Preliminary CNN Training and Analysis

This is a brief example of the methodology used throughout the CNN training and analysis part of this project.

Import packages

```
import numpy as np
import h5py
import glob
import re
import tensorflow as tf

import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from scipy.stats import pearsonr

np.set_printoptions(precision=3, suppress=True)
```

```
2024-02-14 14:08:18.312814: E tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:9342]
2024-02-14 14:08:18.312841: E tensorflow/compiler/xla/stream_executor/cuda/cuda_fft.cc:609] U
2024-02-14 14:08:18.312870: E tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:1518]
2024-02-14 14:08:18.319367: I tensorflow/core/platform/cpu_feature_guard.cc:182] This Tensor
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with
2024-02-14 14:08:19.589028: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT W
```

Setup GPU

First, follow instructions here, or alternatively run:

```
for a in /sys/bus/pci/devices/*; do echo 0 | sudo tee -a $a/numa_node; done
```

We do this as a workaround for this error:

```
gpu_devices = tf.config.experimental.list_physical_devices('GPU')
for device in gpu_devices:
    tf.config.experimental.set_memory_growth(device, True)
print(tf.config.list_physical_devices('GPU'), tf.test.gpu_device_name())
```

```
[]
```

```
2024-02-14 14:08:25.970865: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor
2024-02-14 14:08:26.349400: W tensorflow/core/common_runtime/gpu/gpu_device.cc:2211] Cannot d
Skipping registering GPU devices...
2024-02-14 14:08:26.353220: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor
2024-02-14 14:08:26.353403: W tensorflow/core/common_runtime/gpu/gpu_device.cc:2211] Cannot d
Skipping registering GPU devices...
```

Import and prepare data

```
def extract_floats(string):
    return re.findall(r"[-+]?\d*\.\d+|\d+", string)

def data_load():
    density = 0.15
    files = glob.glob(f"../data/dataset_tumble_*_0.15.h5") #imports all tumbling rates for
    inputs,outputs = [],[]
    for f in files:
        tumble = float(extract_floats(f)[0])
        with h5py.File(f, "r") as fin:
          count = 0
          for key in fin.keys():
              img = fin[key][:]
              img = img.reshape((img.shape[0], img.shape[1],1))
              shape = img.shape
              inputs.append(img)
              outputs.append(tumble)
              count+=1
```

```
    # Scramble the dataset
    order = np.arange(len(outputs)).astype(int)
    order = np.random.permutation(order)
    return np.array(inputs)[order],np.array(outputs)[order],shape
```

```
x,y,shape = data_load()
```

```
print("Number of unique alpha: ", len(np.unique(y)))
print("Shape of x: ", np.shape(x))
print("Shape of y: ", np.shape(y))
```

```
Number of unique alpha:  5
Shape of x:  (70000, 128, 128, 1)
Shape of y:  (70000,)
```

We have 10000 * number of unique alpha snapshots total, we split them into a training set and a validation set:

```
last = 20000
x_train, y_train = x[:-last], y[:-last]
x_val,y_val = x[-last:],y[-last:]

print("Size of training data: ", len(x_train))
print("Size of validation data: ", len(x_val))
```

```
Size of training data:  50000
Size of validation data:  20000
```

Setup and train our model

```
from tensorflow import keras
from keras import backend as K
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Conv2D,Flatten,Dropout,MaxPooling2D,BatchNormali

import contextlib
```

```
@contextlib.contextmanager
def options(options):
  old_opts = tf.config.optimizer.get_experimental_options()
  tf.config.optimizer.set_experimental_options(options)
  try:
    yield
  finally:
    tf.config.optimizer.set_experimental_options(old_opts)
```

Run this after analysis to reset model and release RAM before changing the architecture

```
import gc

K.clear_session()
del prediction
del model
del history

print("Collected: ", gc.collect())
```

