cnn_v12

November 4, 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/
```

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

```
$LD_LIBRARY_PATH:$CONDA_PREFIX/lib/
```

[ ]:
```
import tensorflow as tf
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt

from tensorflow.keras import Sequential, models, Input
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D,␣
↪Dropout, LeakyReLU, AveragePooling2D, GlobalAveragePooling2D,␣
↪BatchNormalization, TimeDistributed, LSTM, SpatialDropout2D
from tensorflow.keras.optimizers import SGD, Adam
```

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

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

```
Default GPU Device: /device:GPU:0
Metal device set to: Apple M1 Pro

systemMemory: 16.00 GB
maxCacheSize: 5.33 GB


2022-11-03 13:01:40.906763: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:306]
Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel
may not have been built with NUMA support.
2022-11-03 13:01:40.906921: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:272]
```

```
Created TensorFlow device (/device:GPU:0 with 0 MB memory) -> physical
PluggableDevice (device: 0, name: METAL, pci bus id: <undefined>)
```

[ ]:
```
physical_devices = tf.config.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(physical_devices[0], True)
```

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", 'SensorTime']]
```

```
[ ]: df['SensorTime'] = pd.to_datetime(df['SensorTime'])
     df['Year'] = df['SensorTime'].dt.year
     df.head()
```

```
[ ]:                                 Filename  Stage  Discharge  \
     0  StateLineWeir_20120609_Farrell_001.jpg   2.99      916.0
     1  StateLineWeir_20120609_Farrell_002.jpg   2.99      916.0
     2  StateLineWeir_20120609_Farrell_003.jpg   2.96      873.0
     3  StateLineWeir_20120609_Farrell_004.jpg   2.94      846.0
     4  StateLineWeir_20120609_Farrell_005.jpg   2.94      846.0

                SensorTime  Year
     0 2012-06-09 13:15:00  2012
     1 2012-06-09 13:15:00  2012
     2 2012-06-09 13:45:00  2012
     3 2012-06-09 14:45:00  2012
     4 2012-06-09 15:45:00  2012
```

0.1.1 Remove outliers

```
[ ]: df = df[df.Stage > 0]
     df = df[df.Discharge > 0]
```

We consider values equal to 0 as outliers because from the photos it doesn't seem that it would be possible that at this time we would have a value of 0 for stage or discharge

```
[ ]: df.shape
```

```
[ ]: (40148, 5)
```

0.1.2 Scale the data

```
[ ]: from sklearn.preprocessing import StandardScaler
     from joblib import load

     scaler = StandardScaler()
     #scaler = load('std_scaler.joblib') # scaler with all the 42059 observations
```

Scale the data based only on the training dataset (in this case the training dataset is from 2012 to 2016)

```
[ ]: data_to_scale_fit = df[(df["Year"] >= 2012) & (df["Year"] <= 2016)][["Stage",␣
     ↪"Discharge"]]
```

```
data_to_scale_fit
```

[ ]:
```
       Stage  Discharge
0       2.99      916.0
1       2.99      916.0
2       2.96      873.0
3       2.94      846.0
4       2.94      846.0
…       …         …
21416   2.38      279.0
21417   2.38      279.0
21418   2.38      279.0
21419   2.38      279.0
21420   2.38      279.0

[20304 rows x 2 columns]
```

