cnn_v10

October 15, 2022

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

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

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

from keras import Sequential, models, Input
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout,␣
 ↪LeakyReLU, AveragePooling2D, GlobalAveragePooling2D, BatchNormalization
from keras.optimizers import SGD, Adam
```

```
2022-10-15 20:05:43.700610: 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.
2022-10-15 20:05:44.195466: E tensorflow/stream_executor/cuda/cuda_blas.cc:2981]
Unable to register cuBLAS factory: Attempting to register factory for plugin
cuBLAS when one has already been registered
2022-10-15 20:05:45.085080: W
tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load
dynamic library 'libnvinfer.so.7'; dlerror: libnvinfer.so.7: cannot open shared
object file: No such file or directory; LD_LIBRARY_PATH:
:/home/nkspartan/miniconda3/envs/tf-gpu/lib/:/home/nkspartan/miniconda3/envs/tf-
```

```
gpu/lib/
2022-10-15 20:05:45.085286: W
tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load
dynamic library 'libnvinfer_plugin.so.7'; dlerror: libnvinfer_plugin.so.7:
cannot open shared object file: No such file or directory; LD_LIBRARY_PATH:
:/home/nkspartan/miniconda3/envs/tf-gpu/lib/:/home/nkspartan/miniconda3/envs/tf-
gpu/lib/
2022-10-15 20:05:45.085292: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Cannot
dlopen some TensorRT libraries. If you would like to use Nvidia GPU with
TensorRT, please make sure the missing libraries mentioned above are installed
properly.
```

[ ]:
```
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

2022-10-15 20:05:46.641759: 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.
2022-10-15 20:05:46.672297: 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
2022-10-15 20:05:46.716563: 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
2022-10-15 20:05:46.716747: 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
2022-10-15 20:05:47.516134: 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
2022-10-15 20:05:47.516764: 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
2022-10-15 20:05:47.516918: 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
2022-10-15 20:05:47.517053: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1616] Created device
/device:GPU:0 with 4173 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	WwCurveLineMax	WwCurveLineMean	WwCurveLineSigma
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	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	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	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	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	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')
```

0.2 Create the dataset pipeline

```
[ ]: #IMG_SIZE = 224
     IMG_SIZE = 512
     BATCH_SIZE = 32
```

```
[ ]: from glob import glob

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

       image_augmentation = Sequential([
```

```
    tf.keras.layers.RandomBrightness([-0.2,0.4], seed=seed),
    tf.keras.layers.RandomContrast(0.4, seed=seed),
    tf.keras.layers.RandomZoom(0.3, seed=seed),
    tf.keras.layers.RandomFlip('horizontal', seed=seed),
    tf.keras.layers.RandomTranslation(height_factor=0.1, width_factor=0.1,␣
↪seed=seed),
    tf.keras.layers.RandomRotation(0.1, seed=seed)
  ])

  def random_image_augmentation(image, probability=0.5):
    if np.random.random() < probability:
      return image_augmentation(image)
    return image

  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 = random_image_augmentation(image, 0.7)

    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=8)
 images_val_ds = filenames_val_ds.map(parse_image, num_parallel_calls=8)
 images_test_ds = filenames_test_ds.map(parse_image, num_parallel_calls=8)