Setting up the model's architecture

```
model = Sequential()

model.add(Conv2D(filters=3, kernel_size=(3,3), padding='same', strides=(3,3), activation='
model.add(BatchNormalization())
model.add(Conv2D(filters=3, kernel_size=(3,3), padding='same', input_shape=shape))
model.add(BatchNormalization())

model.add(MaxPooling2D(pool_size=(3, 3)))

#model.add(AveragePooling2D(pool_size=(2,2), strides=(2,2)))

model.add(Conv2D(filters=6, kernel_size=(3,3), padding='same'))
model.add(BatchNormalization())

model.add(Conv2D(filters=6, kernel_size=(3,3), padding='same'))
model.add(BatchNormalization())
```

```python
model.add(MaxPooling2D(pool_size=(3, 3)))

model.add(Dense(units=128, activation='relu'))

with options({"layout_optimizer": False}):
    model.add(Dropout(0.2))
model.add(Dense(units=10, activation='softmax'))

model.add(Flatten())
model.add(Dense(units=1, activation='linear'))
```

```python
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 43, 43, 3)         30

 batch_normalization (Batch  (None, 43, 43, 3)         12
 Normalization)

 conv2d_1 (Conv2D)           (None, 43, 43, 3)         84

 batch_normalization_1 (Bat  (None, 43, 43, 3)         12
 chNormalization)

 max_pooling2d (MaxPooling2  (None, 14, 14, 3)         0
 D)

 conv2d_2 (Conv2D)           (None, 14, 14, 6)         168

 batch_normalization_2 (Bat  (None, 14, 14, 6)         24
 chNormalization)

 conv2d_3 (Conv2D)           (None, 14, 14, 6)         330

 batch_normalization_3 (Bat  (None, 14, 14, 6)         24
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 4, 4, 6)           0
```

```
 g2D)

 dense (Dense)               (None, 4, 4, 128)         896

 dropout (Dropout)           (None, 4, 4, 128)         0

 dense_1 (Dense)             (None, 4, 4, 10)          1290

 flatten (Flatten)           (None, 160)               0

 dense_2 (Dense)             (None, 1)                 161

=================================================================
Total params: 3031 (11.84 KB)
Trainable params: 2995 (11.70 KB)
Non-trainable params: 36 (144.00 Byte)
_________________________________________________________________
```

Optimizer

```
optimizer = keras.optimizers.Adam(learning_rate=0.001)
model.compile(loss='mean_absolute_error', optimizer=optimizer, metrics=['accuracy'])
```

Training and evaluation

Before training, these are the “predictions”:

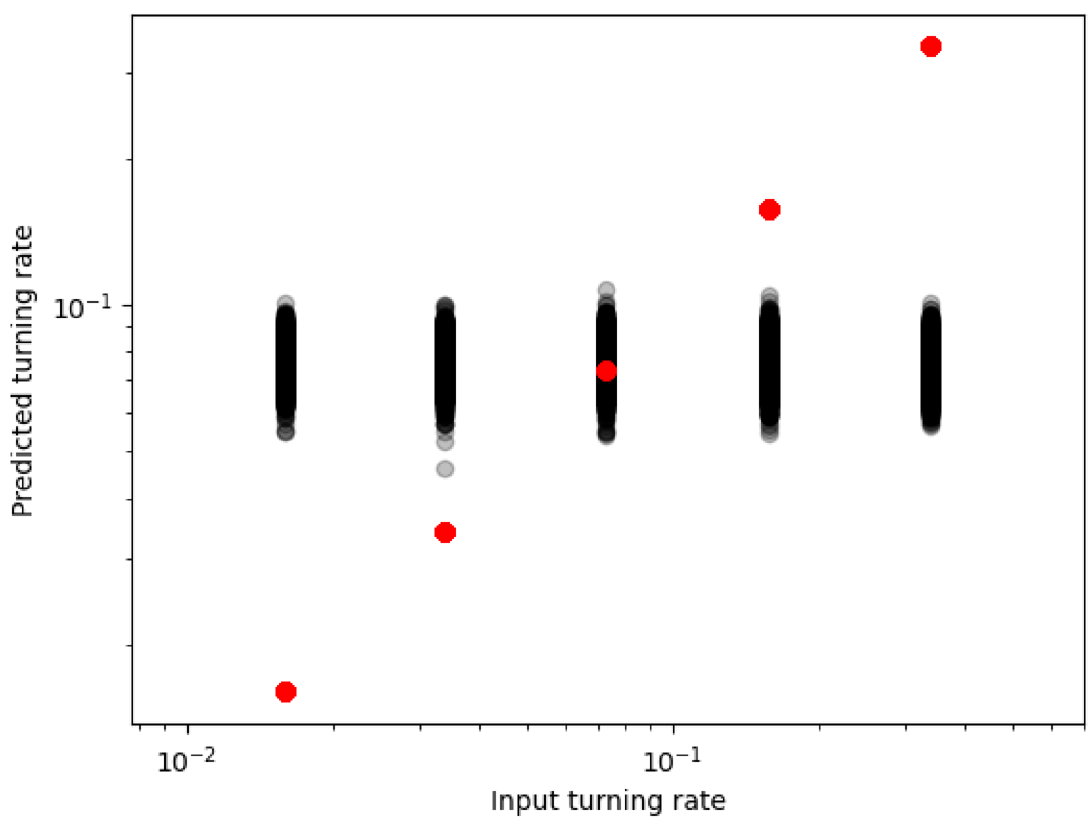
```
prediction = model.predict(x_val, batch_size=64)
print("Shape of prediction : ", np.shape(prediction))

plt.plot(y_val, prediction.T[0], 'o', c='k', alpha=0.25)
plt.plot(y_val, y_val, 'o', color='r')

print("Pearson's correlation coeff: ", pearsonr(y_val, prediction.T[0]).statistic)
plt.xlabel("Input turning rate")
plt.ylabel("Predicted turning rate")
plt.axis("equal")
plt.xscale("log")
plt.yscale("log")
```