[ ]:
```
scaler.fit(data_to_scale_fit)
```

[ ]:
```
StandardScaler()
```

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

[ ]:
```
                                    Filename     Stage  Discharge  \
0      StateLineWeir_20120609_Farrell_001.jpg  0.077964  -0.136077
1      StateLineWeir_20120609_Farrell_002.jpg  0.077964  -0.136077
2      StateLineWeir_20120609_Farrell_003.jpg  0.045759  -0.165451
3      StateLineWeir_20120609_Farrell_004.jpg  0.024290  -0.183894
4      StateLineWeir_20120609_Farrell_005.jpg  0.024290  -0.183894
…                                        …       …         …
42054  StateLineWeir_20191011_Farrell_409.jpg -0.405103  -0.465332
42055  StateLineWeir_20191011_Farrell_410.jpg -0.405103  -0.465332
42056  StateLineWeir_20191011_Farrell_411.jpg -0.405103  -0.465332
42057  StateLineWeir_20191011_Farrell_412.jpg -0.405103  -0.465332
42058  StateLineWeir_20191011_Farrell_413.jpg -0.405103  -0.465332

                SensorTime  Year
0      2012-06-09 13:15:00  2012
1      2012-06-09 13:15:00  2012
2      2012-06-09 13:45:00  2012
3      2012-06-09 14:45:00  2012
4      2012-06-09 15:45:00  2012
…                      …     …
42054 2019-10-11 09:00:00  2019
42055 2019-10-11 10:00:00  2019
42056 2019-10-11 11:00:00  2019
```

```
42057 2019-10-11 12:00:00  2019
42058 2019-10-11 12:45:00  2019

[40148 rows x 5 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
FRAMES = 10
```

[ ]:
```
from dataset_transformer import make_dataset
```

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

with tf.device("/gpu:0"):
    train_ds, train_size, val_ds, val_size, test_ds, test_size =␣
 ↪make_dataset(path, BATCH_SIZE, IMG_SIZE, FRAMES, df, 10, True, "cnn/lstm")
```

```
2022-11-03 13:01:41.907344: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:306]
Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel
may not have been built with NUMA support.
2022-11-03 13:01:41.907363: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:272]
Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0
MB memory) -> physical PluggableDevice (device: 0, name: METAL, pci bus id:
<undefined>)
```

```
20304
7117
12727
```

[ ]:
```
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:]
```

```
2022-11-03 13:04:25.787680: W
tensorflow/core/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU
frequency: 0 Hz
```

```
(32, 10, 224, 224, 3)
(32, 1, 2)
```

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

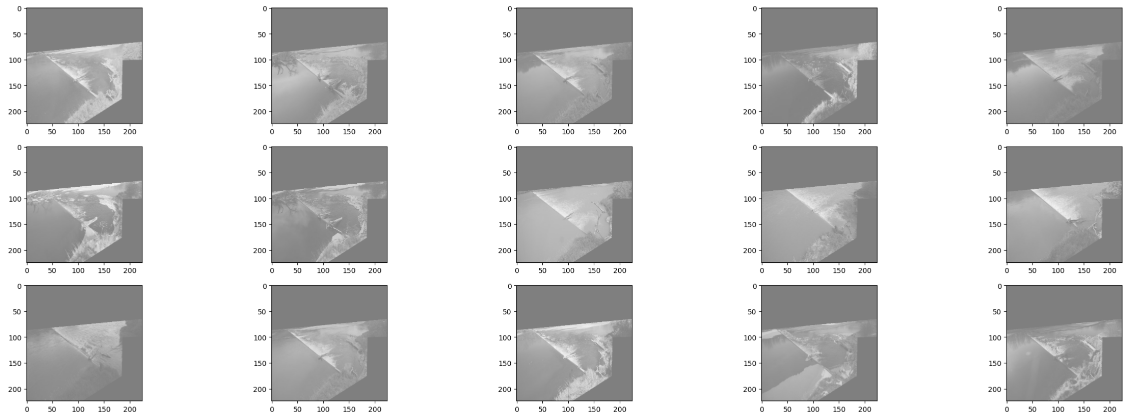
```
(10, 224, 224, 3)
(1, 2)
```

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()[0]
        #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, option="normal"):
    model = Sequential()

    if option == "transfer":
```

```python
        base_model = tf.keras.applications.ResNet50V2(include_top=False,
                                                        weights='imagenet',
                                                        input_shape=input_shape)
        base_model.trainable = False
        base_model._name = 'base_model_ResNet50'

        model.add(base_model)
        model.add(Dropout(0.3))
        model.add(GlobalAveragePooling2D())

        model.add(Dense(512, activation='elu'))
        model.add(Dense(512, activation='elu'))
        model.add(Dense(256, activation='elu'))
        model.add(Dense(128, activation='elu'))
    elif option == "normal":
        model.add(Input(shape=input_shape))

        """model.add(Conv2D(16, kernel_size=(3, 3), activation="elu",␣
↪padding='same', kernel_initializer='he_uniform'))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        model.add(BatchNormalization())

        model.add(Conv2D(32, kernel_size=(3, 3), activation="elu",␣
↪padding='same', kernel_initializer='he_uniform'))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        model.add(BatchNormalization())

        model.add(Conv2D(32, kernel_size=(3, 3), activation="elu",␣
↪padding='same', kernel_initializer='he_uniform'))
        model.add(Conv2D(32, kernel_size=(3, 3), activation="elu",␣
↪padding='same', kernel_initializer='he_uniform'))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        model.add(BatchNormalization())

        model.add(Conv2D(64, kernel_size=(4, 4), activation="elu",␣
↪padding='same', kernel_initializer='he_uniform'))
        model.add(Conv2D(64, kernel_size=(4, 4), activation="elu",␣
↪padding='same', kernel_initializer='he_uniform'))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        model.add(BatchNormalization())

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