 # 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, 10)
```

[ ]:
```
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:]
```

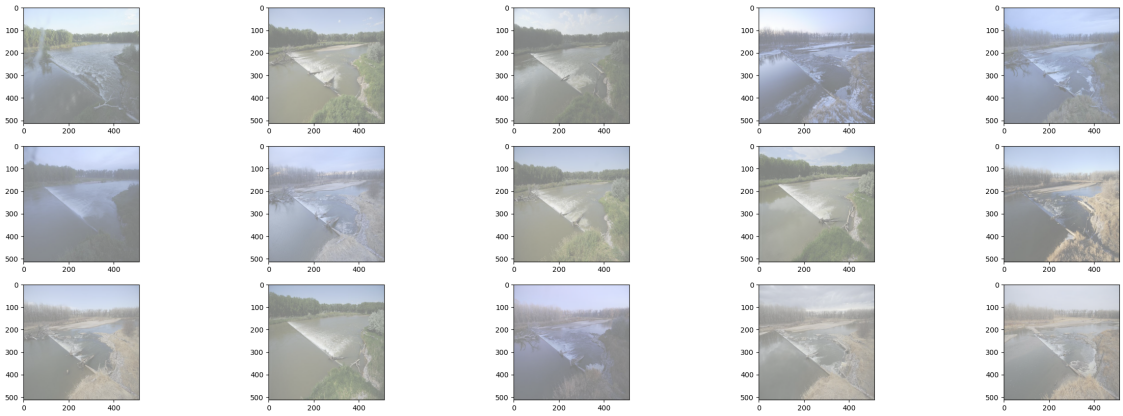
```
(32, 512, 512, 3)
(32, 2)
```

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

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

0.3 Check images

[ ]:
```
fig, ax = plt.subplots(nrows=3, ncols=5, figsize=(30, 10))

for image, stage_discharge in test_ds.take(1):
    images = image[:15]
    for img, ax in zip(images, ax.flatten()):
        img = img.numpy()
        img = img / 2 + 0.5     # unnormalize
        ax.imshow(img)

plt.show()
```



0.4 Create model

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

    if (transfer_learning == True):
        base_model = tf.keras.applications.MobileNetV2(include_top=False,
                                                        weights='imagenet',
                                                        input_shape=input_shape)
        base_model.trainable = False
        base_model._name = 'base_model_MobileNet'

        model.add(base_model)
        model.add(Dropout(0.6))
        model.add(GlobalAveragePooling2D())

        model.add(Dense(512, activation='tanh'))
        model.add(Dense(256, activation='tanh'))
        model.add(Dense(256, activation='tanh'))
        model.add(Dense(64, activation='tanh'))
    else:
        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), strides=(2, 2),␣
↪activation=LeakyReLU(), padding='same'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
        model.add(Conv2D(32, kernel_size=(3, 3), activation=LeakyReLU(0.2),␣
↪padding='same'))
        #model.add(AveragePooling2D(pool_size=(2, 2)))

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

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


        model.add(Flatten())
        model.add(Dense(128, activation='tanh'))
        model.add(Dropout(0.3))
        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], False)
```

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

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 256, 256, 64)      3136

 max_pooling2d_2 (MaxPooling  (None, 64, 64, 64)       0
 2D)

 conv2d_6 (Conv2D)           (None, 32, 32, 64)        65600

 max_pooling2d_3 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 conv2d_7 (Conv2D)           (None, 16, 16, 32)        18464

 conv2d_8 (Conv2D)           (None, 14, 14, 32)        9248

 conv2d_9 (Conv2D)           (None, 13, 13, 32)        4128

 average_pooling2d_1 (Averag  (None, 6, 6, 32)         0
```

```
ePooling2D)

flatten_1 (Flatten)          (None, 1152)              0

dense_10 (Dense)             (None, 128)               147584

dropout_2 (Dropout)          (None, 128)               0

dense_11 (Dense)             (None, 64)                8256

dense_12 (Dense)             (None, 32)                2080

dense_13 (Dense)             (None, 32)                1056

dense_14 (Dense)             (None, 2)                 66

=================================================================
Total params: 259,618
Trainable params: 259,618
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 / 200)

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
model_h = fit_model(train_ds, val_ds, epochs=20, steps=np.ceil(train_size /␣
↪BATCH_SIZE), val_steps=np.ceil(val_size / BATCH_SIZE),␣
↪callbacks=[tensorboard_callback, checkpoint_callback])
```

0.5 Evaluate model

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

