA: 0s - loss: 0.0023 - mse: 0.0023
- rmse: 0.0475 - mae: 0.0311 - mape: 18.9885
Epoch 16: val_mse did not improve from 0.00235
921/921 [==============================] - 102s 110ms/step - loss: 0.0023 - mse:
0.0023 - rmse: 0.0475 - mae: 0.0311 - mape: 18.9881 - val_loss: 0.0037 -
val_mse: 0.0037 - val_rmse: 0.0606 - val_mae: 0.0467 - val_mape: 16.4856
Epoch 17/20
920/921 [============================>.] - ETA: 0s - loss: 0.0022 - mse: 0.0022
- rmse: 0.0470 - mae: 0.0308 - mape: 16.1451
Epoch 17: val_mse did not improve from 0.00235
921/921 [==============================] - 103s 112ms/step - loss: 0.0022 - mse:
0.0022 - rmse: 0.0470 - mae: 0.0308 - mape: 16.1447 - val_loss: 0.0024 -
val_mse: 0.0024 - val_rmse: 0.0485 - val_mae: 0.0339 - val_mape: 11.6829
Epoch 18/20
920/921 [============================>.] - ETA: 0s - loss: 0.0025 - mse: 0.0025
- rmse: 0.0502 - mae: 0.0325 - mape: 21.2667
Epoch 18: val_mse did not improve from 0.00235
921/921 [==============================] - 101s 110ms/step - loss: 0.0025 - mse:
0.0025 - rmse: 0.0502 - mae: 0.0325 - mape: 21.2665 - val_loss: 0.0026 -
val_mse: 0.0026 - val_rmse: 0.0505 - val_mae: 0.0343 - val_mape: 13.6109
Epoch 19/20
920/921 [============================>.] - ETA: 0s - loss: 0.0025 - mse: 0.0025
- rmse: 0.0501 - mae: 0.0320 - mape: 18.3541
Epoch 19: val_mse improved from 0.00235 to 0.00200, saving model to
model_weights/20221013-230243_cnn_best_weights.hdf5
921/921 [==============================] - 101s 110ms/step - loss: 0.0025 - mse:
0.0025 - rmse: 0.0501 - mae: 0.0320 - mape: 18.3536 - val_loss: 0.0020 -
val_mse: 0.0020 - val_rmse: 0.0447 - val_mae: 0.0301 - val_mape: 12.0683
Epoch 20/20
920/921 [============================>.] - ETA: 0s - loss: 0.0020 - mse: 0.0020
- rmse: 0.0449 - mae: 0.0289 - mape: 15.0879
Epoch 20: val_mse improved from 0.00200 to 0.00193, saving model to
model_weights/20221013-230243_cnn_best_weights.hdf5
921/921 [==============================] - 101s 110ms/step - loss: 0.0020 - mse:
0.0020 - rmse: 0.0449 - mae: 0.0289 - mape: 15.0875 - val_loss: 0.0019 -
val_mse: 0.0019 - val_rmse: 0.0440 - val_mae: 0.0303 - val_mape: 11.4709
```

0.4 Evaluate model

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

```
20221013-230243
```

[ ]:
```
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 [==============================] - 15s 57ms/step - loss: 0.0048 - mse:
0.0048 - rmse: 0.0695 - mae: 0.0313 - mape: 20.1478
```

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

```
263/263 [==============================] - 15s 56ms/step
```

[ ]:
```
#small_test_ds = next(iter(test_ds))
```

[ ]:
```
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 66ms/step
250.28289223903695
404.9078107444481
pred stage: 2.3132236003875732, actual stage: 2.32
pred discharge: 257.7745056152344, actual discharge: 268.0
pred stage: 2.8919501304626465, actual stage: 2.88
pred discharge: 877.227294921875, actual discharge: 792.0
pred stage: 2.3750550746917725, actual stage: 2.41
pred discharge: 280.3871765136719, actual discharge: 307.0
pred stage: 3.509460687637329, actual stage: 3.47
pred discharge: 1693.169921875, actual discharge: 1620.0
pred stage: 2.301117420196533, actual stage: 2.33
pred discharge: 1.4392883777618408, actual discharge: 0.0
pred stage: 2.1166560649871826, actual stage: 2.15
pred discharge: 123.49507141113281, actual discharge: 151.0
pred stage: 2.196364164352417, actual stage: 2.24
pred discharge: 177.15699768066406, actual discharge: 197.0
pred stage: 2.647357702255249, actual stage: 2.64
pred discharge: 554.3690795898438, actual discharge: 527.0
pred stage: 3.494729995727539, actual stage: 3.51
pred discharge: 1647.387939453125, actual discharge: 1650.0
pred stage: 3.8796825408935547, actual stage: 3.93
pred discharge: 2127.03271484375, actual discharge: 2210.0000000000005
```

[ ]:

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 199ms/step
```