```
        model.add(BatchNormalization())

        model.add(Conv2D(64, kernel_size=(3, 3), activation="elu",␣
↪padding='same', kernel_initializer='he_uniform'))
        model.add(Conv2D(64, kernel_size=(3, 3), activation="elu",␣
↪padding='same', kernel_initializer='he_uniform'))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        model.add(BatchNormalization())

        model.add(GlobalAveragePooling2D())

        model.add(Dense(512, activation='elu'))
        model.add(Dropout(0.3))
        model.add(Dense(512, activation='elu'))
        model.add(Dropout(0.3))
        model.add(Dense(256, activation='elu'))
        model.add(Dense(64, activation='elu'))"""

        model.add(Conv2D(32, kernel_size=(4, 4), strides=(2, 2),␣
↪padding='same', activation="elu"))
        model.add(MaxPooling2D(pool_size=(2, 2)))

        model.add(Conv2D(32, kernel_size=(4, 4), strides=(2, 2),␣
↪activation="elu", padding='same'))
        model.add(MaxPooling2D(pool_size=(2, 2)))

        model.add(Conv2D(64, kernel_size=(3, 3), activation="elu",␣
↪padding='same'))
        #model.add(AveragePooling2D(pool_size=(2, 2)))

        model.add(Conv2D(64, kernel_size=(3, 3), activation='elu'))

        model.add(Conv2D(64, kernel_size=(2, 2), activation='elu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))

        model.add(Conv2D(64, kernel_size=(3, 3), activation='elu'))

        model.add(Conv2D(64, kernel_size=(2, 2), activation='elu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))

        model.add(Flatten())
        model.add(Dense(256, activation='tanh'))
        model.add(Dropout(0.3))
        model.add(Dense(128, activation='tanh'))
        model.add(Dense(64, activation='tanh'))
        model.add(Dense(32, activation='tanh'))
    elif option == "cnn/lstm":
```

```
        base_model = tf.keras.applications.ResNet50V2(include_top=False,
                                                      weights='imagenet',
                                                      input_shape=(224, 224, 3))
        base_model.trainable = False
        base_model._name = 'base_model_ResNet50'
        model.add(Input(shape=input_shape))
        model.add(TimeDistributed(base_model))
        model.add(TimeDistributed(Dropout(0.3)))
        model.add(TimeDistributed(GlobalAveragePooling2D()))