```
20221015-204855
```

[ ]:
```
best_model = models.load_model(f'model_weights/{date_actual}_cnn_best_weights.
↪hdf5')
#best_model = models.load_model(f'best_models_weights/cnn_best_weights_v9.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 62ms/step - loss: 0.0030 - mse:
0.0030 - rmse: 0.0547 - mae: 0.0386 - mape: 15.8871
```

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

[ ]:
```
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 106ms/step
5.142239835017092
0.9115972227498049
pred stage: 1.9042773246765137, actual stage: 1.94
pred discharge: 30.319110870361328, actual discharge: 73.60000000000002
pred stage: 2.4251697063446045, actual stage: 2.45
pred discharge: 296.9564514160156, actual discharge: 336.0
pred stage: 3.692131757736206, actual stage: 3.72
pred discharge: 1960.296630859375, actual discharge: 2050.0
pred stage: 3.468950033187866, actual stage: 3.43
pred discharge: 1495.0885009765625, actual discharge: 1440.0
pred stage: 2.826493263244629, actual stage: 2.83
pred discharge: 753.9630737304688, actual discharge: 704.0
pred stage: 2.2473175525665283, actual stage: 2.25
pred discharge: 196.7310333251953, actual discharge: 214.0
pred stage: 2.5455737113952637, actual stage: 2.58
pred discharge: 385.1839904785156, actual discharge: 403.0
pred stage: 2.2392189502716064, actual stage: 2.27
pred discharge: 177.2609405517578, actual discharge: 206.0
pred stage: 2.4840328693389893, actual stage: 2.46
pred discharge: 334.6043395996094, actual discharge: 315.0
pred stage: 2.1623551845550537, actual stage: 2.17
pred discharge: 131.10499572753906, actual discharge: 169.0
```

0.5.1 Residual analysis

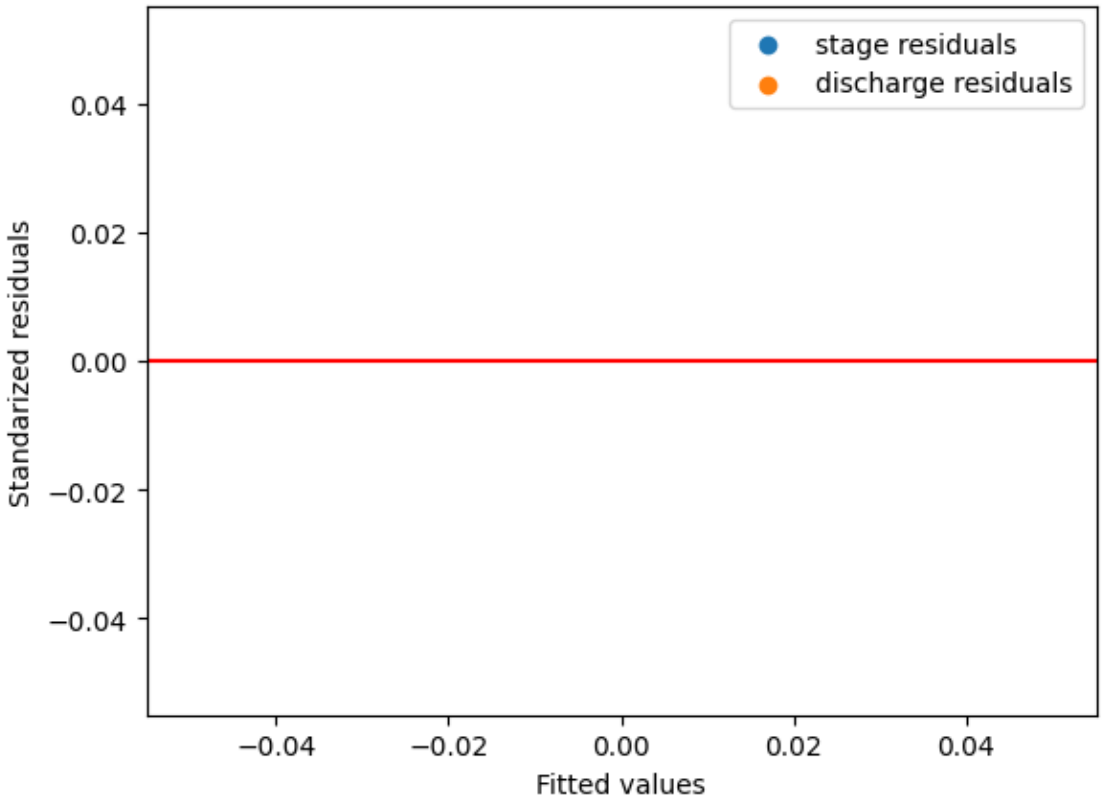
[ ]:
```
y_predictions = np.empty(shape=(1, 2))
y_real = np.empty(shape=(1, 2))