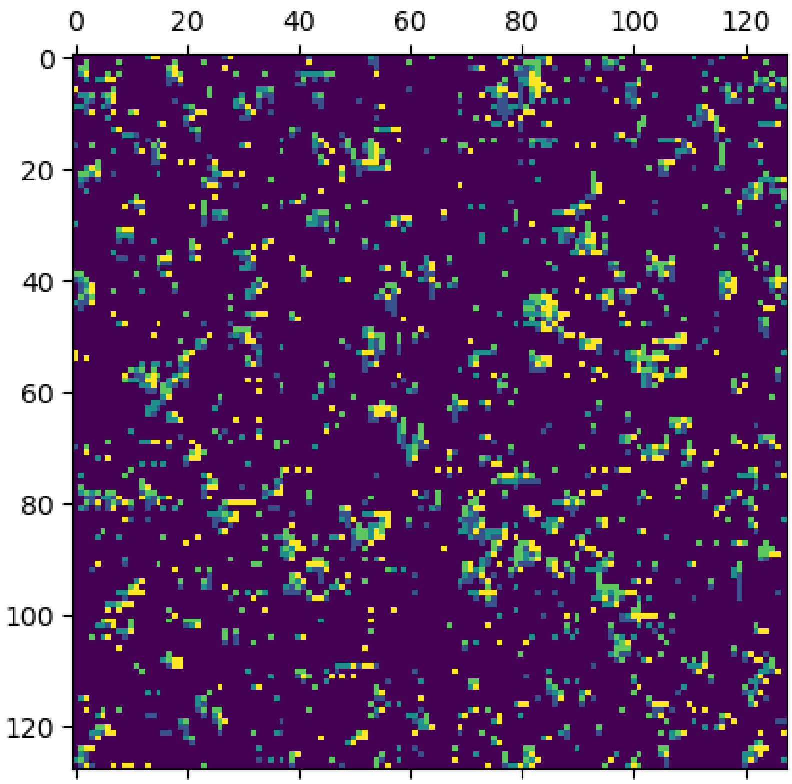
```
2024-02-14 14:12:29.587687: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation o
```

```
313/313 [==============================] - 8s 25ms/step
Shape of prediction :  (20000, 1)
Pearson's correlation coeff:  -0.04497726280558032
```




```
demo_idx = 100
plt.matshow(x_val[demo_idx])
print("Actual: ", y_val[demo_idx])
print("Predicted: ", prediction.T[0][demo_idx])
```

```
Actual:  0.034
Predicted:  0.07371252
```



We can play with the architecture and see how the untrained predictions can change too.