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

        model.add(TimeDistributed(Conv2D(16, kernel_size=(4, 4), strides=(2,␣
↪2), padding='same', activation='elu')))
        model.add(TimeDistributed(MaxPooling2D(pool_size=(3, 3))))

        model.add(TimeDistributed(Conv2D(32, kernel_size=(4, 4), strides=(2,␣
↪2), activation='elu', padding='same')))
        model.add(TimeDistributed(MaxPooling2D(pool_size=(2, 2))))

        model.add(TimeDistributed(Conv2D(32, kernel_size=(3, 3),␣
↪activation='elu', padding='same')))
        model.add(TimeDistributed(MaxPooling2D(pool_size=(2, 2))))

        model.add(TimeDistributed(Conv2D(32, kernel_size=(3, 3),␣
↪activation='elu', padding='same')))
        model.add(TimeDistributed(MaxPooling2D(pool_size=(2, 2))))


        model.add(TimeDistributed(Flatten()))"""

        model.add(LSTM(10, return_sequences=True))
        model.add(LSTM(15))

        model.add(Dense(512, activation='elu'))
        model.add(Dense(256, activation='elu'))
        model.add(Dense(128, activation='elu'))
        model.add(Dense(128, activation='elu'))

    model.add(Dense(output_shape, activation='linear')) # linear regression␣
↪output layer

    return model
```

[ ]:
```
#model = create_model(input_shape, output_shape[0], "normal")
model = create_model(input_shape, output_shape[1], "cnn/lstm")
```

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

```
Model: "sequential_2"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 time_distributed_9 (TimeDis  (None, 10, 7, 7, 2048)   23564800
 tributed)

 time_distributed_10 (TimeDi  (None, 10, 7, 7, 2048)   0
 stributed)

 time_distributed_11 (TimeDi  (None, 10, 2048)         0
 stributed)

 lstm_2 (LSTM)               (None, 10, 10)            82360

 lstm_3 (LSTM)               (None, 15)                1560

 dense_5 (Dense)             (None, 512)               8192

 dense_6 (Dense)             (None, 256)               131328

 dense_7 (Dense)             (None, 128)               32896

 dense_8 (Dense)             (None, 128)               16512

 dense_9 (Dense)             (None, 2)                 258

=================================================================
Total params: 23,837,906
Trainable params: 273,106
Non-trainable params: 23,564,800
_________________________________________________________________
```

[ ]:
```
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-3, momentum=0.9, nesterov=True)
#adam = Adam(learning_rate=1e-3, decay=1e-3 / 200)
adam = Adam(learning_rate=1e-2, decay=1e-2 / 200)

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

[ ]:
```python
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)
```

[ ]:
```python
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)

es_callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', mode='min',␣
↪verbose=1, patience=5)

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

[ ]:
```python
# batch_size = 0 because we already have batch size in tf dataset
with tf.device("/gpu:0"):
    model_h = fit_model(train_ds, val_ds, epochs=60, steps=np.ceil(train_size /␣
↪BATCH_SIZE), val_steps=np.ceil(val_size / BATCH_SIZE),␣
↪callbacks=[tensorboard_callback, checkpoint_callback, es_callback])
```

0.5 Evaluate model

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

```
Running cells with 'Python 3.9.13 ('tf-metal')' requires ipykernel package.

Run the following command to install 'ipykernel' into the Python environment.

Command: 'conda install -n tf-metal ipykernel --update-deps --force-reinstall'
```

[ ]:
```python
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')
```

```
Running cells with 'Python 3.9.13 ('tf-metal')' requires ipykernel package.

Run the following command to install 'ipykernel' into the Python environment.

Command: 'conda install -n tf-metal ipykernel --update-deps --force-reinstall'
```

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

```
Running cells with 'Python 3.9.13 ('tf-metal')' requires ipykernel package.

Run the following command to install 'ipykernel' into the Python environment.