"""for image, stage_discharge in test_ds.take(100):
```

```
    y_predictions = np.concatenate((y_predictions, best_model.predict(x=image)))
    y_real = np.concatenate((y_real, stage_discharge.numpy()))"""
```

[ ]:
```
'for image, stage_discharge in test_ds.take(100):\n    y_predictions =
np.concatenate((y_predictions, best_model.predict(x=image)))\n    y_real =
np.concatenate((y_real, stage_discharge.numpy()))'
```

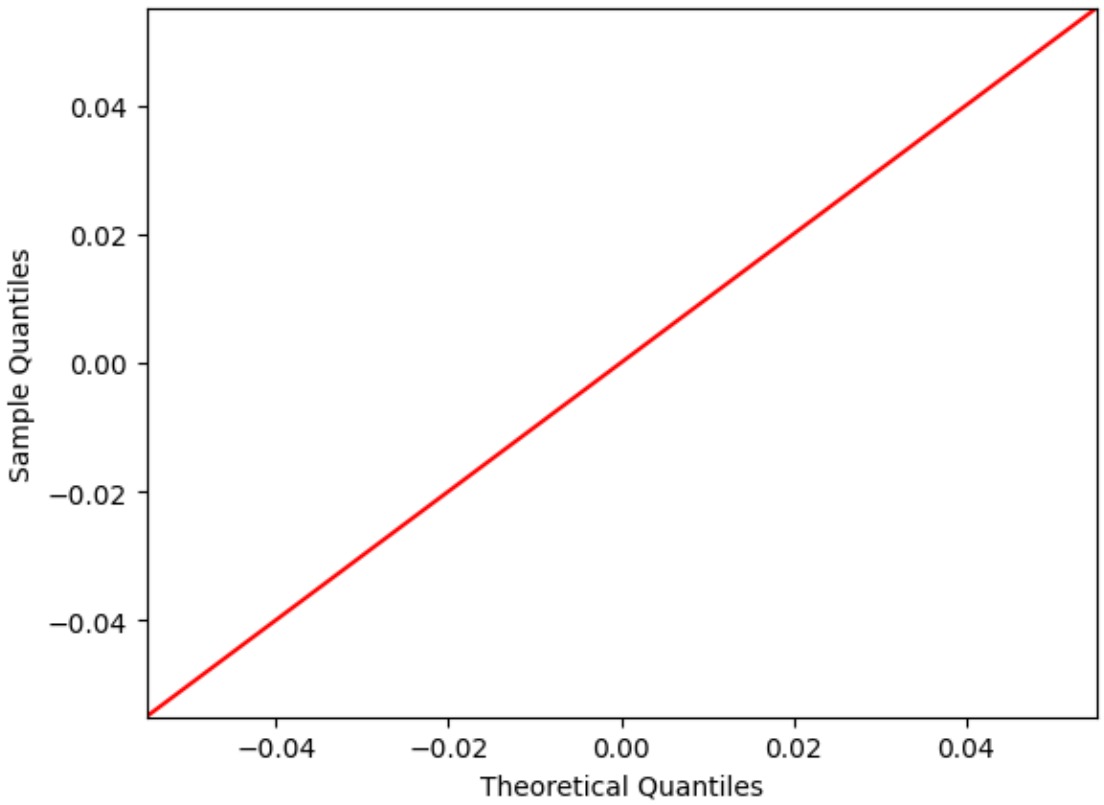
[ ]:
```
residuals = y_real - y_predictions
residuals_std = residuals/residuals.std()