Run the training

```
history = model.fit(
    x_train,
    y_train,
    epochs=10,
    verbose=True,
    batch_size=64,
    validation_data=(x_val, y_val)
)
```

```
Epoch 1/10
781/782 [============================>.] - ETA: 0s - loss: 0.0456 - accuracy: 0.0000e+00782/7
Epoch 2/10
782/782 [==============================] - 50s 64ms/step - loss: 0.0269 - accuracy: 0.0000e+0
```

```
Epoch 3/10
782/782 [==============================] - 50s 64ms/step - loss: 0.0260 - accuracy: 0.0000e+0
Epoch 4/10
782/782 [==============================] - 37s 48ms/step - loss: 0.0236 - accuracy: 0.0000e+0
Epoch 5/10
782/782 [==============================] - 43s 55ms/step - loss: 0.0221 - accuracy: 0.0000e+0
Epoch 6/10
782/782 [==============================] - 43s 55ms/step - loss: 0.0212 - accuracy: 0.0000e+0
Epoch 7/10
782/782 [==============================] - 43s 55ms/step - loss: 0.0208 - accuracy: 0.0000e+0
Epoch 8/10
782/782 [==============================] - 43s 55ms/step - loss: 0.0204 - accuracy: 0.0000e+0
Epoch 9/10
782/782 [==============================] - 49s 63ms/step - loss: 0.0199 - accuracy: 0.0000e+0
Epoch 10/10
782/782 [==============================] - 69s 88ms/step - loss: 0.0196 - accuracy: 0.0000e+0
```

```
2024-02-14 14:13:58.918844: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation o
2024-02-14 14:14:55.119286: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation o
```

```
print("Evaluate on test data:")
results = model.evaluate(x_val, y_val, batch_size=64, verbose=0)
print("Test loss:", results[0])
print("Test accuracy:", results[1])
```

```
Evaluate on test data:
Test loss: 0.01768036000430584
Test accuracy: 0.0
```

```
2024-02-14 14:22:51.483660: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation o
```

Analyse training results

```
prediction = model.predict(x_val, batch_size=64)
print("Shape of prediction : ", np.shape(prediction))
```

```
2024-02-14 14:23:08.376226: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation o
```