Command: 'conda install -n tf-metal ipykernel --update-deps --force-reinstall'
```

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

```
Running cells with 'Python 3.9.13 ('tf-metal')' requires ipykernel package.

Run the following command to install 'ipykernel' into the Python environment.

Command: 'conda install -n tf-metal ipykernel --update-deps --force-reinstall'
```

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

```
Running cells with 'Python 3.9.13 ('tf-metal')' requires ipykernel package.

Run the following command to install 'ipykernel' into the Python environment.

Command: 'conda install -n tf-metal ipykernel --update-deps --force-reinstall'
```

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

        stage_discharge_test_values = stage_discharge.numpy()
        predictions_values = predictions
```

```
    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]
    stage_discharge_test_values = stage_discharge_test_values.numpy().
↪reshape(10, 2)
    predictions_values = predictions[:10]

    for i in range(len(stage_discharge_test_values)):
        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]}")
```

```
Running cells with 'Python 3.9.13 ('tf-metal')' requires ipykernel package.

Run the following command to install 'ipykernel' into the Python environment.

Command: 'conda install -n tf-metal ipykernel --update-deps --force-reinstall'
```

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()))"""
```

```
Running cells with 'Python 3.9.13 ('tf-metal')' requires ipykernel package.
```

```
Run the following command to install 'ipykernel' into the Python environment.

Command: 'conda install -n tf-metal ipykernel --update-deps --force-reinstall'
```

[ ]:
```python
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()
```

```
Running cells with 'Python 3.9.13 ('tf-metal')' requires ipykernel package.

Run the following command to install 'ipykernel' into the Python environment.

Command: 'conda install -n tf-metal ipykernel --update-deps --force-reinstall'
```

[ ]:
```python
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()
```

```
Running cells with 'Python 3.9.13 ('tf-metal')' requires ipykernel package.

Run the following command to install 'ipykernel' into the Python environment.

Command: 'conda install -n tf-metal ipykernel --update-deps --force-reinstall'
```

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

```
Running cells with 'Python 3.9.13 ('tf-metal')' requires ipykernel package.

Run the following command to install 'ipykernel' into the Python environment.

Command: 'conda install -n tf-metal ipykernel --update-deps --force-reinstall'
```

[ ]:
```
import seaborn as sns

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

```
Running cells with 'Python 3.9.13 ('tf-metal')' requires ipykernel package.

Run the following command to install 'ipykernel' into the Python environment.

Command: 'conda install -n tf-metal ipykernel --update-deps --force-reinstall'
```

[ ]:
```
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.")
```

```
Running cells with 'Python 3.9.13 ('tf-metal')' requires ipykernel package.

Run the following command to install 'ipykernel' into the Python environment.

Command: 'conda install -n tf-metal ipykernel --update-deps --force-reinstall'
```

[ ]:
```
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.")
```

```
Running cells with 'Python 3.9.13 ('tf-metal')' requires ipykernel package.

Run the following command to install 'ipykernel' into the Python environment.

Command: 'conda install -n tf-metal ipykernel --update-deps --force-reinstall'
```

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
```

```
Running cells with 'Python 3.9.13 ('tf-metal')' requires ipykernel package.

Run the following command to install 'ipykernel' into the Python environment.

Command: 'conda install -n tf-metal ipykernel --update-deps --force-reinstall'
```

[ ]:
```
activations = activation_model.predict(test_ds.take(1))
```

```
Running cells with 'Python 3.9.13 ('tf-metal')' requires ipykernel package.

Run the following command to install 'ipykernel' into the Python environment.

Command: 'conda install -n tf-metal ipykernel --update-deps --force-reinstall'
```

[ ]:
```
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
```

```python
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')
```

```
Running cells with 'Python 3.9.13 ('tf-metal')' requires ipykernel package.

Run the following command to install 'ipykernel' into the Python environment.

Command: 'conda install -n tf-metal ipykernel --update-deps --force-reinstall'
```