y_real_stage = np.array([i[0] for i in y_real])
residual_stage = np.array([i[0] for i in residuals])

y_real_discharge = np.array([i[1] for i in y_real])
residual_discharge = np.array([i[1] for i in residuals])

plt.scatter(y_real_stage, residual_stage / residual_stage.std(), label="stage␣
 ↪residuals")
plt.scatter(y_real_discharge, residual_discharge / residual_discharge.std(),␣
 ↪label="discharge residuals")
plt.axhline(y=0.0, color='r', linestyle='-')
plt.xlabel("Fitted values")
plt.ylabel("Standarized residuals")

plt.legend()
plt.show()
```

```
/tmp/ipykernel_67186/3264406076.py:10: RuntimeWarning: divide by zero
encountered in divide
  plt.scatter(y_real_stage, residual_stage / residual_stage.std(), label="stage
residuals")
/tmp/ipykernel_67186/3264406076.py:11: RuntimeWarning: divide by zero
encountered in divide
  plt.scatter(y_real_discharge, residual_discharge / residual_discharge.std(),
label="discharge residuals")
```



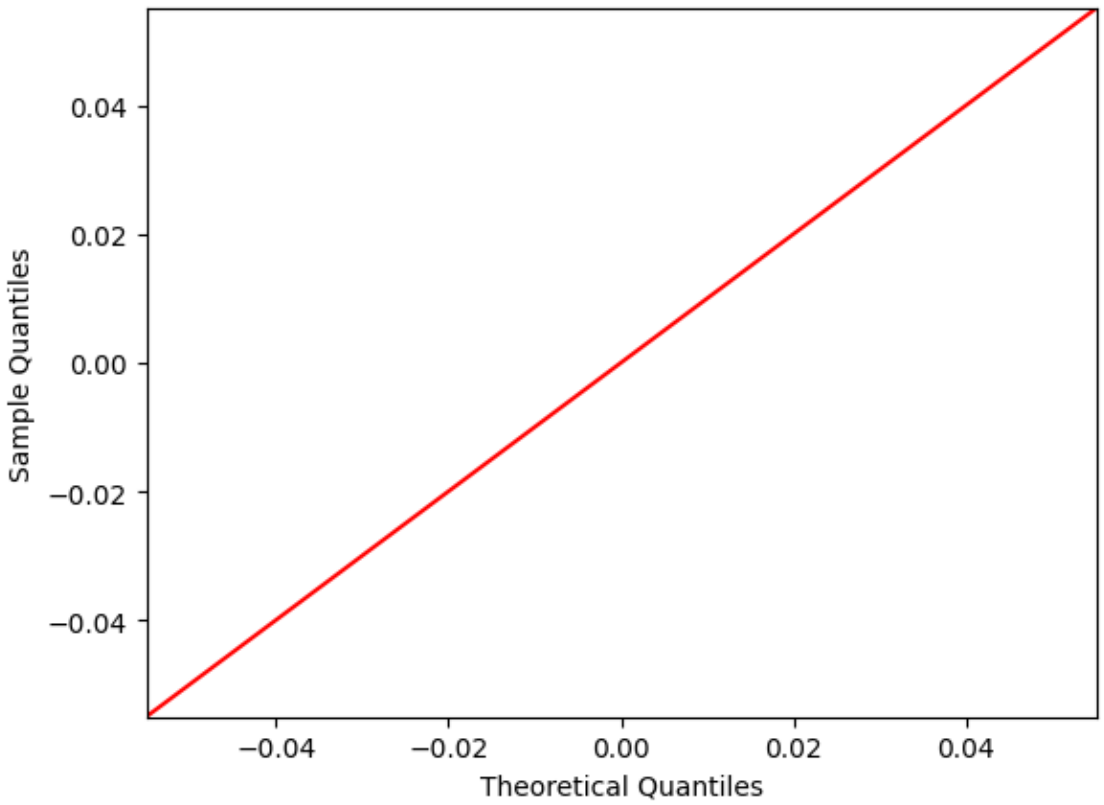
[ ]:
```
import statsmodels.api as sm
from statsmodels.stats.diagnostic import normal_ad