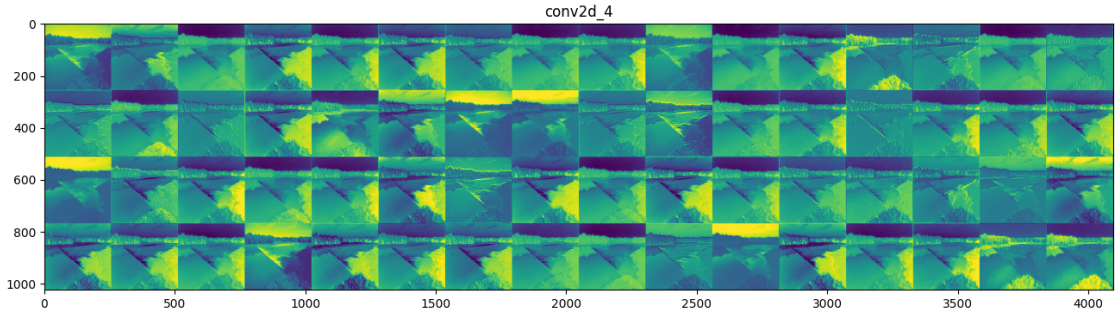
[ ]:
```
import matplotlib.pyplot as plt

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
```

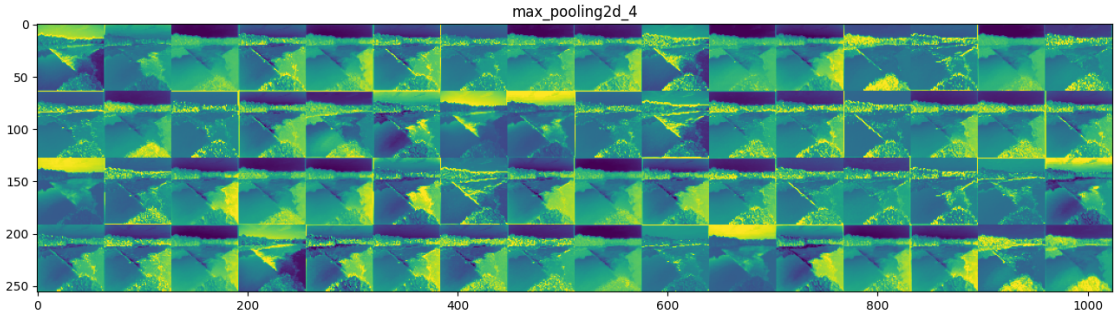
```
for layer_name, layer_activation in zip(layer_names, activations): # Displays␣
↪the feature maps
    n_features = layer_activation.shape[-1] # Number of features in the feature␣
↪map
    size = layer_activation.shape[1] #The feature map has shape (1, size, size,␣
↪n_features).
    n_cols = n_features // 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')
```

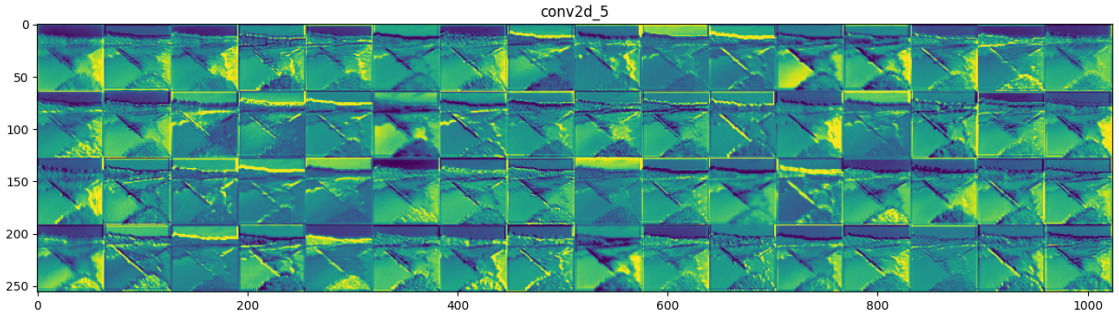
```
conv2d_4
max_pooling2d_4
conv2d_5
max_pooling2d_5
conv2d_6
max_pooling2d_6
conv2d_7
max_pooling2d_7
```