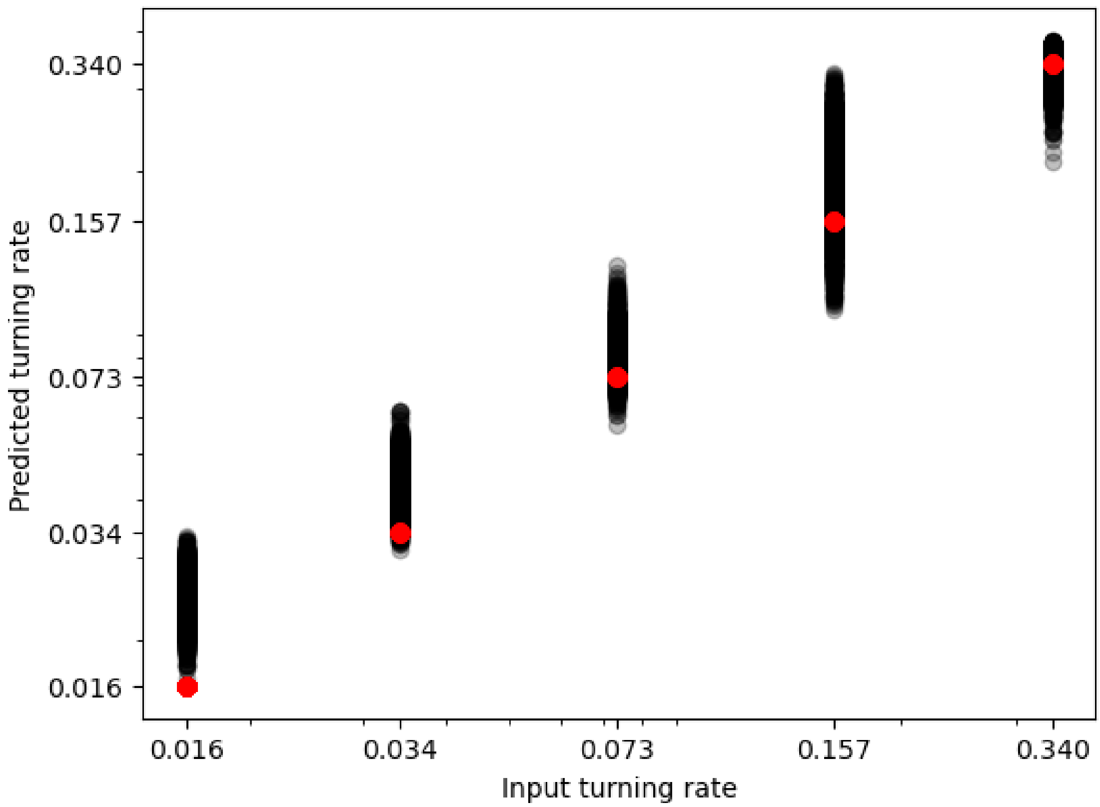
```
313/313 [==============================] - 10s 32ms/step
Shape of prediction :  (20000, 1)
```

```python
fig, ax = plt.subplots()

ax.plot(y_val, prediction.T[0], 'o', c='k', alpha=0.25)
ax.plot(y_val, y_val, 'o', color='r')

print("Pearson's correlation coeff: ", pearsonr(y_val, prediction.T[0]).statistic)
ax.set_xlabel("Input turning rate")
ax.set_ylabel("Predicted turning rate")
#ax.set_aspect("equal")
ax.set_xscale("log")
ax.set_yscale("log")
ax.get_xaxis().set_major_formatter(ticker.ScalarFormatter())
ax.get_yaxis().set_major_formatter(ticker.ScalarFormatter())
ax.set_xticks(np.unique(y))
ax.set_yticks(np.unique(y))
plt.show()
print(np.unique(y))
```

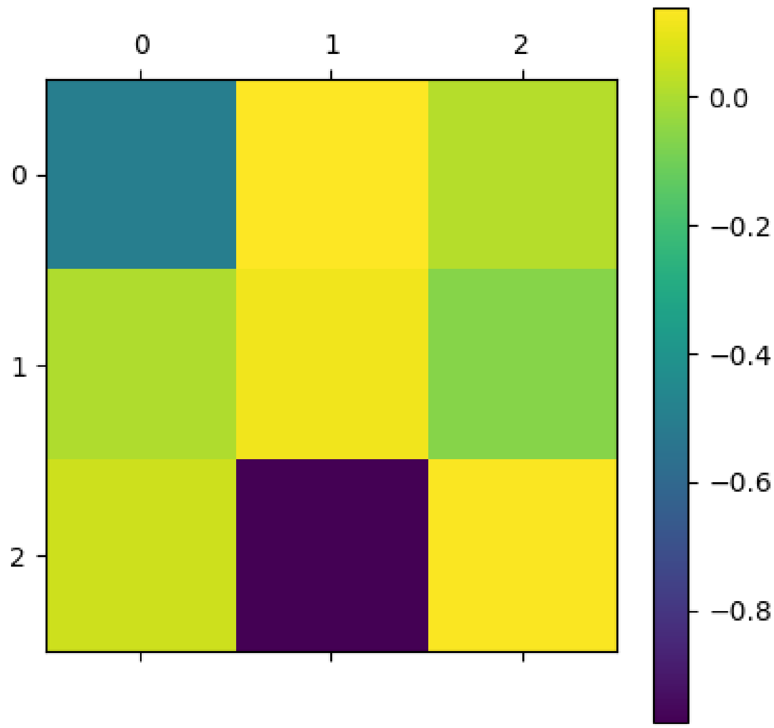
```
Pearson's correlation coeff:  0.9788006506188127
[0.016 0.034 0.073 0.157 0.34 ]
```