figure = sm.qqplot(residual_stage / residual_stage.std(), line ='45',␣
↪label='stage')
plt.show()
```

```
/tmp/ipykernel_67186/3505247562.py:4: RuntimeWarning: divide by zero encountered
in divide
  figure = sm.qqplot(residual_stage / residual_stage.std(), line ='45',
label='stage')
```



[ ]:
```
figure = sm.qqplot(residual_discharge / residual_discharge.std(), line='45',␣
↪label='discharge')
plt.show()
```

```
/tmp/ipykernel_67186/1192211388.py:1: RuntimeWarning: divide by zero encountered
in divide
  figure = sm.qqplot(residual_discharge / residual_discharge.std(), line='45',
label='discharge')
```



[ ]:
```
import seaborn as sns

#sns.histplot(residuals, kde=True, bins = 10)
```

[ ]:
```
stat, pval = normal_ad(residual_stage)
print("p-value:", pval)

if pval<0.05:
    print("Hay evidencia de que los residuos no provienen de una distribución␣
 ↪normal.")
else:
    print("No hay evidencia para rechazar la hipótesis de que los residuos␣
 ↪vienen de una distribución normal.")
```

```
p-value: 0.0
Hay evidencia de que los residuos no provienen de una distribución normal.

/home/nkspartan/miniconda3/envs/tf-gpu/lib/python3.10/site-
packages/numpy/core/_methods.py:265: RuntimeWarning: Degrees of freedom <= 0 for
slice
  ret = _var(a, axis=axis, dtype=dtype, out=out, ddof=ddof,
/home/nkspartan/miniconda3/envs/tf-gpu/lib/python3.10/site-
```

```
packages/numpy/core/_methods.py:257: RuntimeWarning: invalid value encountered
in double_scalars
  ret = ret.dtype.type(ret / rcount)
```

[ ]:
```
stat, pval = normal_ad(residual_discharge)
print("p-value:", pval)

if pval < 0.05:
    print("Hay evidencia de que los residuos no provienen de una distribución␣
 ↪normal.")
else:
    print("No hay evidencia para rechazar la hipótesis de que los residuos␣
 ↪vienen de una distribución normal.")
```

```
p-value: 0.0
Hay evidencia de que los residuos no provienen de una distribución normal.
```

0.6 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))
```

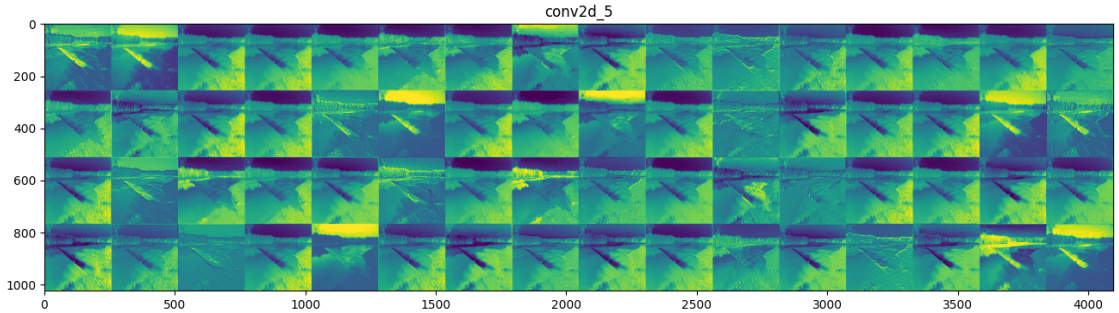
```
1/1 [==============================] - 0s 174ms/step
```