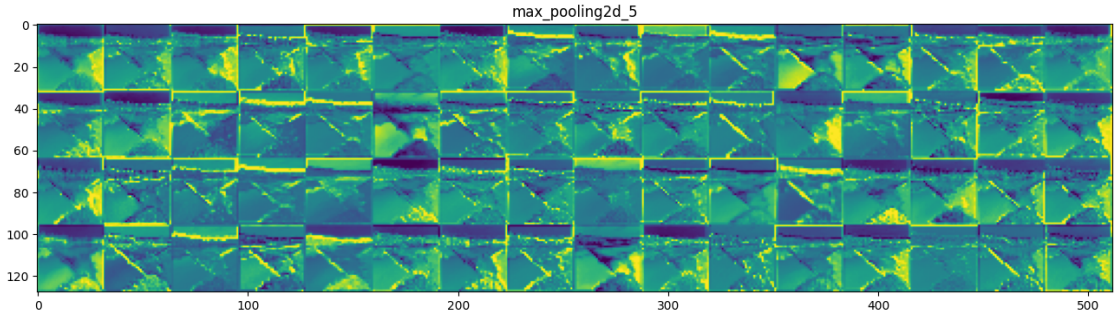
```
/tmp/ipykernel_35667/2269795348.py:24: RuntimeWarning: invalid value encountered
in divide
  channel_image /= channel_image.std()
```

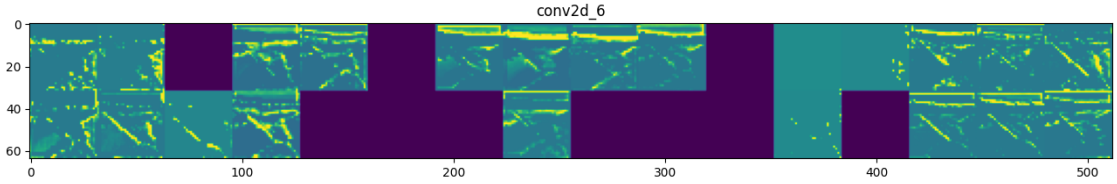
```
MemoryError                               Traceback (most recent call last)
Cell In [64], line 13
     11 size = layer_activation.shape[1] #The feature map has shape (1, size,␣
 ↪size, n_features).
     12 n_cols = n_features // images_per_row # Tiles the activation channels i␣
 ↪this matrix
---> 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 28.7 GiB for an array with shape (153664, 25088␣
 ↪and data type float64
```

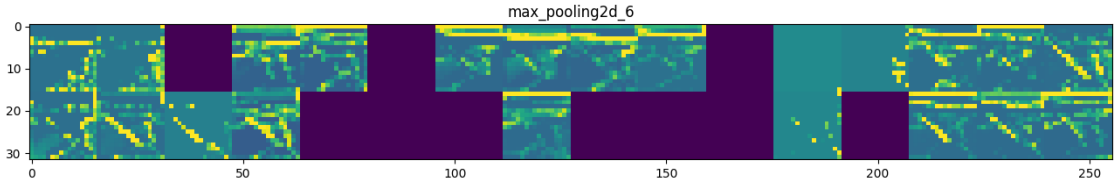


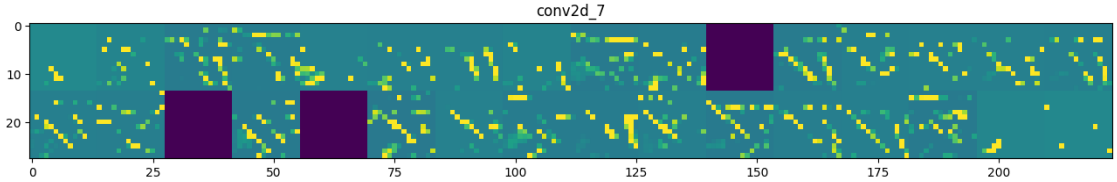


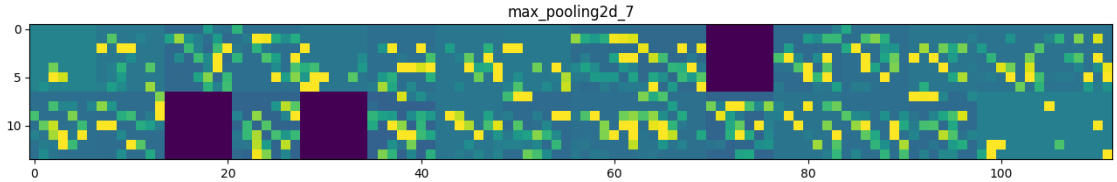



max_pooling2d_5
0
20
40
60
80
100
120
0
100
200
300
400
500


conv2d_6
0
20
40
60
0
100
200
300
400
500


max_pooling2d_6
0
10
20
30
0
50
100
150
200
250


conv2d_7
0
10
20
0
25
50
75
100
125
150
175
200


max_pooling2d_7
0
5
10
0
20
40
60
80
100