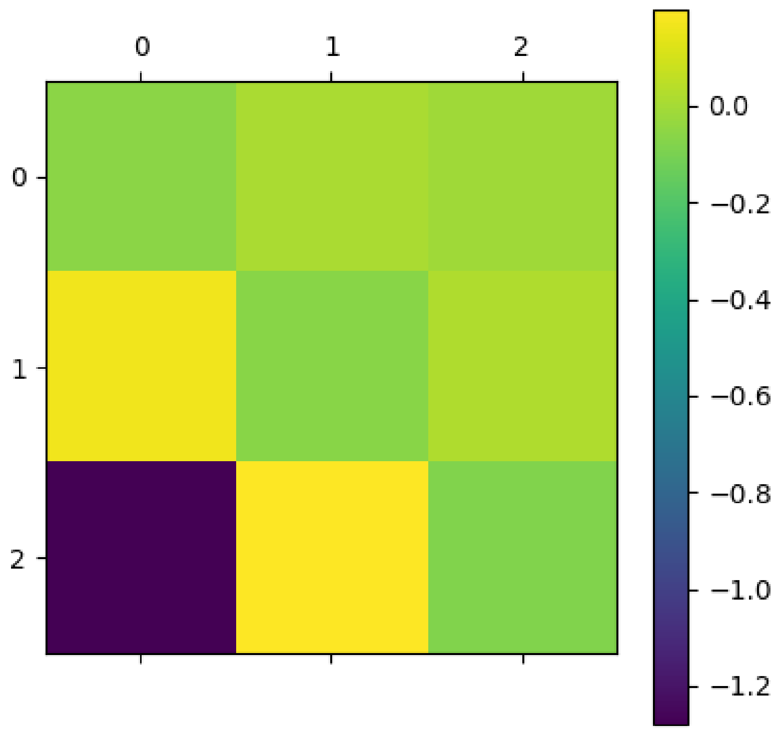


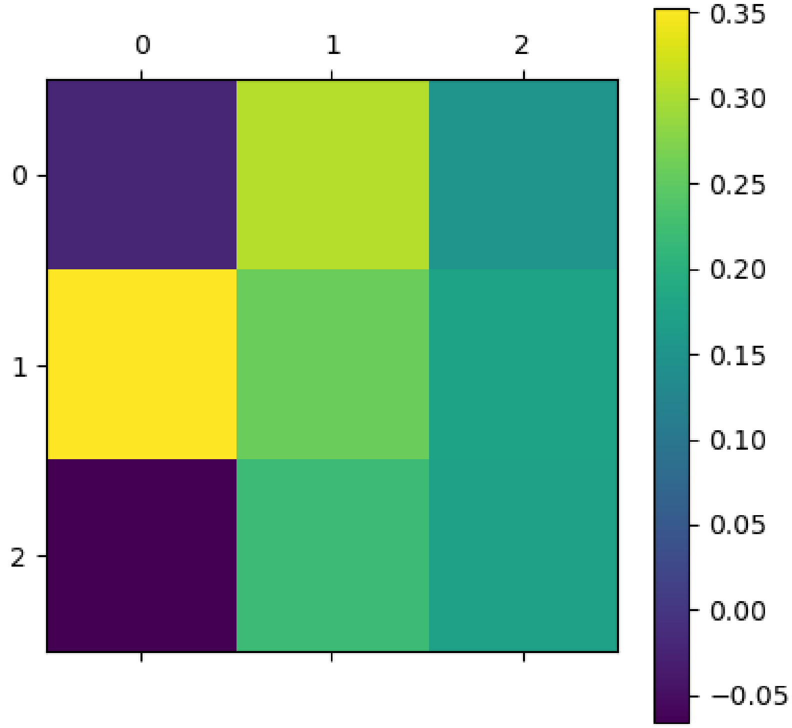
Kernel analysis

```
filters, biases = model.layers[0].get_weights()
print (filters.shape[-1])
for k in range(filters.shape[-1]):
  f = filters[:, :, :, k]
  plt.matshow(f.squeeze())
  plt.colorbar()
```

3


0
1
2
0
1
2
0.0
−0.2
−0.4
−0.6
−0.8


0
1
2
0
1
2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−1.2



Save model (if needed)

```
model.save("../models/cp_14feb_prelim.keras")
```