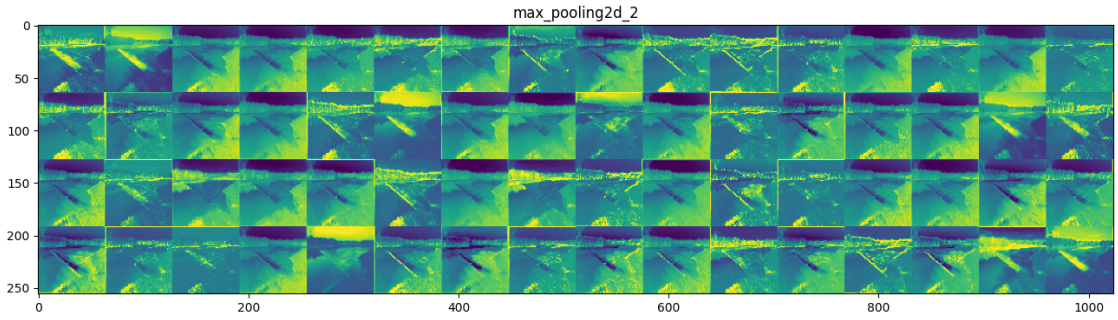
[ ]:
```
import matplotlib.pyplot as plt

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))
```

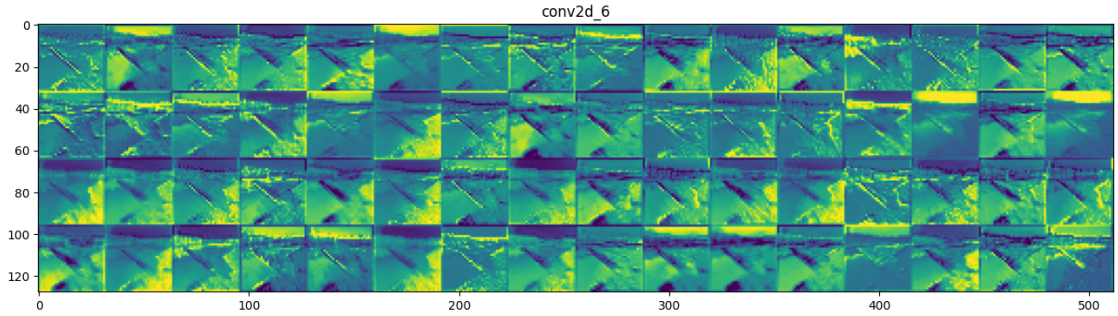
```
    print(layer_name)
    if "flatten" in layer_name or "dense" in layer_name: break

    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')
```

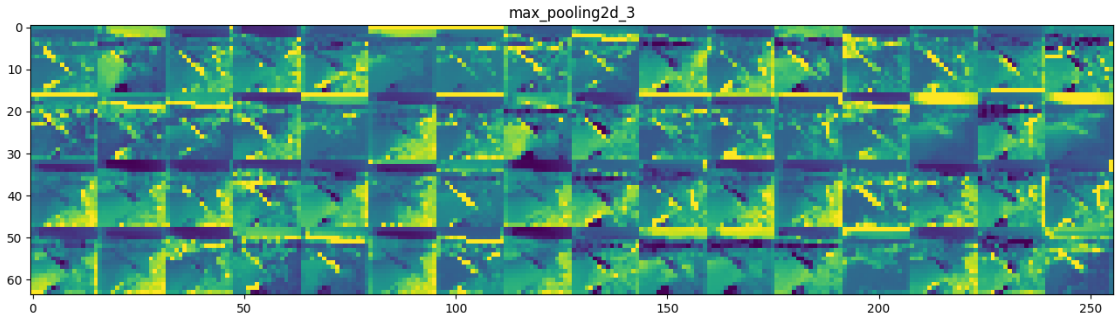
```
conv2d_5
max_pooling2d_2
conv2d_6
max_pooling2d_3
conv2d_7
conv2d_8
conv2d_9
average_pooling2d_1
flatten_1
```

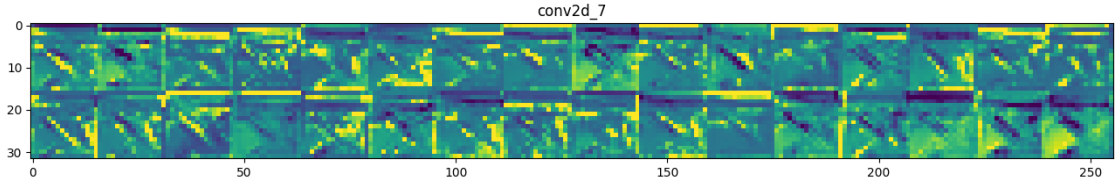
```
/tmp/ipykernel_67186/1031377702.py:24: RuntimeWarning: invalid value encountered
in divide
  channel_image /= channel_image.std()
```

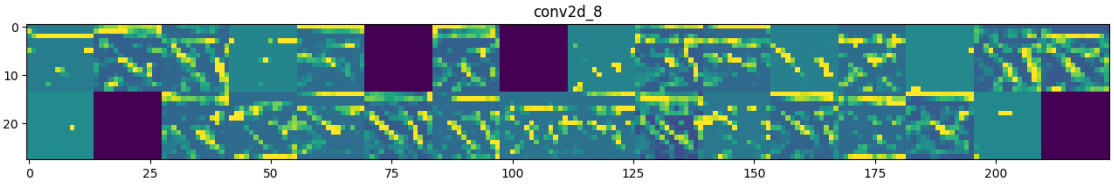


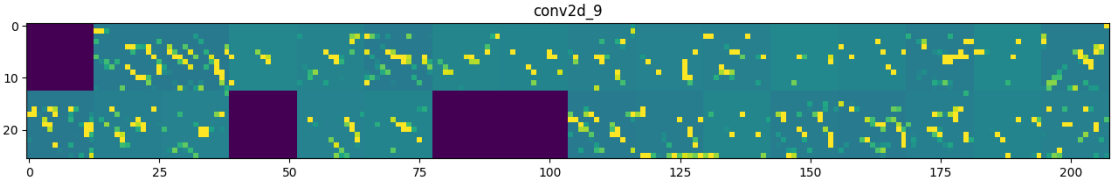

max_pooling2d_2
0
50
100
150
200
250
0
200
400
600
800
1000

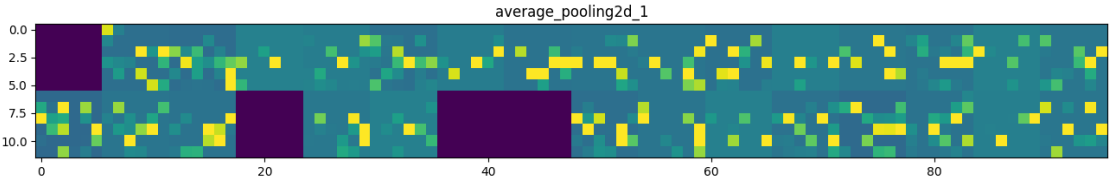

conv2d_6
0
20
40
60
80
100
120
0
100
200
300
400
500


max_pooling2d_3
0
10
20
30
40
50
60
0
50
100
150
200
250


conv2d_7
0
10
20
30
0
50
100
150
200
250


conv2d_8
0
10
20
0
25
50
75
100
125
150
175
200


conv2d_9
0
10
20
0
25
50
75
100
125
150
175
200


average_pooling2d_1
0.0
2.5
5.0
7.5
10.0
0
20
40
60
80